УДК 004.02

А.О. Мельник, Ю.Є. Мицко
Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин.

ВИКОНАННЯ ПОДАНИХ ПОТОКОВИМ ГРАФОМ АЛГОРИТМІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ GPGPU

© Мельник А.О., Мицко Ю.Є., 2012

Здійснено короткий огляд технологій виконання обчислень на графічному процесорі (GPGPU) та особливостей написання програм при їх використанні. Перевірено ефективність способу виконання поданих потоковим графом алгоритмів на прикладі алгоритму швидкого перетворення Фур'є на графічному процесорі (GPU) з підтримкою технології GPGPU. Показано перспективу цього способу як для виконання на GPU, так і на CPU.

Ключові слова: технологія GPGPU, потоковий граф алгоритму, CUDA.

Short overview of general-purpose computation on GPUs (GPGPU) and features of programming for their using is done. The method performing of given algorithm as flow graph at the example of FFT is investigated and verified on GPU that supports GPGPU technology. The prospects of this method execution on GPU and CPU are shown.

Key words: GPGPU technology, algorithm flow graph, CUDA.

Вступ

У наш час швидко розвиваються комп'ютерні технології. Збільшується продуктивність процесорів, їх кількість та кількість ядер у них. Однак багато програм є послідовними, виконуються на одному ядрі, тобто нераціонально використовують ресурси обчислювальної системи. Саме тому нині є актуальним питання пошуку способів їх розпаралелення.

Існує багато різних вирішень проблеми, однак вони не є універсальними і для кожної програми потрібен свій підхід щодо її розпаралелення. Для прикладу, в OpenMP потрібно явно вказувати в коді програми, яка її частина виконуватиметься паралельно. Якщо програма вже створена і виконується послідовно, то за такого підходу для паралельного виконання програмісту потрібно самому вносити зміни в неї. Тому доцільним є пошук способів автоматичного перетворення послідовного коду програми на паралельний. І хоча перероблений програмістом код може бути ефективніший, процес перетворення ним послідовного коду на паралельний може потребувати багато часу та коштів.

1. Огляд технологій GPGPU

Технологія GPGPU (General-Purpose computation on GPUs) дає змогу використовувати для пришвидшення програм загального призначення графічний процесор, який зазвичай разом з центральним процесором використовується для обробки комп'ютерної графіки [1]. Центральний процесор (ЦП) разом з графічним процесором (ГП) є ефективною комбінацією, тому що ЦП містить декілька ядер, оптимізованих для послідовного виконання, а ГП містить тисячі менших, ефективніших ядер для паралельного виконання. Послідовна частина програми виконується на ЦП, тоді як паралельну частину опрацьовує ГП. GPGPU має декілька реалізацій: CUDA, OpenCL, DirectCompute, ATI Stream Technology.

CUDA (Compute Unified Device Architecture) – програмно-апаратна архітектура, розроблена фірмою nVidia для паралельних обчислень на ГП, які підтримують технологію GPGPU [2]. CUDA використовує паралелізм на рівні потоків типу SIMT (Single Instruction Multiple Threads). Масштабована програмна модель дає CUDA можливість виконувати програми з різних ГП: від високопродуктивних до звичайних, масштабуючи кількість процесорів і розділів пам'яті [3].

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

OpenCL (Open Computing Language) – фреймворк (набір програмних засобів) для паралельних обчислень, дає змогу розподілити їх між ГП та ЦП. Він підтримує велику кількість рівнів паралелізму й ефективно працює з гомогенними або гетерогенними системами та з системами, які містять комбінації різних типів процесорів (ЦП, ГП та інших) [4].

DirectCompute – прикладний програмний інтерфейс (API), призначений для організації обчислень на ГП [5]. Вперше з’явився в 11 версії DirectX, згодом адаптований під 9 та 10 версії. DirectCompute орієнтований на операційну систему Windows, є альтернативою CUDA і OpenCL. За допомогою цієї технології складні розрахунки графічних ефектів можна перенести з ЦП на ГП. Також ця технологія дозволяє задіяти ГП для прискорення інших прикладних програм.

AMD ATI Stream Technology – аналог CUDA, набір апаратних та програмних технологій для виконання обчислень на ГП AMD спільно з ЦП. Сфокусовано AMD з ATI Stream в основному на транскодуванні. AMD зазначила, що це сфера, де ГП може бути набагато продуктивнішим, ніж ЦП.

Програми для ГП фірми nVidia в межах архітектури CUDA пропонується писати розширеною мовою C. За вихідним кодом компілятор nvcc генерує виконавчу програму. Мова C доповнюється деякими ключовими словами для опису конфігурації ядра ГП та запуску обчислень.

Перед запуском функції, яка буде виконуватись ядром ГП, потрібно виділити пам’ять у ГП (залежно від потреб задачі), скопіювати вхідні дані з основної пам’яті комп’ютера у виділену пам’ять ГП.

Для простого запуску функції на ГП необхідно вказати конфігурацію ядра у такій конструкції:

kernel_name<<<grid, block>>>(params); де

- kernel_name – назва функції;
- grid – конфігурація блоків типу dim3. Може бути одно-, дво- або тривимірною (для версії 2.0 і вище), максимальне значення кожного виміру також залежить від версії ядра;
- block – конфігурація потоків типу dim3. Також може мати від одного до трьох вимірів. Максимальне значення у вимірах x та y - 512 (1024 для версії 2.0 і вище), а для виміру z – 64. Також загальна кількість потоків у блоці не повинна перевищувати 512 (1024 для версії 2.0 і вище);
- params – вхідні параметри для функції.

Після виконання функції потрібно скопіювати результат з пам’яті ГП в основну пам’ять і звільнити пам’ять ГП.

2. Постановка задачі

Для виявлення просторових і часових залежностей операцій алгоритму його подають у вигляді потокового графа (ПГА) [6]. Наявність в ПГА просторових і часових залежностей між операціями алгоритму повністю вирішує проблему його розпаралелення. Відповідно, не потрібно писати паралельні програми для реалізації алгоритму в багатопроцесорних та інших паралельних комп’ютерних системах.

Метою дослідження є перевірка ефективності способу виконання поданих потоковим графом алгоритмів з описом їх структури, на ГП, який підтримує технологію GPGPU (рис. 1).

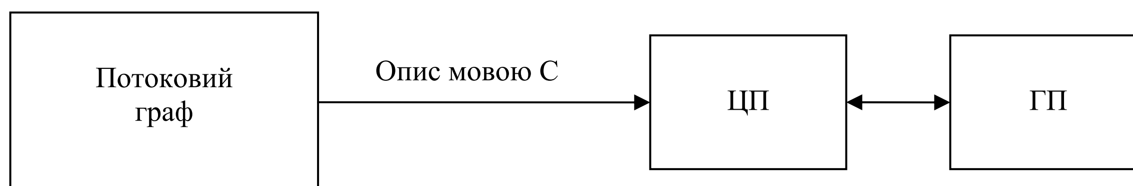


Рис. 1. Спосіб виконання поданих потоковим графом алгоритмів на системі ЦП-ГП

Для реалізації способу вибрано ГП фірми nVidia з архітектурою CUDA.

Для розв’язання задачі необхідно:

1. Вибрати алгоритм, який можна розпаралелити. Зважаючи на широке використання та можливості розпаралелення, вибрано алгоритм швидкого перетворення Фур’є (ШПФ).

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

2. Побудувати потоковий граф вибраного алгоритму для різної кількості вхідних даних.
3. Написати програму реалізації вибраного алгоритму традиційним способом для виконання на ЦП.
4. Написати програму реалізації вибраного алгоритму на основі побудованого відповідно до п. 2 потокового графа для виконання на ЦП і ГП.
5. Виконати створені програми з різною кількістю вхідних даних на ЦП та на системі ЦП і ГП та порівняти час їх виконання.

3. Потоковий граф алгоритму ШПФ

Щоб перевірити працездатність наведеного вище способу, вибрали алгоритм швидкого перетворення Фур'є. Створено два потокових графи, які подають алгоритм ШПФ для 65536 та 1024 точок. Відповідно розмір матриць 16×65536 та 10×1024. Базову операцію "метелик" ШПФ, яка є функціональним оператором потокового графа, наведено на рис. 2, формула (1):

$$X_{out} = X_{in} + Y_{in}W^{n};$$
$$Y_{out} = X_{in} - Y_{in}W^{n}; \qquad (1)$$

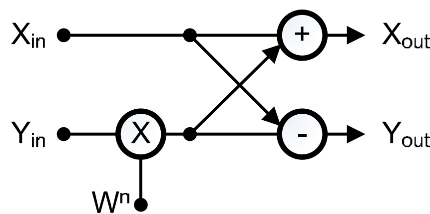
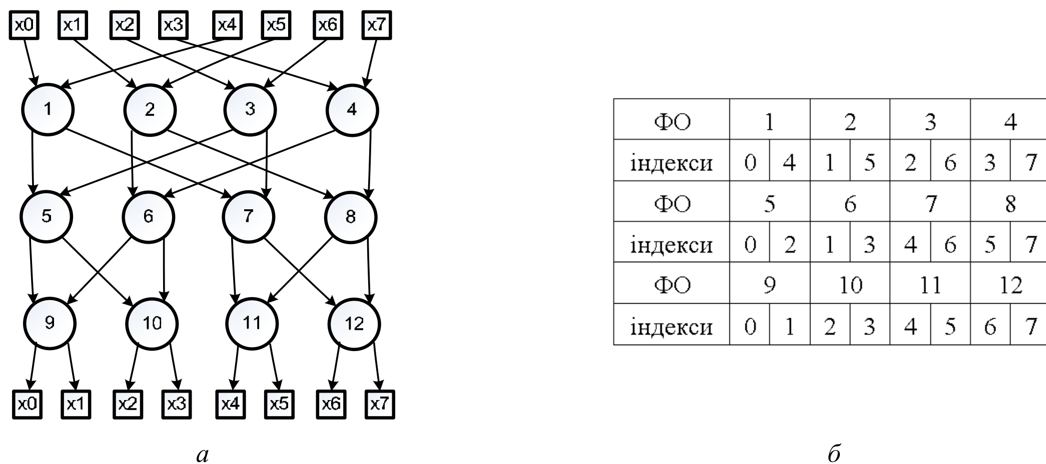


Рис. 2. Базова операція ШПФ "метелик"

На рис. 3, а зображено потоковий граф алгоритму ШПФ [3] для кількості точок 8 та матрицю, яка описує його структуру (рис. 3, б). Матриця створена для спрощення опису графа мовою програмування.



ФО	1		2		3		4	
індекси	0	4	1	5	2	6	3	7
ФО	5		6		7		8	
індекси	0	2	1	3	4	6	5	7
ФО	9		10		11		12	
індекси	0	1	2	3	4	5	6	7

а *б*

Рис. 3. Потоковий граф алгоритму ШПФ для 8 точок (а) та відповідна йому матриця (б)

4. Реалізація тестового алгоритму

Традиційний спосіб реалізації алгоритму ШПФ для виконання на ЦП – рекурсивний. Функція recurseFFT приймає вказівник на масив та розмір масиву. В ході рекурсивного спуску масив ділиться на дві рівні частини, доки його розмір більший за 2 (рис. 4, а). Функція butterfly приймає вказівник на масив та індекси елементів, над якими потрібно виконати базову операцію ШПФ "метелик".

Порядок виконання алгоритму ШПФ на основі потокового графа для ЦП і ГП зображено на рис.4, б. Функція hostMatrixFFT приймає вказівник на матрицю (M), вказівник на масив даних (X) та розмірність матриці (rowSize, colSize).

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Код функцій recurseFFT та hostMatrixFFT:

void recurseFFT(float *X, int N){
register int halfN = N/2;
float a, b;
for(int i = 0; i < half; ++i){
// операція "метелик" над
// X[i] та X[half+i]
}
if(N > 2){
recurseFFT(X, half);
recurseFFT(X + half, half);
}
}
	void hostMatrixFFT(int *M, float *X, int rowSize,
	int colSize){
	register int p1, p2;
	for(int i = 0; i < colSize; ++i){
	for(int j = 0; j < rowSize; j+=2){
	p1 = M[rowSize*i + j];
	p2 = M[rowSize*i + j + 1];
	// операція "метелик" над
	// X[p1] та X[p2]
	}
	}
	}

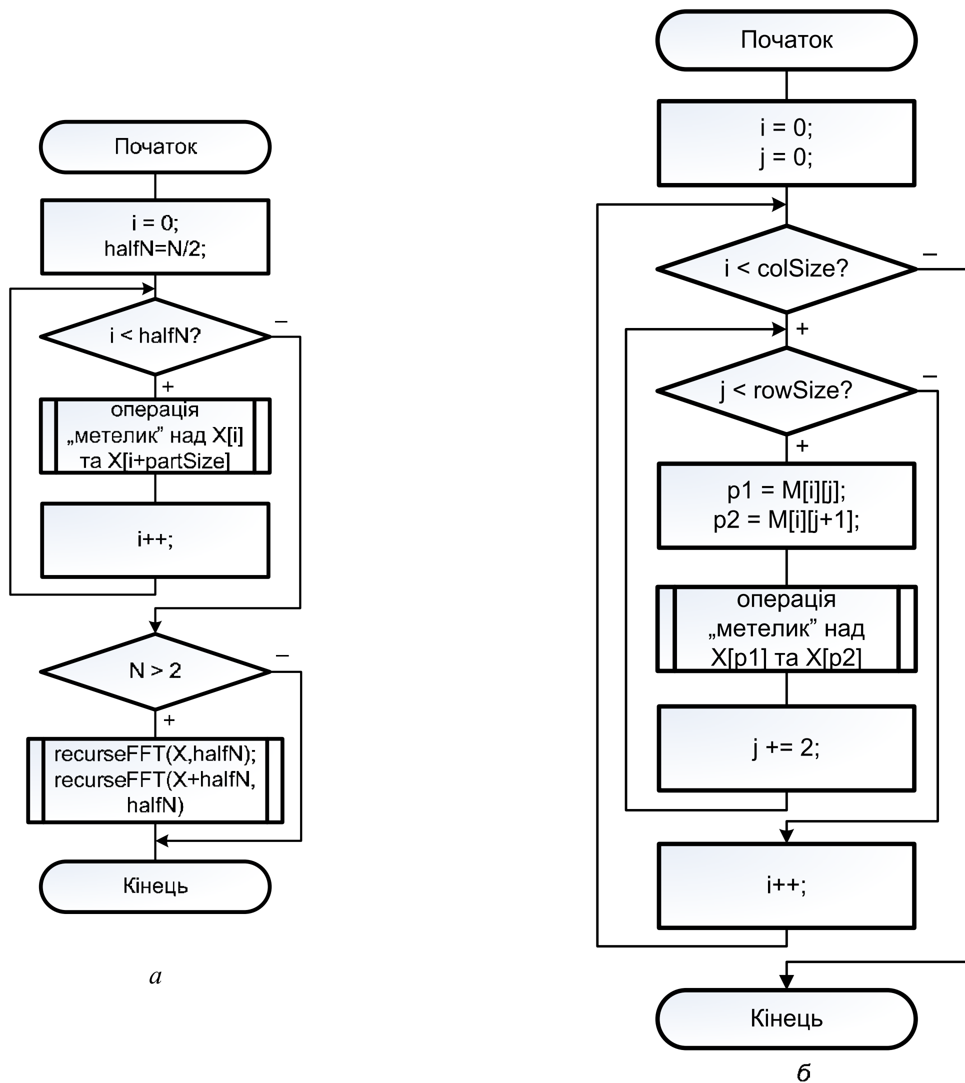


Рис. 4. Блок-схеми виконання рекурсивної функції recurseFFT (а) та функції на основі потокового графа hostMatrixFFT (б)

Для виконання на ГП варіанта поданого потоковим графом алгоритму ШПФ функцію hostMatrixFFT було модифіковано в devMatrixFFT. Оскільки devMatrixFFT виконуватиметься на кожному задіяному ядрі ГП, в ній виконується розподіл вхідного масиву між потоками. Вхідний масив X копіюється частинами у вихідний масив Y, яким і виконуються операції. Зауважимо, що в такому варіанті кількість блоків у конфігурації ядра ГП не повинна перевищувати кількості мультипроцесорів у ньому, оскільки між ними не відбувається синхронізації.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Код функції devMatrixFFT:над

```
__global__ void devMatrixFFT( int* M, float* X, float* Y, int colSize, int rowSize){
	// Обчислення індексів для кожного ядра
	int process_count = rowSize / (blockDim.x * gridDim.x);
	int start = process_count * (blockIdx.x * blockDim.x + threadIdx.x);
	int end = start + process_count;
	for(int i = start; i < end; ++i){   // Копіювання вхідного масиву у робочий
		Y[i] = X[i];
	}
	int p1, p2;      // Виконання обчислень
	int shift = 0;
	for(int i = 0; i < colSize; ++i){
		for(int j = start; j < end; j+=2){
			p1 = M[shift + j];
			p2 = M[shift + j + 1];
			// операція "метелик" над
			// X[p1] та X[p2]
		}
		shift += rowSize;    __syncthreads();
	}
}
```

Іншою модифікацією буде функція devMatrixMultyFFT, яка ділить вхідний масив на підмасиви, і над кожним з них виконує задані алгоритмом ШПФ перетворення. Для цього розмір матриці повинен бути кратний до розміру вхідних даних. Кількість виконавчих блоків у конфігурації ядра ГП повинна дорівнювати кількості перетворень, які будуть виконані над вхідним масивом. Кожен блок (мультипроцесор) виконує окреме перетворення над своєю частиною даних, що дає змогу за певного розміру вхідного масиву скопіювати його в локальну пам'ять мультипроцесора та повинно значно пришвидшити виконання алгоритму.

Код функції devMatrixMultyFFT:

```
template <int SIZE>
__global__ void devMatrixMultyFFT( int* M, float* X, float* Y, int colSize, int rowSize){
	int dataShift = blockIdx.x * rowSize; // Обчислення індексів для кожного ядра
	int process_count = rowSize / blockDim.x;
	int matrixStart = process_count * threadIdx.x;
	int matrixEnd = matrixStart + process_count;
	int dataStart = dataShift + matrixStart;
	int dataEnd = dataStart + process_count;
	__shared__ float local_data[SIZE];   //Копіювання вхідних даних у локальну пам'ять
	int pos = dataStart;
	for(int i = matrixStart; i < matrixEnd; ++i){
		local_data[i] = X[pos++];
	} __syncthreads();
	int p1, p2;                          // Виконання алгоритму ШПФ
	int shift = 0;
	for(int i = 0; i < colSize; ++i){
		for(int j = matrixStart; j < matrixEnd; j+=2){
			p1 = M[shift + j];
			p2 = M[shift + j + 1];
			// операція "метелик" над
			// local_data [p1] та local_data [p2]
		}
		shift += rowSize;
		__syncthreads();
	}   __syncthreads();
	pos = dataStart;     // Копіювання результату в глобальну пам'ять
	for(int i = matrixStart; i < matrixEnd; ++i){
		Y[pos++] = local_data[i];
	}
}
```


Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

5. Порівняння реалізацій алгоритму

Запуск створених програм виконувався на двох системах: персональному комп’ютері та ноутбуці. Характеристики відеокарт обох систем подано в табл. 1. ЦП першої системи Intel Core2Duo 1.8 GHz, другої – AMD Phenom II 2X 3.1 GHz.

Таблиця 1

Характеристики систем

Система	1	2
Центральний процесор		
Назва	Intel Core2Duo	AMD Phenom II 2X
Частота, ГГц	1.8	3.1
Відеокарта		
Назва	GeForce 8600M GS	GeForce GTX 560
ГП	G86	GF114
Обчислювальні можливості (Compute capability)	1.1	2.1
Кількість мультипроцесорів	2	7
Кількість ядер в мультипроцесорі	8	48
Загальна кількість ядер	16	336
Число 32-бітних регістрів у мультипроцесорі, К	8	32
Розподілена пам’ять у мультипроцесорі, КБ	16	48
Локальна пам’ять виконуваного потоку, КБ	16	512
Глобальна пам’ять констант, КВ	64	

Для обох тестів вхідний масив має 65536 елементів, в першому виконано одне ШПФ над усім масивом, в другому масив поділено на 64 частини і над кожною виконано відповідне перетворення. В другому тесті розмір підмасиву – 1024 елементи типу float, це 1024 * 4 = 4096 Кб. Отже, у другому тесті використано внутрішню пам’ять мультипроцесорів відеокарт. У табл. 2 подано конфігурації ядер ГП.

Таблиця 2

Конфігурації ядра ГП

Тест	Система 1	Система 2
1	2 блоки по 512 потоків	4 блоки по 1024 потоки
2	64 блоки по 256 потоків	64 блоки по 512 потоків

Таблиця 3

Результати першого тесту (для вхідного масиву 1 x 65536)

Функція	Час виконання, мс	
	Система 1	Система 2
recurseFFT	40,62	21,62
hostMatrixFFT	38,73	21,33
devMatrixFFT	12,4	1,05

З результатів першого тесту (табл. 3) видно, що час виконання hostMatrixFFT і recurseFFT практично однаковий, однак алгоритм hostMatrixFFT дає змогу розпаралелити розрахунки на декілька ядер ЦП, що зменшить час виконання. На першій системі перевага паралельного виконання на ГП порівняно з рекурсивним на ЦП досягла 40,62/12,4 = 3,27. На другій системі – 21,62/1,05 = 20,5.

Результати другого тесту наведено в табл. 4.

Таблиця 4

Результати другого тесту (для вхідного масиву 64 x 1024)

Функція	Час виконання, мс	
	Система 1	Система 2
recurseFFT	25,8	14,5
hostMatrixFFT	24,4	13,61
devMatrixMultyFFT	2,5	0,084

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Результати другого тесту показали велику перевагу системи ЦП з ГП, яка реалізує поданий потоковим графом алгоритм над ЦП, який реалізує рекурсивний алгоритм. Для першої системи перевага 25,8/2,5 = 10,3; для другої – 14,5/0,084 = 172,6.

Висновки

Проведено огляд технологій GPGPU та особливості написання програм при їх використанні. Досліджено реалізацію програм з використанням способу виконання поданих потоковим графом алгоритмів на графічних процесорах фірми nVidia.

За результатами тестування підтверджено високу ефективність способу.

Дослідження показали, що цей спосіб дає кращі результати у разі застосування його на вхідних даних, розмір яких дає змогу завантажити їх в локальну пам’ять мультипроцесорів відеокарти, оскільки локальна пам’ять мультипроцесора швидша за глобальну пам’ять відеокарти.

Також дослідження показали перспективу розвитку способу виконання поданих потоковим графом алгоритмів на центральних процесорах. Виконання тестової програми, якою подано алгоритм обчислення за цим способом, показало приблизно однакові результати з традиційним способом виконання алгоритму. Оскільки потоковий граф дає змогу розпаралелити виконання алгоритму, це повинно дати ефект і на багатоядерних та багатопотокових центральних процесорах.

1. Zibula A. General Purpose Computation on Graphics Processing Units (GPGPU) using CUDA // within the seminar Parallel Programming and Parallel Algorithms (Winter Term 2009/2010). 2. NVIDIA “*CUDA Architecture Overview. Introduction & Overview.*“*http://developer.download.nvidia.com/ /compute/cuda/docs/CUDA_Architecture_Overview.pdf, 2009. 3. NVIDIA* “*NVIDIA CUDA CProgramming Guide*“ *2011. 4. Kowalik Janusz, Puzniakowski Tadeusz. Using OpenCL: Programming Massively Parallel Computers (Advances in Parallel Computing) // Har/Cdr. – 2012. 5. Jason Zink Practical rendering and computation with Direct3D 11 // CRC Press. – 2011. 6. Мельник А. О. Спеціалізовані комп*’*ютерні системи реального часу / А. О. Мельник – Львів: Нац. ун-т* “*Львівська політехніка*“*, 2002. – 60 с.*

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua