Донецкий Национальный Технический Университет

Лабораторная работа № 2

«Знакомство с ключевыми моментами работы с GPGPU

на базе технологии Nvidia Cuda»

Выполнил:

ст. группы ИПОИм -17

Лысенко А. С.

Проверила:

доцент каф. ПМИ

Назарова И. А.

Покровск 2017

**Задание к лабораторной работе**

1. Реализовать программу, которая описана в пункте 4 теоретической части

2. Написать аналогичную программу, со следующими отличиями:

a. Предусмотреть возможности получения векторов для расчета из файла или их случайной генерации

b. Вариант получения векторов и вариант вывода результата расчета должен задаваться аргументом командной строки

c. Реализовать поддержку векторов большего размера, чем размер блока грида

3. Модифицировать написанную во втором пункте программу для сравнения временных характеристик в зависимости от размера векторов. Имеются в виду показатели времени выполнения суммирования на CPU, чистого времени суммирования на GPU и времени с передачей данных в GPU. Сделать выводы с полученными результатами.

4. Реализовать программу, осуществляющую составления двух матриц в которой выставляются требования аналогичны указанным во втором пункте. А также выполнить оценку аналогичную описанной в третьем пункте.

2. а)

Пример кода получения вектора для расчёта из файла

Запись в файл

ofstream f("C:\\Users\\User\\Documents\\Visual Studio 2010\\Projects\\CudaLab1\\TestAddVector\\file0.txt");

// Записываем значения в файл

for (int i = 0; i < SIZE; i++)

{

f << i << endl;

vec1[i] = vec2[i] = i;

}

Чтение из файла

ifstream f("C:\\Users\\User\\Documents\\Visual Studio 2010\\Projects\\CudaLab1\\TestAddVector\\file0.txt");

// Чтение значений из файла

for (int i = 0; i < SIZE; i++)

{

f >> vec1[i]; // обрахунок з файлу

vec2[i] = vec1[i]; // обрахунок з файлу

}

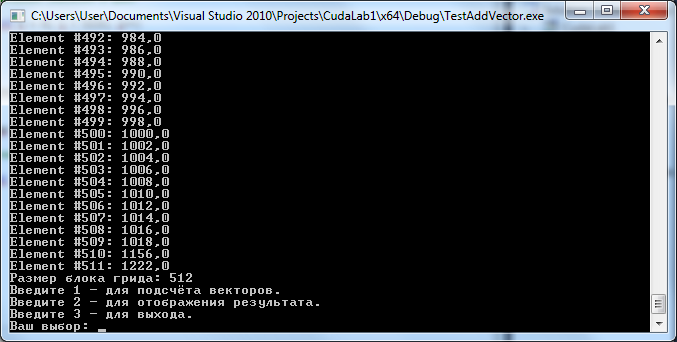


Рис. 1 – Чтение вектора из файла

Пример кода получения вектора при случайной генерации

for (int i = 0; i < SIZE; i++)

{

vec1[i] = rand(); // випадкова генерація

vec2[i] = vec1[i]; // випадкова генерація

}

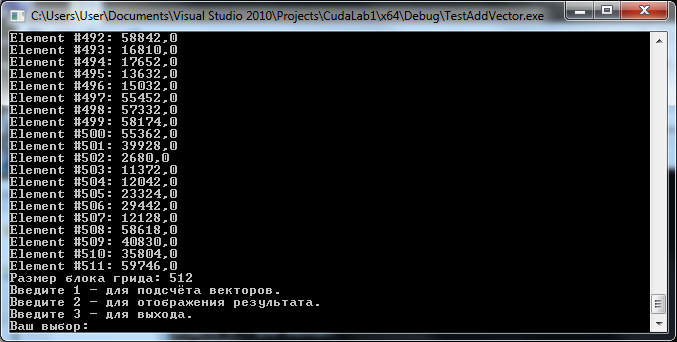


Рис. 2 – Случайная генерация вектора

б) Вариант получения векторов и вариант вывода результата расчета задаётся аргументом командной строки

int main()

{

int Size;

setlocale(LC\_ALL, "Rus");

int n;

cout << "Введите 1 - для подсчёта векторов." << endl;

cout << "Введите 2 - для отображения результата." << endl;

cout << "Введите 3 - для выхода." << endl;

cout << "Ваш выбор: ";

cin >> n;

if(n == 1)

{

Size = computingVectors();

computingVectors();

cout << "Подсчёт окончен." << endl;

}

if(n == 2)

showVectors(Size);

if(n == 3)

return 0;

do

{

if(n!= 1)

{

cout << "Введите 1 - для подсчёта векторов." << endl;

cout << "Введите 2 - для отображения результата." << endl;

cout << "Введите 3 - для выхода." << endl;

cout << "Ваш выбор: ";

}

else if ( n == 1)

{

cout << "Введите 1 - для подсчёта векторов." << endl;

cout << "Введите 2 - для отображения результата." << endl;

cout << "Введите 3 - для выхода." << endl;

cout << "Ваш выбор: ";

}

cin >> n;

if(n == 1)

{

computingVectors();

cout << "Подсчёт окончен." << endl;

}

if(n == 2)

showVectors(Size);

} while(n == 1 || n == 2);

if(n == 3)

delete[] vec1, vec2, vec3;

\_getch();

return 0;

}

Где функции,

void computingVectors() – вычисление вектора

void showVectors(int Size) – вывод результата расчёта, где Size – размер блока.

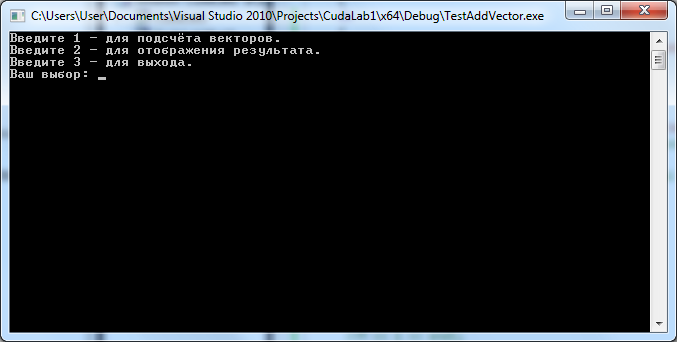


Рис. 3 – Вариант получения векторов и вариант вывода результата расчета задаётся аргументом командной строки

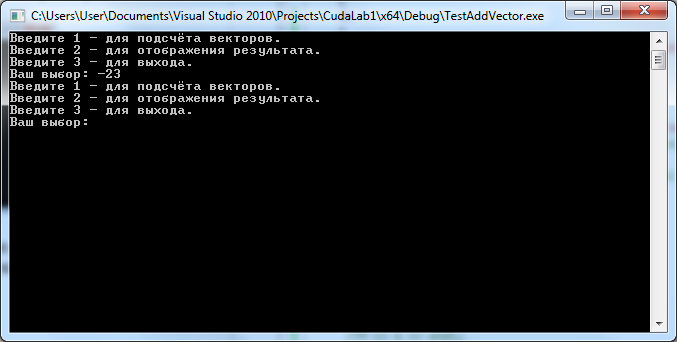


Рис. 4 – Вариант получения векторов и вариант вывода результата расчета с ошибочным значением

с) Поддержка векторов большего размера, чем размер блока грида

if(f == false)

{

do

{

cout << "Введите размер блока грида: (0 - 1024): " ;

cin >> Size;

} while(Size < 0 || Size > 1024);

f = true;

}

Использования значения в ядре

addVector<<<Size, SIZE>>>(devVec1, devVec2, devVec3);

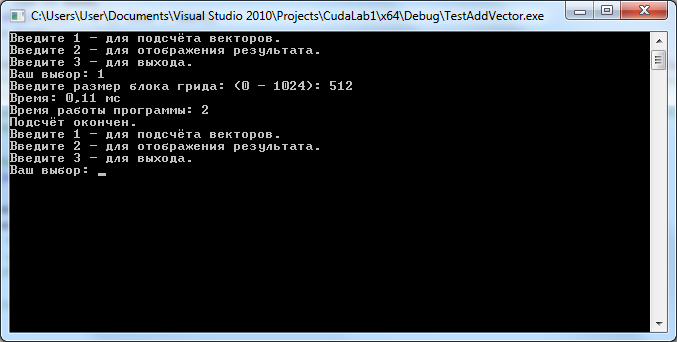


Рис. 5 – Поддержка векторов большего размера, чем размер блока грида

3) Для сравнении значений на CPU буду использовать встроенную функцию с++ clock( );

Пример работы в программе

// Покажчики на пам'ять відеокарти

float \*devVec1, \*devVec2, \*devVec3;

// Виділяємо пам'ять для векторів на відеокарті

cudaMalloc((void\*\*)&devVec1, sizeof(float) \* SIZE);

cudaMalloc((void\*\*)&devVec2, sizeof(float) \* SIZE);

cudaMalloc((void\*\*)&devVec3, sizeof(float) \* SIZE);

// Копіюємо дані в пам'ять відеокарти

cudaMemcpy(devVec1, vec1, sizeof(float) \* SIZE, cudaMemcpyHostToDevice);

cudaMemcpy(devVec2, vec2, sizeof(float) \* SIZE, cudaMemcpyHostToDevice);

unsigned int start\_time = clock();

addVector<<<Size, SIZE>>>(devVec1, devVec2, devVec3);

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

if( b == false)

{

cout << "Время работы программы: " << search\_time << endl;

b = true;

}

Для сравнении значений на GPU буду использовать встроенную функцию cuda cudaEventRecord (eventName, 0), cudaEventElapsedTime(&gpuTime, start, stop);

eventName start, stop;

cudaEvent\_t start , stop;

cudaEventCreate (&start);

cudaEventCreate (&stop);

float gpuTime;

// Покажчики на пам'ять відеокарти

float \*devVec1, \*devVec2, \*devVec3;

// Виділяємо пам'ять для векторів на відеокарті

cudaMalloc((void\*\*)&devVec1, sizeof(float) \* SIZE);

cudaMalloc((void\*\*)&devVec2, sizeof(float) \* SIZE);

cudaMalloc((void\*\*)&devVec3, sizeof(float) \* SIZE);

// Копіюємо дані в пам'ять відеокарти

cudaMemcpy(devVec1, vec1, sizeof(float) \* SIZE, cudaMemcpyHostToDevice);

cudaMemcpy(devVec2, vec2, sizeof(float) \* SIZE, cudaMemcpyHostToDevice);

cudaEventRecord (start, 0 );

addVector<<<Size, SIZE>>>(devVec1, devVec2, devVec3);

cudaEventRecord (stop , 0 );

// Обробник події

cudaEvent\_t syncEvent;

// Створюємо подію

cudaEventCreate(&syncEvent);

// Записуємо подію

cudaEventRecord(syncEvent, 0);

// Синхронізуємо подію

cudaEventSynchronize(syncEvent);

cudaEventSynchronize(stop);

// Тільки тепер отримуємо результат розрахунку

cudaMemcpy(vec3, devVec3, sizeof(float) \* SIZE, cudaMemcpyDeviceToHost);

cudaEventElapsedTime(&gpuTime, start, stop);

if( d == false)

{

printf("Время: %.2f мс\n", gpuTime);

d = true;

}

Вывод выполнение вычислений на GPU в 10 раз быстрее чем на CPU

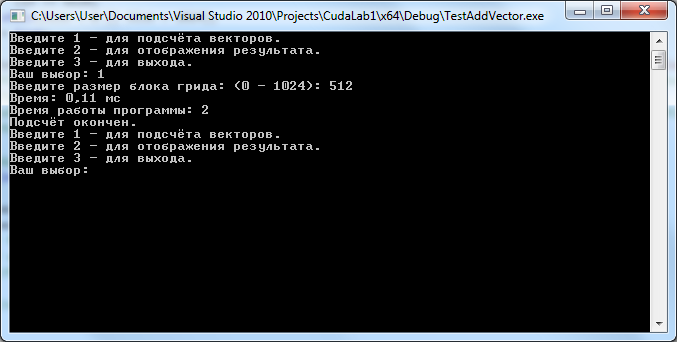
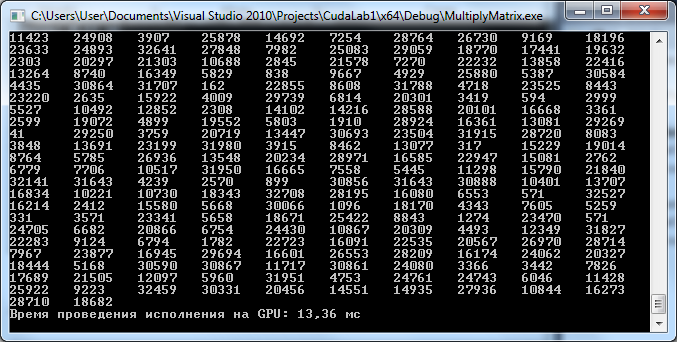


Рис. 5 – Время работы программы на CPU и GPU

4) Реализовать программу, осуществляющую сложение двух матриц



Листинг программы на С++

Сложение векторов

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <conio.h>

#include <iostream>

#include <fstream>

#include <locale>

#include <ctime>

#define SIZE 512

using namespace std;

// Функція складання двох векторів

\_\_global\_\_ void addVector(float\* left, float\* right, float\* result)

{

// Отримати id поточної нитки

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

if(idx >= 1024) return;

// Розраховуємо результат.

result[idx] = left[idx] + right[idx];

}

// Виділяємо пам'ять під вектора

float \*vec1 = new float[SIZE], \*vec2 = new float[SIZE], \*vec3 = new float[SIZE];

bool f = false;

bool b = false;

bool d = false;

int Size;

float gpuTime;

int computingVectors()

{

cudaEvent\_t start , stop;

cudaEventCreate (&start);

cudaEventCreate (&stop);

if(f == false)

{

do

{

cout << "Введите размер блока грида: (0 - 1024): " ;

cin >> Size;

} while(Size < 0 || Size > 1024);

f = true;

}

ifstream f("C:\\Users\\User\\Documents\\Visual Studio 2010\\Projects\\CudaLab1\\TestAddVector\\file0.txt");

//ofstream f("C:\\Users\\User\\Documents\\Visual Studio 2010\\Projects\\CudaLab1\\TestAddVector\\file0.txt");

//fclose(f);

// Ініціалізіруем значення векторів

for (int i = 0; i < SIZE; i++)

{

vec1[i] = rand(); // випадкова генерація

vec2[i] = vec1[i]; // випадкова генерація

// f >> vec1[i]; // обрахунок з файлу

// vec2[i] = vec1[i]; // обрахунок з файлу

//f << i << endl;

//vec1[i] = vec2[i] = i;

}

// Покажчики на пам'ять відеокарти

float \*devVec1, \*devVec2, \*devVec3;

// Виділяємо пам'ять для векторів на відеокарті

cudaMalloc((void\*\*)&devVec1, sizeof(float) \* SIZE);

cudaMalloc((void\*\*)&devVec2, sizeof(float) \* SIZE);

cudaMalloc((void\*\*)&devVec3, sizeof(float) \* SIZE);

// Копіюємо дані в пам'ять відеокарти

cudaMemcpy(devVec1, vec1, sizeof(float) \* SIZE, cudaMemcpyHostToDevice);

cudaMemcpy(devVec2, vec2, sizeof(float) \* SIZE, cudaMemcpyHostToDevice);

unsigned int start\_time = clock();

cudaEventRecord (start, 0 );

addVector<<<Size, SIZE>>>(devVec1, devVec2, devVec3);

cudaEventRecord (stop , 0 );

// Обробник події

cudaEvent\_t syncEvent;

// Створюємо подію

cudaEventCreate(&syncEvent);

// Записуємо подію

cudaEventRecord(syncEvent, 0);

// Синхронізуємо подію

cudaEventSynchronize(syncEvent);

cudaEventSynchronize(stop);

// Тільки тепер отримуємо результат розрахунку

cudaMemcpy(vec3, devVec3, sizeof(float) \* SIZE, cudaMemcpyDeviceToHost);

cudaEventElapsedTime(&gpuTime, start, stop);

if( d == false)

{

printf("Время: %.2f мс\n", gpuTime);

d = true;

}

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

if( b == false)

{

cout << "Время работы программы: " << search\_time << endl;

b = true;

}

// Вивільняються ресурсів

// Видалення події

cudaEventDestroy(syncEvent);

// Вивільнення пам'яті на відеокарті

cudaFree(devVec1);

cudaFree(devVec2);

cudaFree(devVec3);

// Вивільнення памьяті основної програми

return Size;

}

void showVectors(int Size)

{

// Результати розрахунку

for (int i = 0; i < SIZE; i++) printf("Element #%i: %.1f\n", i, vec3[i]);

cout << "Размер блока грида: " << Size << endl;

}

int main()

{

int Size;

setlocale(LC\_ALL, "Rus");

int n;

cout << "Введите 1 - для подсчёта векторов." << endl;

cout << "Введите 2 - для отображения результата." << endl;

cout << "Введите 3 - для выхода." << endl;

cout << "Ваш выбор: ";

cin >> n;

if(n == 1)

{

Size = computingVectors();

computingVectors();

cout << "Подсчёт окончен." << endl;

//cout << "Введите 2 - для отображения результата." << endl;

}

if(n == 2)

showVectors(Size);

if(n == 3)

return 0;

do

{

if(n!= 1)

{

cout << "Введите 1 - для подсчёта векторов." << endl;

cout << "Введите 2 - для отображения результата." << endl;

cout << "Введите 3 - для выхода." << endl;

cout << "Ваш выбор: ";

}

else if ( n == 1)

{

cout << "Введите 1 - для подсчёта векторов." << endl;

cout << "Введите 2 - для отображения результата." << endl;

cout << "Введите 3 - для выхода." << endl;

cout << "Ваш выбор: ";

}

cin >> n;

if(n == 1)

{

computingVectors();

cout << "Подсчёт окончен." << endl;

}

if(n == 2)

showVectors(Size);

} while(n == 1 || n == 2);

if(n == 3)

delete[] vec1, vec2, vec3;

\_getch();

return 0;

}

Листинг программы на С++

Сложение матриц

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <iostream>

#include <iomanip>

#include <clocale>

using namespace std;

#define BLOCK\_SIZE 16 // submatrix size

#define N 256 // matrix size is N\*N

\_\_global\_\_ void matMult ( float \* a, float \* b, int n, float \* c )

{

int bx = blockIdx.x; // block index

int by = blockIdx.y;

int tx = threadIdx.x; // thread index

int ty = threadIdx.y;

float sum = 0.0f; // computed subelement

int ia = n \* BLOCK\_SIZE \* by + n \* ty; // a [i][0]

int ib = BLOCK\_SIZE \* bx + tx;

// Multiply the two matrices together;

for ( int k = 0; k < n; k++ )

{

sum += a [ia + k] + b [ib + k\*n];

}

/\*for(int k = 0; k < n; k++)

{

printf("%.2f\n", a[ia + k]);

printf("%.2f\n", b[ib + k\*n]);

}\*/

// Write the block sub-matrix to global memory;

// each thread writes one element

int ic = n \* BLOCK\_SIZE \* by + BLOCK\_SIZE \* bx;

c [ic + n \* ty + tx] = sum;

}

int main ( int argc, char \* argv [] )

{

setlocale(LC\_ALL, "Rus");

int numBytes = N \* N \* sizeof ( float );

// allocate host memory

float \* a = new float [N\*N];

float \* b = new float [N\*N];

float \* c = new float [N\*N];

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

for ( int j = 0; j < N; j++ )

{

int k = N\*i + j;

a [k] = rand();

b [k] = rand();

}

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

for ( int j = 0; j < N; j++ )

{

int k = N\*i + j;

//cout << setw(3) << a[k] << b[k];

printf("%.0f\t", a[k]);

printf("%.0f\t", b[k]);

}

printf("\n");

//cout << endl;

// allocate device memory

float \* adev = NULL;

float \* bdev = NULL;

float \* cdev = NULL;

cudaMalloc ( (void\*\*)&adev, numBytes );

cudaMalloc ( (void\*\*)&bdev, numBytes );

cudaMalloc ( (void\*\*)&cdev, numBytes );

// set kernel launch configuration

dim3 threads ( BLOCK\_SIZE, BLOCK\_SIZE );

dim3 blocks ( N / threads.x, N / threads.y);

// create cuda event handles

cudaEvent\_t start, stop;

float gpuTime = 0.0f;

cudaEventCreate (&start);

cudaEventCreate (&stop);

// asynchronously issue work to the GPU (all to stream 0)

cudaEventRecord (start, 0);

cudaMemcpy (adev, a, numBytes, cudaMemcpyHostToDevice );

cudaMemcpy (bdev, b, numBytes, cudaMemcpyHostToDevice );

matMult<<<blocks, threads>>> (adev, bdev, N, cdev);

cudaMemcpy (c, cdev, numBytes, cudaMemcpyDeviceToHost );

cudaEventRecord (stop, 0);

cudaEventSynchronize (stop);

cudaEventElapsedTime (&gpuTime, start, stop);

// print the cpu and gpu times

printf("Время проведения исполнения на GPU: %.2f мс\n", gpuTime);

// release resources

cudaEventDestroy (start);

cudaEventDestroy (stop);

cudaFree (adev);

cudaFree (bdev);

cudaFree (cdev);

delete a;

delete b;

delete c;

\_getch();

return 0;

}