­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­



Лабораторна робота №4

з курсу «**Технології розподілених систем та паралельних обчислень**»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 11

Виконав студент гр. КНз-41

Чалий Михaйло

Перевірив

Фармага І.В.

­­

Львів 2016

## Мета роботи

Метою роботи є отримання навиків реалізації паралельних обчислень в гетерогенних обчислювальних середовищах з допомогою технології OpenCL.

## Теоретична частина

Технологія OpenCL (Open Computing Language) від Khronos Group є однією з технологій високопродуктивних паралельних та розподілених обчислень. Вона дає змогу реалізовувати концепцію паралелізму даних на широкому колі обчислювальних пристроїв, зокрема на центральних процесорах (Central Processing Unit – CPU), на графічних картах, що підтримують інтерфейси загального використання (General-Purpose Graphics Processing Unit – GPGPU), на програмованих користувачем вентильних матрицях (Field-Programmable Gate Array – FPGA), та на процесорах цифрової обробки сигналів (Digital Signal Processor – DSP). Ці класи обчислювальних пристроїв мають SIMD (Single Instruction Multiple Data) архітектуру і дають змогу максимально ефективно виконувати запрограмовані алгоритми, що використовують декомпозицію даних великих задач. На відміну від аналогічних (основні: NVIDIA CUDA, Microsoft DirectCompute, AMD FireStream), технологія OpenCL підтримується максимально широким колом виробників обчислювальних пристроїв.

Технологія OpenCL замислювалася як індустріальний стандарт для створення програмних додатків, які могли б виконуватися в гетерогенному обчислювальному середовищі. Більше того, вона розроблена так, щоб забезпечувати комфортну роботу з такими пристроями, які знаходяться тільки в планах розробки і навіть з тими, які ще ніхто не придумав. Для координації роботи всіх цих пристроїв в гетерогенній системі виділяють один керуючий пристрій, що взаємодіє з рештою засобами OpenCL API (Application Programming Interface). Такий пристрій називається "*хост*" (англ. "*host*"), він визначається поза OpenCL.

Технологія виходить з найбільш загальних передумов, що дають уявлення про пристрій з підтримкою OpenCL: так як цей пристрій передбачається використовувати для обчислень – в ньому є якийсь "процесор" в загальному розумінні цього слова. Щось, що може виконувати команди. Так як OpenCL створений для паралельних обчислень, то такий процесор може мати засоби паралелізму всередині себе, наприклад, кілька фізичних чи логічних ядер. Також елементарним способом нарощування продуктивності паралельних обчислень є установка декількох таких процесорів на пристрої. В гетерогенній системі може бути декілька OpenCL пристроїв, у загальному випадку, з різною архітектурою. Крім обчислювальних ресурсів пристрій має якийсь обсяг пам'яті. Ніяких особливих вимог до цієї пам'яті не пред'являється – вона може бути розміщена як на пристрої, так і на хості, наприклад, як це зроблено для вбудованих відеокарт. Особливості доступу до пам’яті не визначаються стандартом OpenCL і залежать від конкретних реалізацій.

## Завдання

Скласти програму з використанням технології OpenCL, що реалізує індивідуальне завдання (ваш варіант) до лабораторної роботи №3 "Використання функціональної декомпозиції для розв’язку обчислювальних задач", застосовуючи декомпозицію даних задачі. Провести аналіз результатів виконання програми. Порівняти результати з отриманими у попередніх роботах.

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | матриця | | |
| bi=11i2 для парних і  bi=11/i для непарних і | A1(b1-2c1) | A2(B2-2C2)  Cij =1/(i2+j) |

## Реалізація

Лістінг Program.cs

using OpenCL.Net.Extensions;

using OpenCL.Net;

using System;

using System.Linq;

namespace L4OpenCL

{

class Program

{

private const int ArrayLength = 1024;

static readonly int n = 10;

static readonly Random rnd = new Random();

static float[] Source(int n, Func<int, float> init) => Enumerable.Range(0, n).Select(init).ToArray();

static float[] SourceZeroMatrix() => Source(n \* n, \_ => 0f);

static float[] SourceZeroVector() => Source(n, \_ => 0f);

static float[] SourceRndMatrix() => Source(n \* n, \_ => (float)rnd.NextDouble());

static float[] SourceRndVector() => Source(n, \_ => (float)rnd.NextDouble());

//static float[] SourceMatrix(Func<int, int, float> init) => Source(n \* n, i => init(i%n, i));

static float[] Transpose(OpenCL.Net.Environment env, float[] A, int w, int h)

{

var len = (uint)A.Length;

var bufferA = env.Context.CreateBuffer(A, MemFlags.ReadOnly);

var bufferB = env.Context.CreateBuffer(Source(A.Length, \_ => 0f), MemFlags.WriteOnly);

var kernel = new Kernel.Transpose(env.Context);

kernel.Compile("-cl-opt-disable");

kernel.Run(env.CommandQueues[0], bufferB, bufferA, w, h, len);

// WTF: We should wait for run

var results = new float[A.Length];

env.CommandQueues[0].ReadFromBuffer(bufferB, results);

return results;

}

static float[] AddMatrix(OpenCL.Net.Environment env, float[] A, float[] B)

{

var len = (uint)A.Length;

var bufferA = env.Context.CreateBuffer(A, MemFlags.ReadOnly);

var bufferB = env.Context.CreateBuffer(B, MemFlags.ReadOnly);

var bufferC = env.Context.CreateBuffer(Source(A.Length, \_ => 0f), MemFlags.WriteOnly);

var kernel = new Kernel.AddMatrix(env.Context);

kernel.Compile("-cl-opt-disable");

kernel.Run(env.CommandQueues[0], bufferC, bufferA, bufferB, len);

// WTF: We should wait for run

var results = new float[A.Length];

env.CommandQueues[0].ReadFromBuffer(bufferC, results);

return results;

}

static float[] SubstractMatrix(OpenCL.Net.Environment env, float[] A, float[] B)

{

var len = (uint)A.Length;

var bufferA = env.Context.CreateBuffer(A, MemFlags.ReadOnly);

var bufferB = env.Context.CreateBuffer(B, MemFlags.ReadOnly);

var bufferC = env.Context.CreateBuffer(Source(A.Length, \_ => 0f), MemFlags.WriteOnly);

var kernel = new Kernel.SubstractMatrix(env.Context);

kernel.Compile("-cl-opt-disable");

kernel.Run(env.CommandQueues[0], bufferC, bufferA, bufferB, len);

// WTF: We should wait for run

var results = new float[A.Length];

env.CommandQueues[0].ReadFromBuffer(bufferC, results);

return results;

}

static float[] MultiplyScalar(OpenCL.Net.Environment env, float[] A, float scalar)

{

var len = (uint)A.Length;

var bufferA = env.Context.CreateBuffer(A, MemFlags.ReadOnly);

var bufferB = env.Context.CreateBuffer(Source(A.Length, \_ => 0f), MemFlags.WriteOnly);

var kernel = new Kernel.MultiplyScalar(env.Context);

kernel.Compile("-cl-opt-disable");

kernel.Run(env.CommandQueues[0], bufferB, bufferA, scalar, len);

// WTF: We should wait for run

var results = new float[A.Length];

env.CommandQueues[0].ReadFromBuffer(bufferB, results);

return results;

}

static float[] MultiplyMatrix(OpenCL.Net.Environment env, float[] A, float[] B)

{

var len = (uint)A.Length;

var bufferA = env.Context.CreateBuffer(A, MemFlags.ReadOnly);

var bufferB = env.Context.CreateBuffer(B, MemFlags.ReadOnly);

var bufferC = env.Context.CreateBuffer(SourceZeroMatrix(), MemFlags.WriteOnly);

var kernel = new Kernel.MultiplyMatrix(env.Context);

kernel.Compile("-cl-opt-disable");

kernel.Run(env.CommandQueues[0], bufferC, bufferA, bufferB, n, n, len);

// WTF: We should wait for run

var results = new float[A.Length];

env.CommandQueues[0].ReadFromBuffer(bufferC, results);

return results;

}

static float[] MultiplyMatrixVector(OpenCL.Net.Environment env, float[] A, float[] b)

{

var len = (uint)b.Length;

var bufferA = env.Context.CreateBuffer(A, MemFlags.ReadOnly);

var bufferB = env.Context.CreateBuffer(b, MemFlags.ReadOnly);

var bufferC = env.Context.CreateBuffer(SourceZeroVector(), MemFlags.WriteOnly);

var kernel = new Kernel.MultiplyMatrixVector(env.Context);

kernel.Compile("-cl-opt-disable");

kernel.Run(env.CommandQueues[0], bufferC, bufferA, bufferB, n, len);

// WTF: We should wait for run

var results = new float[b.Length];

env.CommandQueues[0].ReadFromBuffer(bufferC, results);

return results;

}

static void DumpMatrix(float[] A)

{

Console.WriteLine(String.Join(",", A));

}

static void Main(string[] args)

{

using (var env = "\*NVidia\*".CreateCLEnvironment())

{

var A = SourceRndMatrix();

var b = Source(n, i => (float)((i % 2 == 0) ? 11.0f \* Math.Pow(i, 2) : 11.0f / i));

var y1 = MultiplyMatrixVector(env, A, b);

Console.WriteLine($"y1: {String.Join(",", y1)}");

var A1 = SourceRndMatrix();

var b1 = SourceRndVector();

var c1 = SourceRndVector();

var c12 = MultiplyScalar(env, c1, 2.0f);

var b12 = SubstractMatrix(env, b1, c12);

var y2 = MultiplyMatrixVector(env, A1, b12);

Console.WriteLine($"y2: {String.Join(",", y2)}");

var A2 = SourceRndMatrix();

var B2 = SourceRndMatrix();

var C2 = SourceRndMatrix(); // (x, y) => (x + y == 0) ? 0.0 : 1 / (Math.Pow(x, 2) + y));

var C22 = SubstractMatrix(env, B2, C2);

var Y3 = MultiplyMatrix(env, A2, C22);

Console.WriteLine($"Y3: {String.Join(",", Y3)}");

//var x1 = y2.Multiply(y1.Transpose());

//var x2 = Y3.Multiply(y2.Transpose().Multiply(y2).At(0, 0));

//var x3 = y1.Transpose().Multiply(Y3.Power(2)).Multiply(y2).At(0, 0);

//var x4 = Y3;

//var x5 = Y3.Multiply(y1).Multiply(y1.Transpose()).Multiply(Y3);

//var R = x1.Add(x2).Add(x3).Add(x4).Add(x5);

//Console.WriteLine($"R: {String.Join(",", R)}");

}

}

}

}

Лістінг Kernel.cl

\_\_kernel void Transpose(\_\_global float\* B, \_\_global float\* A, int w, int h)

{

int tx = get\_global\_id(0);

int ty = get\_global\_id(1);

if (tx < w && yIndex < h)

{

int iin = tx + ww \* ty;

int iout = ty + hh \* tx;

B[iout] = A[iin];

}

}

\_\_kernel void MultiplyScalar(\_\_global float\* B, \_\_global float\* A, float scale)

{

int id = get\_global\_id(0);

B[id] = A[id] \* scale;

}

\_\_kernel void AddMatrix(\_\_global float\* C, \_\_global float\* A, \_\_global float\* B)

{

int id = get\_global\_id(0);

C[id] = A[id] + B[id];

}

\_\_kernel void SubstractMatrix(\_\_global float\* C, \_\_global float\* A, \_\_global float\* B)

{

int id = get\_global\_id(0);

C[id] = A[id] - B[id];

}

\_\_kernel void MultiplyMatrixVector(\_\_global float\* c, \_\_global float\* A, \_\_global float\* b, int wA)

{

int tx = get\_global\_id(0);

float value = 0;

for (unsigned int k = 0; k < wA; ++k) {

value += A[tx \* wA + k] \* b[k];

}

c[tx] = value;

}

\_\_kernel void MultiplyMatrix(\_\_global float\* C, \_\_global float\* A, \_\_global float\* B, int wA, int wB)

{

int tx = get\_global\_id(0);

int ty = get\_global\_id(1);

float value = 0;

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

{

float elementA = A[ty \* wA + k];

float elementB = B[k \* wB + tx];

value += elementA \* elementB;

}

C[ty \* wA + tx] = value;

}

## Результат

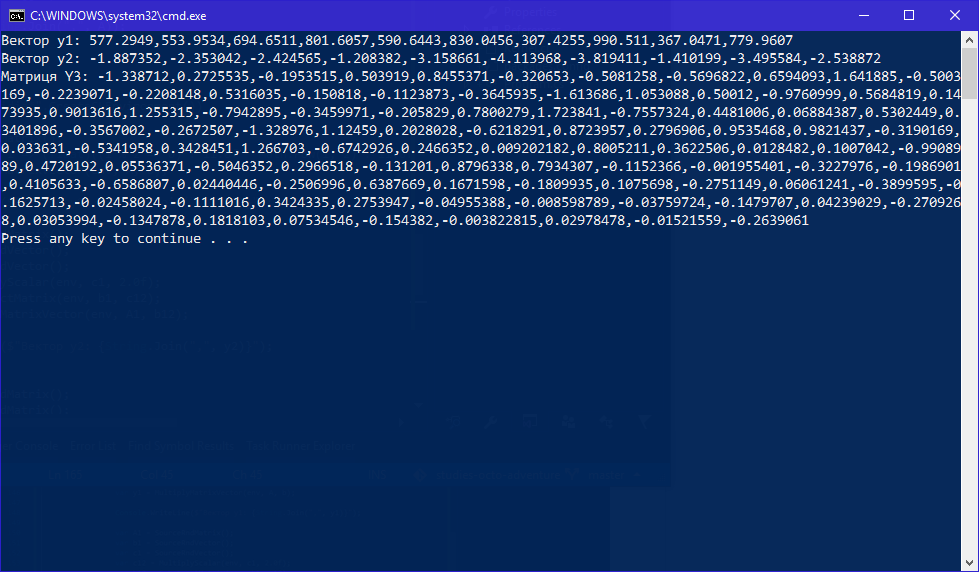


Рис. 1. Результат роботи програми

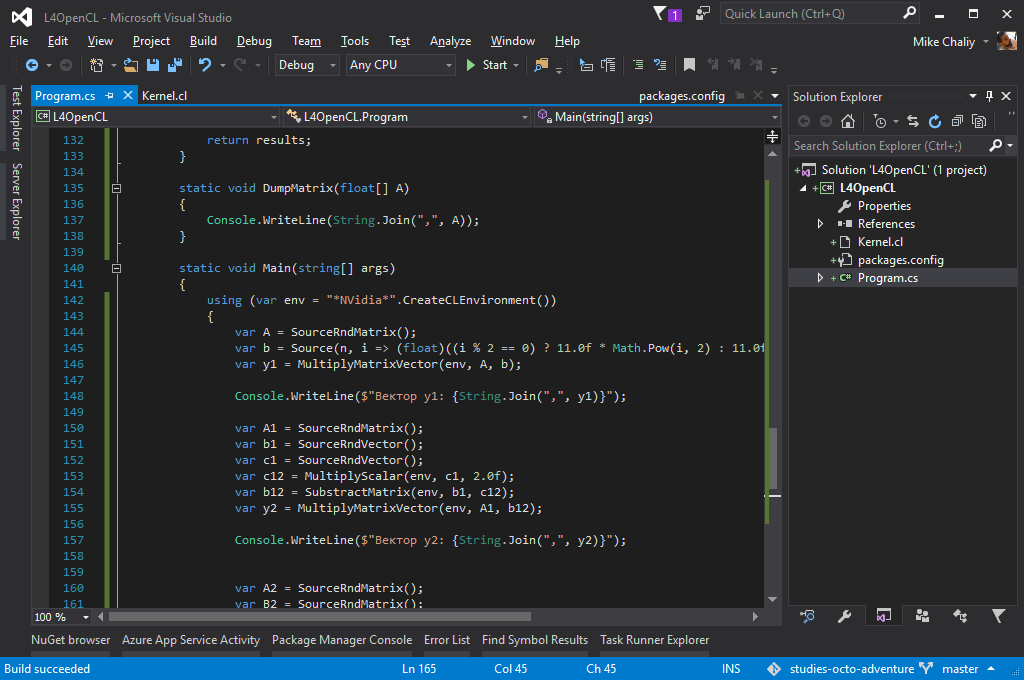


Рис. 2. Код програми в Visual Studio

## Висновки

Отримав навики реалізації паралельних обчислень в гетерогенних обчислювальних середовищах з допомогою технології OpenCL.