Лабораторна робота №17. Застосування CUDA-технологій в матричних операціях

МЕТА РОБОТИ: Розпаралелення матричних обчислень мовою CUDA C.

2.1. Програма роботи

2.1.1. Отримати завдання.

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

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

2.1.4. Оформити електронний звіт про роботу та захистити її.

2.2. Вказівки до виконання роботи

2.2.1. Студент, згідно з індивідуальним номером, вибирає своє завдання з розділу 2.5 і записує його до звіту.

2.2.2. Оголошення класу (структури), основну та відповідні допоміжні функції необхідно запрограмувати так, як це показано у розд. 2.4.

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

2.2.4. 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.

2.3.Теоретичні відомості

GPU розглядається як спеціалізований обчислювальний пристрій, який:

- є співпроцесором до CPU;
- володіє власною пам'яттю;
- надає можливість параллельного виконання величезної кількості окремих потоків

CUDA використовує велику кількість окремих потоків для обчислень, часто кожному обчислювальному елементові відповідає один потік. Всі нитки групуються в ієрархію - сітка / блок / потік (див.рис.2.3.1.). Верхній рівень - сітка - відповідає ядру і об'єднує всі потоки, що виконують дане ядро. Сітка (grid) є

одновимірним або двомірним масивом блоків (block). Кожен блок являє собою одно- двох- або тривимірний масив потоків (threads).

При цьому кожен блок являє собою повністю незалежний набір взаємодіючих між собою потоків, потоки з різних блоків не можуть між собою взаємодіяти.

Фактично блок відповідає незалежно розв'язуваній підзадачі, так, наприклад, якщо потрібно знайти добуток двох матриць, то матрицю-результат можна розбити на окремі меншого розміру підматриці. Знаходження кожної такої підматриці може відбуватися абсолютно незалежно від знаходження інших підматриць. Знаходження такої підматриці - завдання окремого блоку, всередині блоку кожному елементові підматриці відповідає окремий потік.

При цьому потоки всередині блоку можуть взаємодіяти між собою через:

- загальну пам'ять (що розділяється пам'яті)
- функцію синхронізації всіх ниток блоку (__synchronize)

З апаратної точки зору всі потоки розбиваються на так звані warp-и - блоки підряд ідучих потоків, які одночасно (фізично) виконуються і можуть взаємодіяти один з одним. Кожен блок потоків розбивається на декілька warp'ів, розмір warp'а для усіх існуючих зараз GPU дорівнює 32.

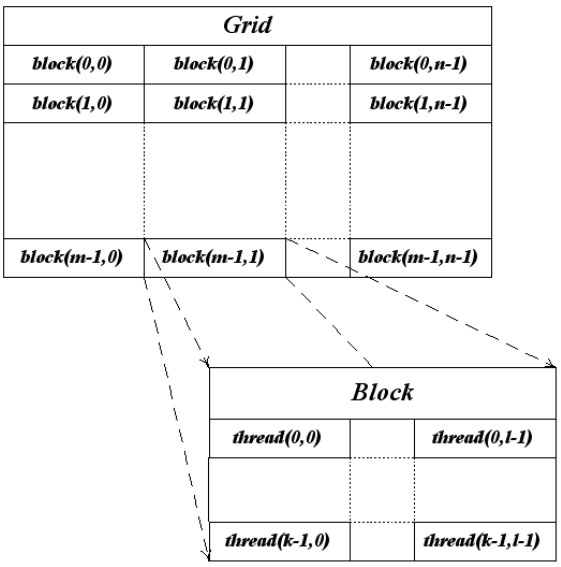

Рис.2.3.1. Ієрархія потоків в CUDA

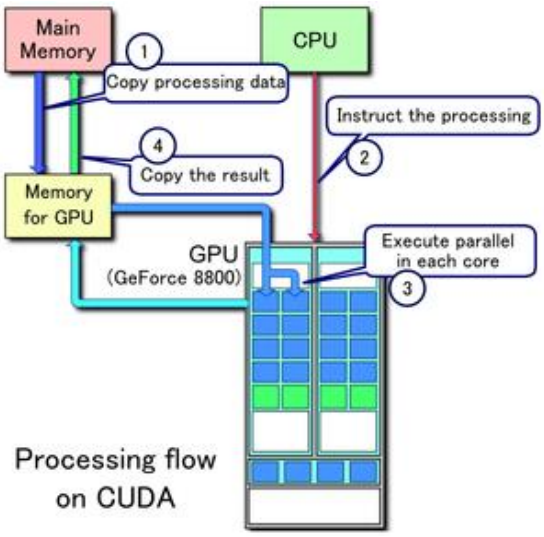

Рис.2.3.2. Цикл виклику ядра

Типовий цикл CUDA програми протікає наступним чином:

1. Виділення пам’яті на GPU
2. Копіювання даних із пам’яті CPU на GPU.
3. Конфігурація та запуск ядра
4. Синхронізація CUDA потоків для забезпечення того, що пристрій виконав всі свої завдання, перш ніж робити подальші операції із пам'яттю GPU.
5. Копіювання даних із GPUв пам’ять CPU.
6. Звільнення пам’яті GPU.

Програми для CUDA (відповідні файли зазвичай мають розширення.cu) пишуться на «розширеній» мові C (а тепер і C++) і компілюються компілятором nvcc.

Введені в CUDA розширення мови C складаються з:

- специфікаторів функцій, що показують, де буде виконуватися функція і звідки вона може бути викликана;
- специфікатором змінних, які задають тип пам'яті, що використовується для даних змінних;
- директиви, що служить для запуску ядра, котра задає як дані, так і ієрархію потоків;
- вбудованих змінних, що містять інформацію про поточні потоки;
- runtime, що включає додаткові типи даних.

В CUDA використовуються наступні специфікатори функцій (табл.1):

Специфікатор	Фукція виконується на	Функція може бути викликана із
device	device (GPU)	device (GPU)
global	device (GPU)	host (CPU)
host	host (CPU)	host (CPU)

Табл.1. Специфікатори функцій в CUDA

CUDA надає ряд функцій, котрі можуть бути використані лише на CPU (так званий CUDA Host API).Ці функції відповідають за:

- керування GPU;
- роботу з контекстом;
- роботу з пам’яттю;
- роботу з модулями;
- керування виконанням коду;
- роботу з текстурами;
- взаємодію з OpenGL і Direct3D;

2.4. Зразок виконання роботи

Запускаємо Visual studio 2010. Ми вже знаємо як створювати проет. Для тих хто забув повторіть наступне «Создать проект -> NVIDIA -> Cuda 5.0 Runtime» і давайте назвемо його «CudaExample2». Після таких дій тиснемо кнопку «ОК». Рис.1.

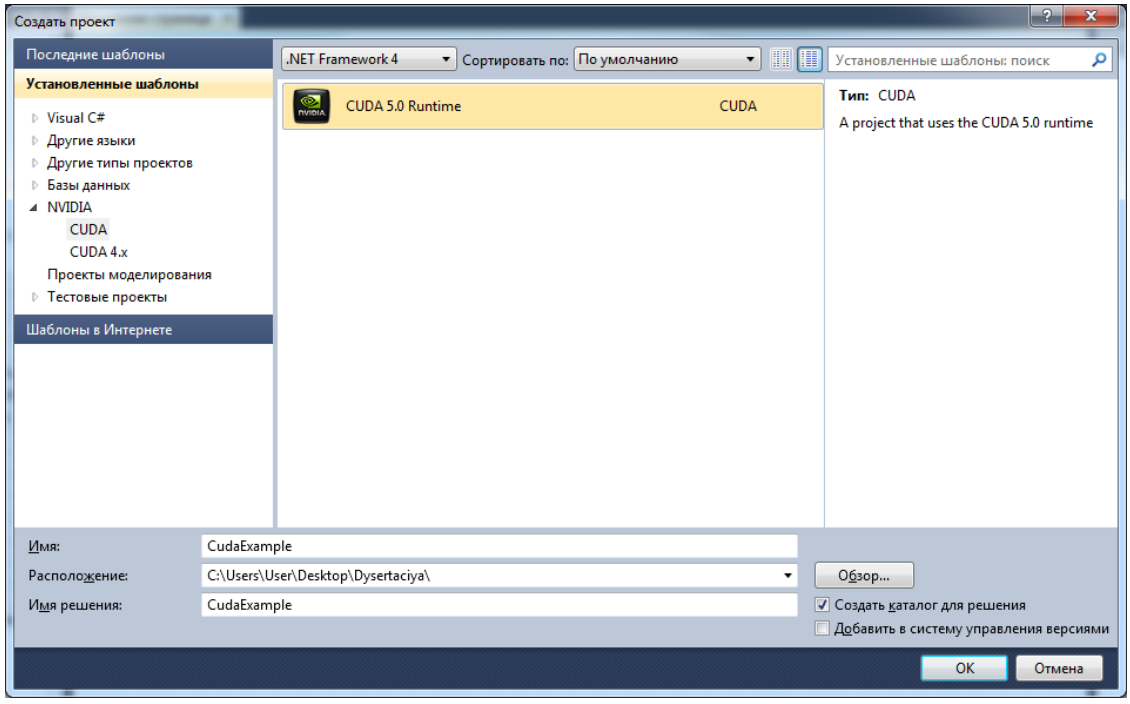

Рис.1.

1. У вас відкрився CUDA проект. Перш за все вам кидається в очі «exe»файл, який створюється по замовчуванні і написаний на подобі C або C++ (своєрідний мікс). В попередні роботі ми отримували базу по програмуванню CUDA і розглядали прості приклади. Щоб нагадати вам суть CUDA , то насправді використовується дуже хитрий процес. Тобто, ідея полягає в тому що CUDA розбиває програму на дві частити. Одна частина це плюсова частина(тобто та яка виконується на CPU), а друга - та яка виконується на GPU. Рис.2.

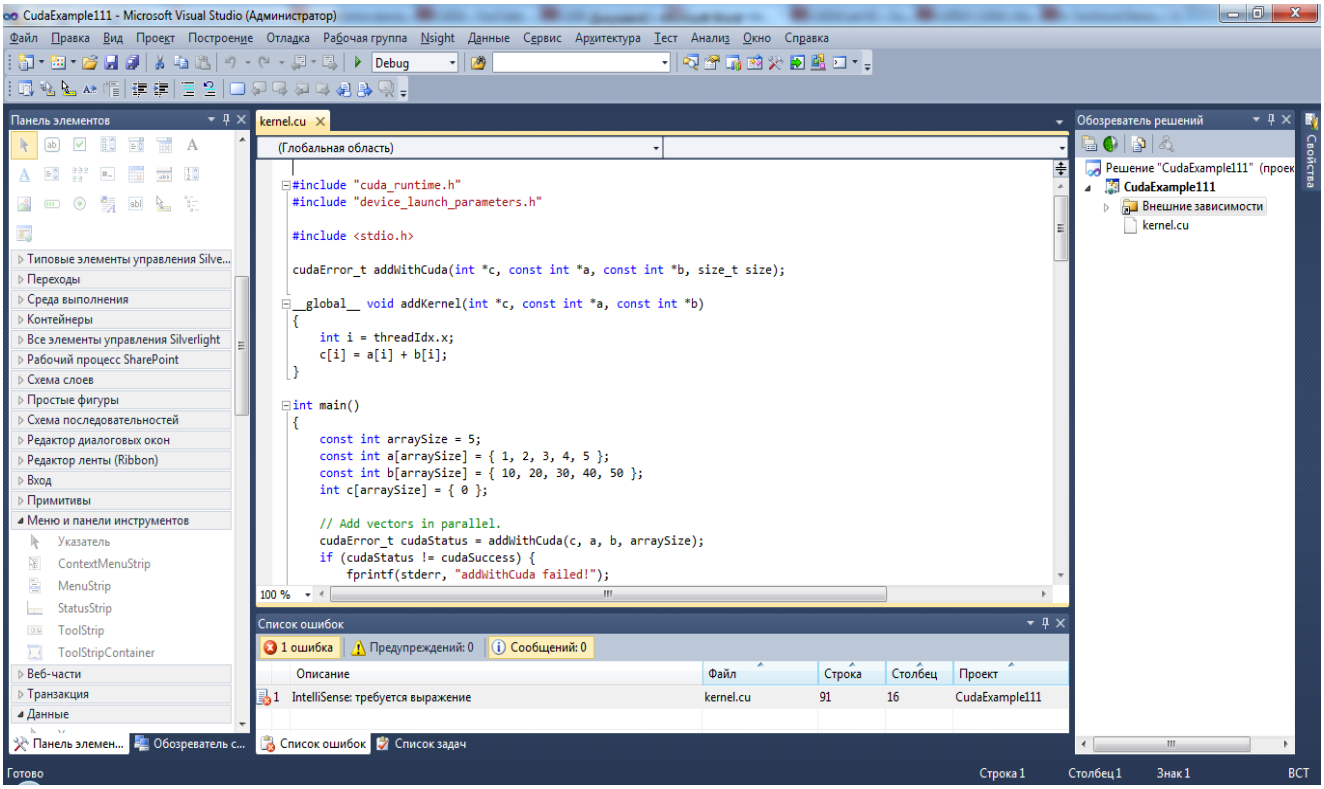

Рис.2.

1. Для того, щоб зрозуміти що таке CUDA в повній мірі, то ми розглянемо більш складний приклад ніж був у лабораторній роботі №1 і провіримо багаж знань який ми засвоїли за період навчання. Перед вами по замовчуванні як це говорилося з початку відкрилася програма додавання двох масивів. Я пропоную відтворити програму з «нуля», тобто з початку.
2. Стираємо увесь вміст вкладки Kernel.cu.
3. Наступною нашою задачею буде:
   1) Створити два масиви чисел (в кожному масиві по 5 чисел).
   2) Створити третій масив чисел і у нього помістити суму перших двох масивів.

      Для тих хто не взмозі виконати цієї задачі дивитись (додаток 1).
4. Ця програма виконується на CPU. Тепер ми переходимо до головної частини нашої лабораторної. Ми хочемо, щоб програма працювала на GPU. Для цього нам потрібно(аналогічно з лабораторною роботою №1), на виділені конструкти Рис.3., виділити пам'ять в GPU і перекинути туди наші дані.

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

void addArrays (int* a, int* b, int* c, int count)
{
     int i=0;
     while (i<count)
     {
         c[i] = a[i]+b[i];
         ++i;
     }
}

int main()
{
    int a[] = {10, 20, 30, 40, 50};
    int b[] = {1, 2, 3, 4, 5};
    int c[] = {5};

    addArrays(a,b,c, 5);

    for( int i=0; i<5; i++)
```

Рис.3.

5. Як це робиться? Насправді все дуже просто. Перш за все, ми говорим, що наші масиви є на «host» тобто на комп'ютері і виділяєм пам'ять на GPU(застосовуючи команду cudaMalloc). Рис 4.

```
int main()
{
    int ha[] = {10, 20, 30, 40, 50};
    int hb[] = {1, 2, 3, 4, 5};
    int hc[] = {5};

    int *da, *db, *dc;

    int size = sizeof(int)*5;
    cudaMalloc((void**)&da, size);
    cudaMalloc((void**)&db, size);
    cudaMalloc((void**)&dc, size);
```

Рис.4.

6. Наступним нашим кроком є копіювання даних. Тобто дані з «ha, hb, hc» нам потрібно скопіювати у «da, db, dc». Це робиться за допомогою команди «cudaMemcpy». Рис.5.

```
cudaMemcpy(da, ha, size, cudaMemcpyKind::cudaMemcpyHostToDevice);
cudaMemcpy(db, hb, size, cudaMemcpyKind::cudaMemcpyHostToDevice);
```

Рис.5.

7. Відповідно нам потрібно все це порахувати. Тобто ми використовуєм функцію, яка буде викликатися з CPU, але праювати на GPU(addArrays, 5-

кількісьть ниток, а 1- кількість потокових блоків) . Написавши параметри «da, db, dc», ми робимо безпосередньо виклик в GPU. Рис.6.

```
addArrays<<<1, 5>>>(da,db,dc);
```

Рис.6.

8. Відповідно давайте опишемо функцію addArrays. Використовуючи так звані ядра kernels, які починаються з «global». Тобто живе на GPU, а викликається з CPU. Рис.7.

```
__global__ void addArrays(int* a, int* b, int* c)
{
    int idx = threadIdx.x;
    c[idx]= a[idx] + b[idx];
}
```

Рис.7.

9. На даний момент ми заповнили «dc» на приладі(GPU). Тепер нам потрібно забрати дані(використовуючи «cudaMemcpy»). Рис.8.

```
cudaMemcpy(hc,dc, size, cudaMemcpyKind::cudaMemcpyDeviceToHost);
for (int i=0; i<5; ++i)
    cout << hc[i] << endl;
```

Рис.8.

10. Наступним нашим кроком буде проаналізувати наш проект за допомого вкладки «Nsight» Рис.9.

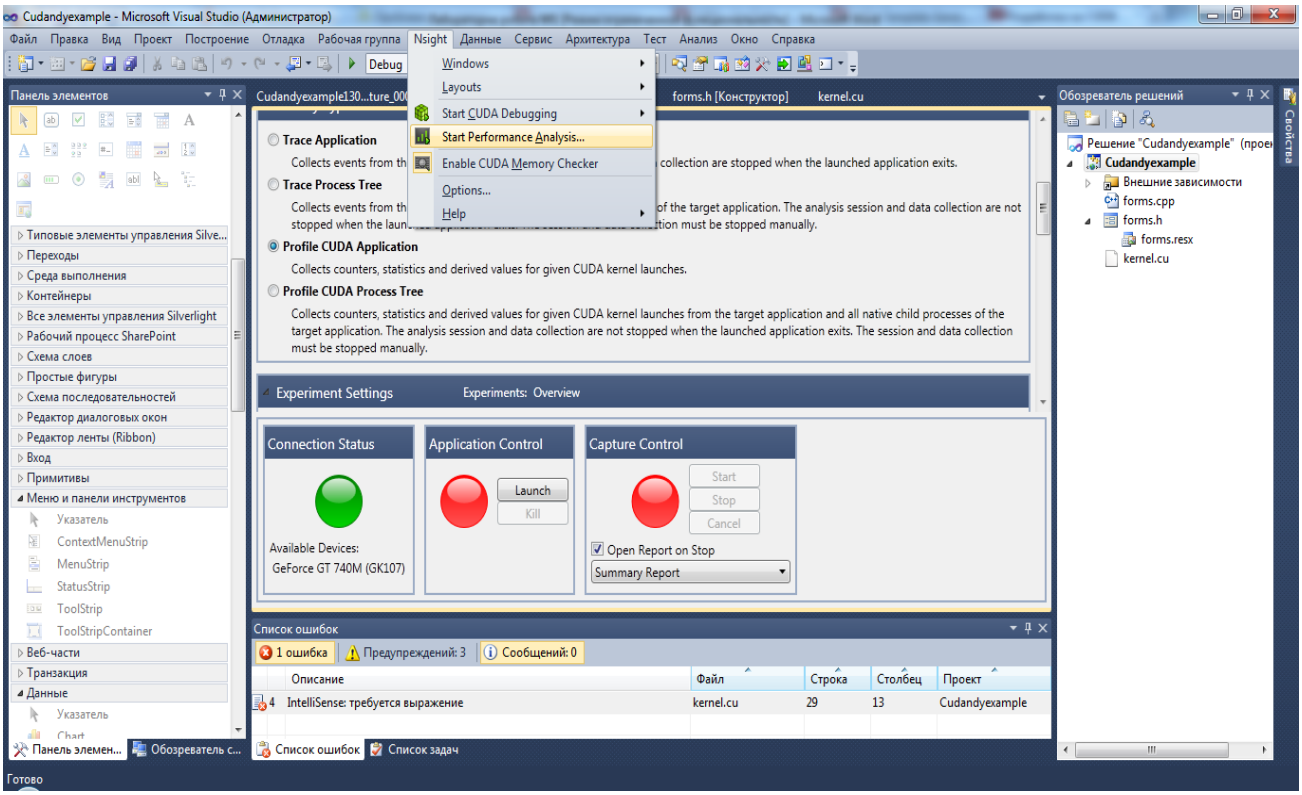

Рис.9.

2.5. Індивідуальні завдання

Створити масиви з 7 чисел і виконати наступні маніпуляції з ними:

1). (a[]+b[]-c[])/d[];
2). a[]*b[]+c[];
3). a[]*a[]+b[]*b[]+c[];
4). a[]/b[]*c[];
5). a[]-b[]-c[]*d[];
6). q[]-w[]-e[];
7). a[]+b[]+c[]-d[])/2;
8). (a[]*2b[])/a*2;
9). a[]+b[]+w[]/2;
10). a[]-b[]+2*a[]*b[];

2.6. Контрольні запитання

1). Яка ідея технології CUDA?
2). Які специфікатори функцій використовуються в CUDA?
3). Пояснити роботу потоків в CUDA?
4). Яким чином протікає цикл CUDA програми?