Лабораторна робота №16. CUDA-технології обчислення арифметичних виразів

МЕТА РОБОТИ: ознайомитись з основними компонентами інтегрованого середовища Visual Studio CUDA C 2012, навчитись створювати та реалізовувати проекти програм.

1.1. Програма роботи

1.1.1. Отримати завдання.

1.1.2. Написати програми відповідних класів, основну та відповідні допоміжні функції, згідно з вказівками до виконання роботи.

1.1.3. Підготувати власні коректні вхідні дані (вказати їх формат і значення) і проаналізувати їх.

1.1.4. Оформити електронний звіт про роботу та захистити її.

1.2. Вказівки до виконання роботи

1.2.1. Студент, згідно з індивідуальним номером, вибирає своє завдання з розділу 1.5 і записує його до звіту.

1.2.2. Оголошення класу (структури), основну та відповідні допоміжні функції необхідно запрограмувати так, як це показано у розд. 1.4.

1.2.3. Власні вхідні дані мають бути коректними, знаходитися в розумних межах і відповідати тим умовам, які стосуються індивідуального завдання.

1.2.4. Звіт має містити такі розділи:

- мету роботи та завдання з записаною умовою задачі;
- коди всіх використовуваних .cu файлів, а також пояснення до них;
- результати реалізації програми, які виведені на консоль;
- висновки, в яких наводиться призначення програми, обмеження на її застосування і можливі варіанти удосконалення, якщо такі є.

1.3.Теоретичні відомості

CUDA (англ. *Compute Unified Device Architecture*) — технологія GPGPU (англ. General-purpose computing on Graphics Processing Units), що дозволяє програмістам реалізовувати мовою програмування C алгоритми, що виконуватимуться на графічних процесорах Geforce восьмого покоління і вище (Geforce 8 Series, Geforce 9 Series, Geforce 200 Series), Nvidia Quadro і Tesla компанії Nvidia. Технологія CUDA розроблена компанією Nvidia. Технологія CUDA — це середовище розробки на C, яка дозволяє програмістам і розробникам писати програмне забезпечення для вирішення складних обчислювальних завдань за менший час завдяки багатоядерній обчислювальній потужності графічних процесорів. Простіше кажучи, графічна підсистема комп'ютера з підтримкою CUDA може бути використана, як обчислювальна. CUDA дає розробникові можливість на свій розсуд організовувати доступ до

набору інструкцій графічного прискорювача і управляти його пам'яттю, організовувати на ньому складні паралельні обчислення. Графічний процесор з підтримкою CUDA стає потужною програмованою відкритою архітектурою подібно до сьогоднішніх центральних процесорів. Все це надає в розпорядження розробника низькорівневий, розподілюваний і високошвидкісний доступ до устаткування, роблячи CUDA необхідною основою при побудові серйозних високорівневих інструментів, таких як компілятори, математичні бібліотеки, програмні платформи.

1.4. Зразок виконання роботи

1. Запускаємо Visual studio 2010, і як за звичай ми створюєм проект на C# так і тут ми нажимаємо «Создать проект» Рис.1.

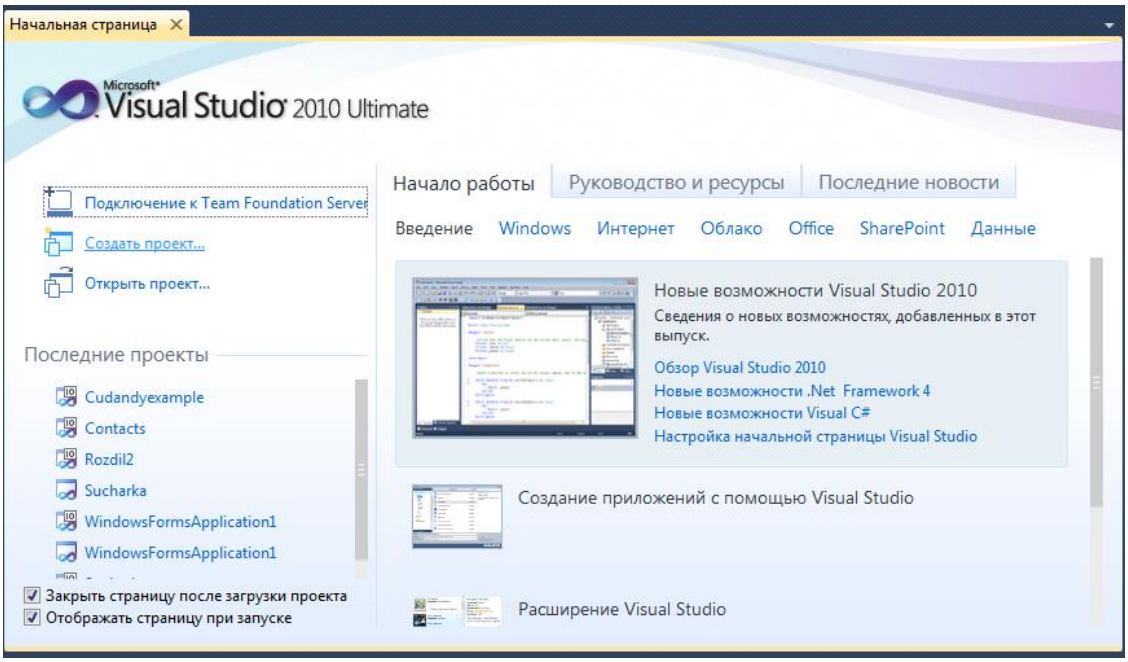

Рис.1.

2. Наступним нашим кроком буде вибрати вкладку «NVIDIA -> Cuda 5.0 Runtime» і давайте назвемо його «CudaExample». Після таких дій тиснемо кнопку «ОК». Рис.2.

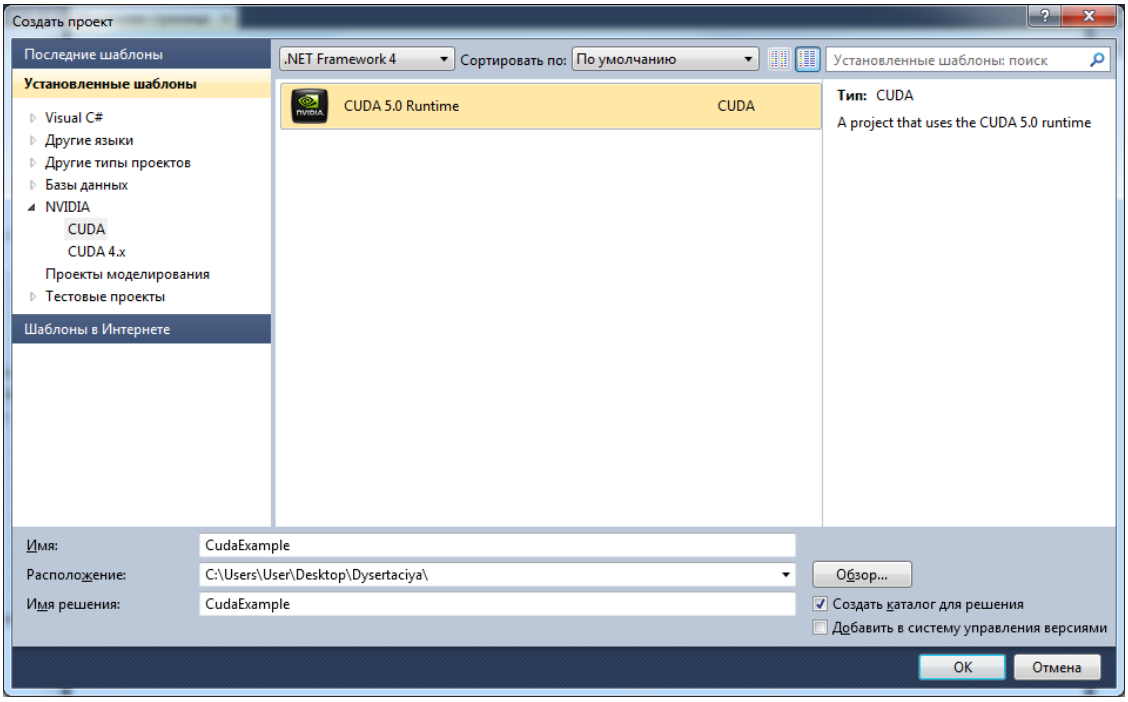

Рис.2.

3. У вас відкрився CUDA проект. Рис.3.

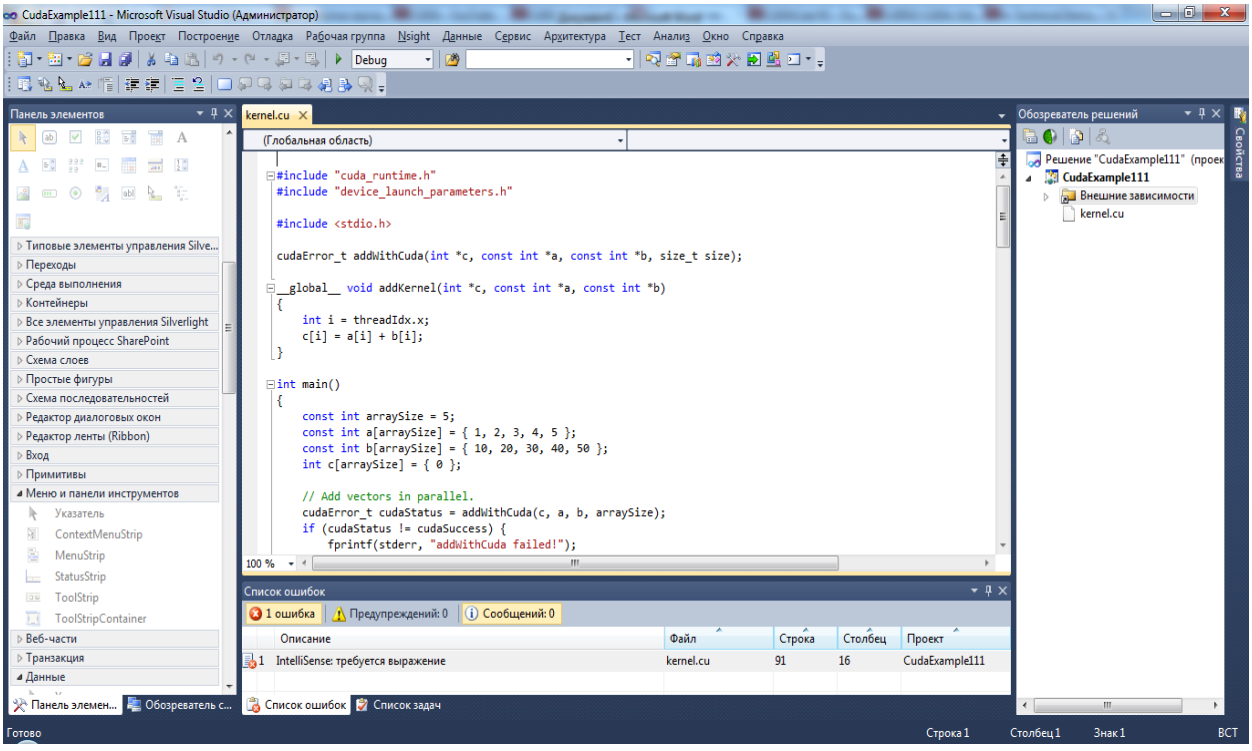

Рис.3.

1. Оскільки ми починаєм вивчати CUDA C, то стираєм дану програму, яка відкрилась позамовчуванні і почнемо з простого прикладу-програми «Привіт світ». Рис.4.

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <iostream>
#include <stdio.h>
using namespace std;

int main( void ) {
    printf("PS. Hello World!\n");
    getchar();
    return 0;
}
```

Рис.4.

Як ви бачити цей приклад працює повністю на CPU. Звичайно ви задасте собі кілька запитань:

1) Чим же ш відрізняється CUDA C від C;
2) Чи хіба CUDA не повинна працювати на GPU.

Відповіді звичайно не складні. Цей приклад приведений для того щоб показати, що насправді різниці між C і CUDA C нема. А робота з GPU ще попереду. З кожним кроком по лабораторній ми будем розширювати горизонти нашої програми і пізнання CUDA.

Для початку розглянемо дещо, що може використовувати GPU для виконання коду. Функція яка виконується на приладі(GPU) називається ядром(kernel). На Рис.5. приведено приклад коду із застосуванням kernel.

```
__global__ void kernel( void ){
}

int main( void ) {
    kernel<<<1,1>>>();
    printf("PS. Hello World!\n");
    getchar();
    return 0;
}
```

Рис.5.

Ми бачимо, що від попереднього варіанту програми у нас добавилися, функція kernel і класифікатор __global__. Добавляючи класифікатор __global__ ми тим самим кажемо компілятору що ця функція повинна компілюватися GPU,а не CPU. В данному випадку nvcc передає функцію kernel() компілятору, який обробляє код для GPU, а main() – компілятору для CPU. Але у вас виникне запитання, що це за дивний виклик kernel(). Цей дивний виклик служить для запиту до коду, а всі ці скобки – це позначення аргументу, якого ми збираємося передати виконуючому середовищу. Це не аргумент функції, виконаний GPU, а параметри, які впливають на то, як виконуюче середовище буде запускати код для GPU.

Тепер перейдемо до наступної модифікації нашої програми – передавання параметрів. Рис.6.

```
__global__ void add( int a, int b, int *c )
    {
        *c=a+b;
    }

int main( void ) {
    int c;
    int *dev_c;
    cudaMalloc((void**)&dev_c, sizeof(int));
    add<<<1,1>>>(2,7,dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost );
    printf("2+7=%d\n",c);
    cudaFree(dev_c);
    getchar();
    return 0;
}
```

Рис.6.

Як ми бачимо нових рядків тут багато, а концепцій всього дві:

1) Ми повинні передавати параметри ядру, як любій іншій функції;

2) Ми повинні виділяти пам'ять, якщо хочемо щоб GPU зробив дещо корисне, а саме вернув дані на CPU.

Ну якщо говорити про перший пункт то розумієш що нічого особливого там нема не дивлячись на скобки, виклик ядра виглядає і працює як виклик любої написаної на стандартному C функції.

Інтересніше виділення памяті з допомогою cudaMalloc(). Вона дуже похожа на стандартну функцію malloc(), але говорить виконуючому середовищу CUDA, що пам'ять має бути виділена на GPU. Перший аргумент – це покажчик на покажчик, в якому буде повернений адрес виділеної області пам'яті, а другий – розмір цієї області.

Сформулюєм правила використання покажчиків на пам'ять GPU:

1) Дозволяється передавати покажчики на пам'ять, виділеною cudaMalloc(), функціям, які виконуються на GPU.

2) Дозволяється використовувати покажчики на пам'ять, виділену cudaMalloc(), для читання і запису в цю пам'ять в код, який виконується в GPU.

3) Дозволяється передавати покажчик на пам'ять, виділену cudaMalloc(), функціям, які виконуються на CPU.

4) Не дозволяється використовувати покажчик на пам'ять, виділену cudaMalloc(), для читання і запису в цю пам'ять в коді, який виконується на CPU.

Ну і на кінець залишили дві функції: cudaFree(), cudaMemcpy().

Перша застосовується для звільнення пам'яті, яка виділена функцією cudaMalloc().

За допомогою другої ми можемо звернутися до памяті з коду який виконується на CPU. Вона, порівнюючи із звичайною функцією memcpy(), приймає додаткові параметри, які говорять про те, який з двох покажчиків адресую пам'ять GPU. В нашому випадку послідній параметр cudaMemcpy() рівний cudaMemcpyDeviceToHost, тобто початковий покажчик адресую пам'ять GPU, а кінцевий – пам'ять CPU.

1.5. Індивідуальні завдання

1). (2*2+2)*56;
2). (89-60)*(69-40);
3). (64/8)*10-1;
4). (34-4-4)+78;
5). 78/2*8;
6). (90/2)/2
7). 1+2+3+4+5*8/3;
8). 9876*10/120;
9). (67+67)/(89+56-65);
10). 456/2-5;

1.6. Контрольні запитання

1). Що таке технологія CUDA?
2). Поясніть принцип роботи функції cudaMalloc()?
3). Поясніть принцип роботи функції cudaMemcpy()?
4). Поясніть принцип роботи функції cudaFree()?
5). Яка роль функції kernel і класифікатора __global__ у програмі?