УДК 621.391

ВИКОРИСТАННЯ ГРАФІЧНИХ ПРОЦЕСОРІВ В ЗАДАЧАХ ЦИФРОВОЇ ОБРОБКИ СИГНАЛІВ В РЕАЛЬНОМУ ЧАСІ

О.В. Євчук, В.А. Ровінський, Ю.Й. Стрілецький

Івано-Франківський національний технічний університет нафти і газу, вул. Карпатська, 15, м. Івано-Франківськ, 76019, тел. (03422)4-60-77

Розглянуто можливість підвищення обчислювальної потужності персонального комп'ютера для реалізації згортки дискретизованого сигналу із використанням графічних процесорів. Доведено зростання швидкості виконання згортки для блоків даних різної довжини. Доведена практична цінність залучення обчислювальних потужностей графічного процесора із програмним інтерфейсом Compute Unified від NVIDIA Device Architecture (CUDA) .
Ключові слова: графічний процесор, обчислювальна потужність, згортка.

Рассмотрена возможность повышения вычислительной мощности ПК для реализации свертки дискретизованный сигнала с использованием графических процессоров. Доказано увеличение скорости выполнения свертки для блоков даных разной длины. Доказана практическая ценность привлечения вычислительных мощностей графического процессора с программным интерфейсом Compute Unified от NVIDIA Device Architecture (CUDA).
Ключевые слова: графический процессор, вычислительная мощность, свёртка.

The possibility of increasing computing power of a personal computer for convolution of discrete signal using graphics processors was considered. Proved growth rate of reduction for blocks of different lengths. We prove the practical value of bringing computing power of GPU Compute Unified interface software from NVIDIA Device Architecture (CUDA).
Key words: graphic processor, computation power, convolution.

В задачах неруйнівного контролю та діагностики об'єктів часто виникає необхідність цифрової обробки великих масивів результатів вимірювань фізичних величин в реальному часі. Це стосується, зокрема, задач вібраційної діагностики та контролю акустоемісійними методами, де необхідність забезпечення прийнятної роздільної здатності в часі в сукупності із наявністю високочастотних інформативних складових у сигналі вимагає обробки значних обсягів даних, а, отже, значних обчислювальних потужностей.

Суттєвого підвищення продуктивності обробки даних на персональних комп'ютерах можна досягти за рахунок використання обчислювальних потужностей графічних процесорів [1, 2]. Основне призначення графічних процесорів (GPU) – обробка графічних зображень, зокрема тривимірних. Специфіка такої обробки зумовлює побудову графічних процесорів за принципом масивно-паралельної архітектури. Традиційна модель обчислень на центральному процесорі (CPU) передбачає послідовну модель виконання, тобто програма – це набір послідовних інструкцій, які виконуються одна за одною. Натомість, модель обчислень GPU передбачає дуже високий ступінь паралелізму. В GPU наявні великі масиви обчислювальних модулів, кожен з яких працює певною мірою незалежно, але всі вони виконують одну послідовність дій, одну програму над великим масивом даних. CPU, на відміну від GPU, містять незначну кількість важких процесорних ядер. Великі розміри кеша забезпечують досить високу продуктивність при виконанні послідовного коду, але при цьому дуже важко вмістити велику кількість ядер на кристал. Для представлених на ринку Intel-процесорів типовою є наявність 4-8 ядер на кристалі, в той час як для NVIDIA масив процесорів налічує сотні одиниць. Наприклад, GPU 10-ї серії Tesla містить 240 процесорних ядер, які працюють паралельно [3], а в 2009 р. була розроблена нова архітектура NVIDIA Fermi (рис. 1). Тепер GPU складається з 16-ти потокових мультипроцесорів, які містять по 32 шейдерних ядра, що в сукупності дає 512 CUDA-ядер. Блоки розташовані навколо загальної кеш-пам'яті другого рівня. Кожен з блоків складається з планувальника і організатора, виконавчих модулів і файлів регістрів і кеш-пам'яті першого рівня.

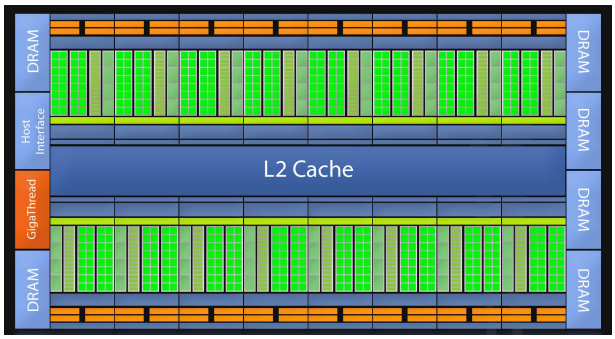

Рисунок 1 – Архітектура Fermi

Кардинальні відмінності між архітектурами CPU та GPU – це те, що в графічних процесорах більша частина площі кристала витрачається на самі обчислювальні модулі, яких міститься дуже багато, в той час як Intel-подібні центральні процесори витрачають площу кристала на кеш і мають досить невелику кількість потужних обчислювальних ядер, розпаралелювання виконання задач яких покладається на операційну систему і відбувається часто за досить простими принципами – “одне ядро - один програмний потік (thread)” або “ одне ядро - одна програма”.

Якщо в обчислювальній системі не застосовується складна графічна обробка, пов’язана з візуалізацією тривимірних динамічних об’єктів, то більша частина обчислювальних потужностей графічного процесора не використовується. Щоб все-таки задіяти ці потужності для задач неграфічного характеру, які також потребують інтенсивних паралельних обчислень, було розроблено спеціальні програмні інтерфейси, одним із яких є CUDA C для графічних процесорів NVIDIA з архітектурою CUDA [4]. CUDA базується на стандартному C, який розширений додатковими мета-конструкціями для того, щоб ефективно програмувати паралельні завдання. Модель CUDA підтримує гетерогенні обчислення, і програма будується як сукупність секцій: деякі секції є масивно паралельними і виконуються на GPU (т.зв. device-код), і є секції, які виконуються послідовно на CPU (host-код). В одній програмі можна змішувати паралельні і послідовні ділянки коду. Такий код передається на компіляцію компілятору nvcc, який створює об’єктні файли, що передаються для збірки програми редактору зв’язків Microsoft Visual Studio

Однією із найбільш обчислювально затратних операцій є згортка, що використовується в алгоритмах цифрової фільтрації, обчислення кореляційних функцій тощо. В останні роки задача обчислення згортки з використанням графічних процесорів активно досліджується, зокрема в [5, 6] розглядається опис реалізації швидкої згортки з рівномірним розбиттям ядра згортки на блоки та поблоковим надходженням на обробку вхідного сигналу для систем обробки аудіосигналів у реальному часі. Найбільший виграш у швидкодії порівняно з реалізацією на центральному процесорі досягається у випадку обробки кількох блоків вхідного сигналу одночасно, що, однак, вносить додаткову затримку вихідного сигналу відносно вхідного, а також для багатоканальної згортки, особливо на новіших чіпах NVIDIA із великою кількістю процесорних ядер.

У випадку невеликої довжини ядра згортки та відсутності багатоканальної обробки реалізація прямого алгоритму обчислення згортки

$$y_i = \sum_{k=0}^{M-1} h_k \cdot x_{i-k} \qquad (1)$$

може більш ефективно використовувати паралелізм GPU, ніж швидкий алгоритм з використанням перетворення Фур’є. Для дослідження швидкодії даного алгоритму було написано програму з використанням CUDA SDK.

Функція обчислення згортки для GPU виглядає наступним чином:

```
__global__ void gpuBruteConvol(const float* X, const float* H, float* Y, int N, int M)
{
 int k, m, k0;
 float sum=0;
int i = blockDim.x * blockIdx.x + threadIdx.x;
 if(i < N+M-1)
  {
k0 = i<N ? 0: i-N+1;
m = i<M ? i:M;
for(k=k0; k<m; k++)
{
sum+=H[k]*X[i-k];
}
Y[i]=sum;
  }
}
```

Виклик цієї функції з головної програми оформляється наступним чином:

```
cutilSafeCall( cudaMemcpy(d_X, h_X, N*sizeof(float), cudaMemcpyHostToDevice) );
cutilSafeCall( cudaMemcpy(d_H, h_H, M*sizeof(float), cudaMemcpyHostToDevice) );
int blocksPerGrid = (N+M-1 + threadsPerBlock
```

```
- 1) / threadsPerBlock;
    gpuBruteConvol<<<blocksPerGrid,
threadsPerBlock>>>(d_X, d_H, d_Y, N, M);

    cutilSafeCall( cudaMemcpy(h_Y, d_Y, (N+M-
1)*sizeof(float), cudaMemcpyDeviceToHost) );
```

Тестування проводились на ПК з центральним процесором Core 2 Duo Q8200 (4 ядра, 2,3ГГц, 2Гб ОЗП) та графічною платою NVIDIA GeForce GT 9500 (4 процесорних ядра, 1,4 ГГц, 512 Мб ОЗП). Довжина ядра M приймалась рівною від 8 до 512, а довжина сигналу N – в межах від 10^4 до 10^6 відліків. Для визначення часу виконання коду на GPU застосовувались функції CUDA C для роботи з об'єктами типу cudaEvent_t, для CPU – функція Windows API QueryPerformanceCounter.

Для наведеної вище реалізації час обчислень на GPU виявився більшим, ніж для CPU, в середньому вдвічі. Відносно мала кількість процесорних ядер на графічній платі та нижчі обчислювальні характеристики, ніж у центрального процесора, можуть частково пояснювати такий результат, однак даний код має також деякі можливості для оптимізації. Зокрема, використання пам'яті констант (Constant Memory) дозволяє зменшити час читання даних за умови, що всі потоки в деякому пулі потоків (warp) одночасно звертаються до однієї і тієї ж комірки пам'яті констант. Модифікація коду за умови розміщення в пам'яті констант ядра згортки полягає в оголошенні масиву ядра згортки як глобальної змінної з модифікатором __constant__ та у використанні для цього масиву функції cudaMemcpyToSymbol замість cudaMemcpy. В результаті код GPU виявився швидшим за код CPU в середньому в півтора рази (рис. 2). Ця залежність зберігається для різних довжин масиву сигналу N, за винятком поєднання дуже малої довжини ядра (8, 16) з невеликою довжиною масиву сигналу (до 10000, в цьому випадку виграш у часі відсутній).

Можна було б продовжити оптимізацію, розмістивши в пам'яті констант також і вхідний сигнал. Однак обмежений розмір пам'яті констант (64Кбайт, що відповідає 16384 елементам з плаваючою комою одинарної точності) вимагав би поділу сигналу на невеликі блоки та поступової передачі їх на обробку, тобто фактично реалізації секційної згортки, наприклад, методом накладання-додавання, що також збільшує і кількість арифметичних операцій. Крім того, оскільки звертання до пам'яті констант, на відміну від глобальної пам'яті GPU, є серіалізованими, то виграш в часі досягається тільки при одночасному звертанні до однієї і тієї ж адреси, а коли кожен із потоків в пулі звертається до іншої адреси, швидкодія, навпаки, може зменшитись. З формули (1) очевидно, що синхронне звертання до елементів ядра можливе тільки при розподіленому по M адресам звертанні до елементів сигналу, і навпаки.

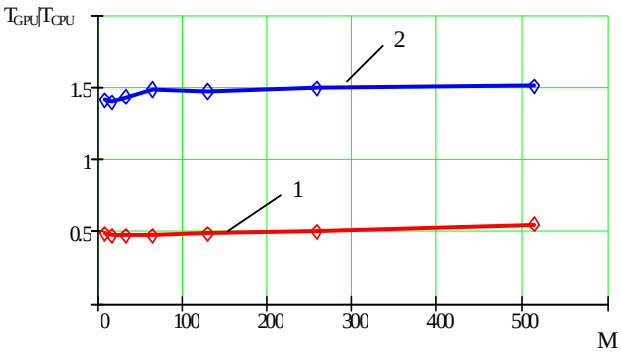

Рисунок 2 – Виграш в часі для неоптимізованої версії коду (1) та з використанням пам'яті констант (2) для різних довжин ядра згортки M при N=100000

Було досліджено також залежність швидкодії алгоритму від кількості блоків та потоків, що передаються на виконання графічному процесору. Ці кількості задаються при виклику коду GPU:

```
    gpuBruteConvol<<<blocksPerGrid,
threadsPerBlock>>>(d_X, d_H, d_Y, N, M);
```

В базовому алгоритмі кількість потоків приймається рівною максимально допустимій для даного процесора (визначається за допомогою функції cudaGetDeviceProperties), а кількість блоків – як результат ділення довжини сигналу на кількість потоків із заокругленням до більшого цілого. При такому розподілі кожен потік обчислює лише одне значення вихідного сигналу. Оскільки кількість процесорних ядер суттєво менша за загальну кількість потоків, існує імовірність, що GPU витрачає деякий додатковий час на переключення задач між потоками, хоча ці затрати і будуть суттєво меншими, ніж в аналогічному випадку у центрального процесора, в силу особливостей архітектури GPU. Для перевірки залежності часу обчислень від кількості блоків було модифіковано код функції gpuBruteConvol:

```
    int delta=blockDim.x*gridDim.x;
      while(i < N+M-1)
      {
// ...як раніше...

    i+=delta;
```

}

Результати тестування свідчать, що час обчислень практично не залежить від кількості блоків, хоча є дуже незначна тенденція до зменшення часу обчислення із збільшенням кількості блоків (рис. 3). Отже, оптимальною з цієї точки зору є первинна версія алгоритму. Аналогічні результати було отримано також для різних кількостей потоків всередині блоку.

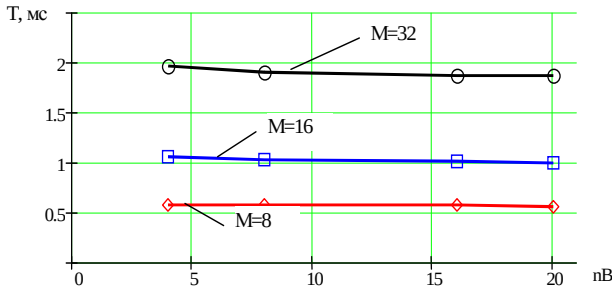

Рисунок 3 - Залежність часу обчислення згортки для сигналу довжиною 10000 відліків та різних довжин ядра M

ВИСНОВКИ

Проведені дослідження виявили, що використання графічних процесорів дозволяє ефективно розпаралелити обчислення в алгоритмі дискретної згортки сигналу. Це дає моливість скористатися обчислювальними можливостями персонального комп'ютера для вирішення задач неруйнівного контролю, які раніше можна було реалізувати тільки з використанням цифрових сигнальних процесорів.

1. J.D. Owens. A survey ofgeneral-purpose computation on graphics hardware /1 J.D. Owens, D. Luebke, N. Govindaraju, [at al.] // Proceedings of EUROGRAPHICS. - 2005. - http://www.idav.ucdavis.edu/func/return_pdf.pub_id=907. 2. Jason Sanders. CUDA by example : an introduction to general-purpose / Jason Sanders, Edward Kandrot //. GPU programming. Addison-Wesley. - 2010. – 290 p. 3. Вычислительный процессор NVIDIA® TeslaTM C1060. - http://www.nvidia.ru/object/tesla_c1060_ru.html. 4. NVIDIA CUDA C Programming Guide. Version 3.2 . – NVIDIA - 2010. - http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf. 5. Michael Rush, Convolution engine utilizing NVIDIA's G80 processo. - 2007. - http://www.ece.ucdavis.edu/mmrush/mmrush_eec277_final_writeup.pdf. 6. F.Wefers. High-performance real-time FIR-filtering using fast convolution on graphics hardware / F.Wefers, J.Berg. // Proc. of the 13th Int. Conference on Digital Audio Effects (DAFx-10), Graz, Austria. - September 6-10, 2010.

Поступила в редакцію 03.12.2010 р.

Рекомендував до друку докт. техн. наук, проф. Заміховський Л.М.