УДК 519.688

І.П. Струбицька

Дослідження ефективності обчислень загального призначення на графічному процесорі

В статье проведено тестирование для сравнения времени выполнения одних и тех же программ на центральном и графическом процессорах, в частности умножения матриц. Для решения задачи использована технология CUDA. Вычислены производительность, ускорение и эффективность параллельной системы.

The test to compare the time of execution of the same programs at the central and graphics processors, matrix multiplication in particular, is conducted in the article. CUDA technology is used to solve the problem. Productivity, speeding and efficiency of the parallel system is calculated.

Ключевые слова: *распараллеливание, параллельные вычисления, графический процессор, GPU, CUDA, умножения матриц, дискретные динамические модели, метод оптимизации.*

Вступ

Проблема ефективності обробки великих об'ємів даних є однією з актуальних на сьогоднішній день. Багатоядерні процесори можуть виконувати одночасно лише декілька потоків, при цьому більш висока продуктивність має іноді дуже велику ціну.

На даний момент на ринку графічні процесори випускають дві компанії NVIDIA і AMD. Є ще Intel, проте він спеціалізується випуском GPU (graphics processing unit) для вбудованих відеокарт. AMD FireStream — це набір апаратних і програмних технологій, які дозволяють використовувати графічні процесори AMD спільно з центральним процесором для прискорення багатьох додатків. NVIDIA CUDA є аналогом технології AMD FireStream тільки для процесорів компанії NVIDIA.

Tesla — це сімейство обчислювальних систем NVIDIA на основі GPU з архітектурою CUDA, які можна використовувати для наукових і технічних обчислень загального призначення. Tesla не може повністю замінити звичайний універсальний процесор, але дозволяє використовувати обчислювальний ресурс множини своїх ядер для розв'язку ресурсномістких задач. Перевагами цих процесорів є велика енергоефективність, недоліком — менша універсальність. GeForce — сімейство GPU і чіпсетів материнських плат компанії NVIDIA, яке орієнтується на споживацький ринок. GeForce переважно використовується у відеоадаптерах для персональних і переносних комп'ютерів. AMD FireStream — це потоковий процесор, розроблений компанією AMD, призначений для збільшення ефективності розв'язку задач, з високою степінню паралелізму.

Відповідно до співвідношення ціна/якість було вибрано графічний процесор NVIDIA GeForce.

NVIDIA забезпечила підтримку програмування GPU на C, C++, Fortran, OpenCL и DirectCompute.

Технологія NVIDIA CUDA — це фундаментально нова архітектура обчислень на графічних процесорах, яка призначена для вирішення комплексу обчислювальних задач споживачів, бізнесу і технічної індустрії.

Ідея застосування швидких шейдерів графічних процесорів для проведення загальних обчислень, зародилася декілька років тому. На початках використовувалися різні технології на базі OpenGL і DirectX, які полягали в тому, що у GPU під видом текстур зображення передають дані, а під видом шейдерів функції для їх обробки. Недоліком таких вирішень була висока складність програмування і невелика швидкість обміну даними між графічним процесором і основною системою в напрямку від GPU до системи.

На поточний момент обчислення на графічних процесорах з технологією CUDA — це інноваційне поєднання обчислювальних особливостей нового покоління графічних процесорів NVIDIA, які обробляють відразу тисячі потоків з високим рівнем інформаційного завантаження, які доступні через стандартну мову програмування C.

Сьогодні компанія NVIDIA пропонує підтримку обчислень на всіх рівнях: апаратному (універсальні процесори GPU, висока швидкість обміну даними), драйверному (використання універсальних механізмів не прив'язаних до конкретних технологій), користувацькому (розробка бібліотек, компіляторів і SDK з прикладами програм і документацією) [1].

Постановка задачі

Щоб оцінити наскільки добре дана технологія підходить для задач чисельного моделювання, проведемо власне тестування для порівняння часу виконання одних і тих самих програм на

центральному і графічному процесорах. Усі тести були проведені на наступному апаратному забезпеченні:

- Процесор: Core2Duo E8400, 3ГГц;
- Графічний процесор NVIDIA GeForce GTS250, 1024 Мб, (16 мультипроцесорів по 8 процесорів).

Як приклад для тестування було вибрано задачу множення квадратних матриць.

Для множення матриць на центральному процесорі вибрано класичний підхід (множення стрічки на стовпець) за формулою:

$$C_{ij} = \sum_i A_{ik} B_{kj}.$$

Матриці можуть бути транспоновані, при цьому змінюється порядок читання з пам'яті, що дуже важливо для обчислень на центральному процесорі, і впливає на ефективність. Також задачу було апробовано на різних типах даних (float і int). Результати виконання тесту на центральному процесорі (усереднений час на основі 10 запусків) приведені у таблиці 1 (час вказано в секундах).

Як і очікувалося, обчислення з даними типу float на центральному процесорі проводились довше. Найшвидше перемножуються матриці за методом , так як при цьому здійснюється послідовна вибірка даних з пам'яті, власне варіант $A^T B$ обчислюється довше за все [2].

Як тестову задачу для розрахунку на графічному процесорі було вибрано два алгоритми: класичний (аналогічно алгоритму на центральному процесорі) і оптимальний (запропонований розробниками CUDA) [3].

При класичному алгоритмі для знаходження одного елементу множення двох матриць потрібно зчитати $2N$ значень з глобальної пам'яті і виконати $2N$ арифметичних операцій. Тобто основним обмеженням ефективності такого алгоритму є зчитування з глобальної пам'яті, а не самі обчислення [4].

Алгоритм, який запропонований розробниками CUDA [3-4], полягає у використанні спільної пам'яті. Матриці A і B розбиваємо на блоки розміром 16x16. Кожний блок обчислює одну 16x16 матрицю C' (під матрицю шуканого добутку):

$$C' = A'_1 * B'_1 + A'_2 * B'_2 + \ldots + A'_{N/16} * B'_{N/16}.$$

Тобто обчислення матриці C' здійснюється за $N/16$ кроків. На кожному кроці у спільну пам'ять вивантажується одна підматриця A і одна підматриця B. Оскільки кожний потік блоку вивантажує один елемент із кожної підматриці, то кожний потік робить тільки два звернення до глобальної пам'яті за один крок. Тобто при множенні матриць на кожний елемент добутку C потрібно виконати $2N/16$ зчитувань з глобальної пам'яті. При цьому кількість арифметичних операцій не змінилась.

У таблиці 2 приведено загальний час роботи програми (усереднення на основі 100 запусків, в секундах), а в дужках час, що затрачений на процеси не пов'язані з обчисленнями (виділення пам'яті у графічному процесорі, передача даних та інше). Розпаралелення проводились з конфігурацією потоків 16x16.

Для множення матриць великих розмірів класичним алгоритмом необхідний великий об'єм оперативної пам'яті.

Таблиця 1. Час множення матриць на центральному процесорі

Порядок матриць	AB		$A^T B$		AB^T	
	int	**float**	**int**	**float**	**int**	**float**
512x512	0,532	1,031	0,770	1,047	0,097	0,922
1024x1024	4,801	8,470	8,050	9,328	0,767	7,403
2048x2048	60,500	88,256	85,500	100,750	7,703	60,453

Таблиця 2. Час множення матриць на графічному процесорі

Порядок матриць	Класичний алгоритм		Алгоритм CUDA	
	int	**float**	**int**	**float**
512x512	0,19 (0,037)	0,183 (0,028)	0,038 (0,028)	0,035 (0,028)
1024x1024	1,27 (0,064)	1,25 (0,066)	0,128 (0,06)	0,096 (0,06)
2048x2048	9,55 (0,195)	9,45 (0,195)	0,713 (0,183)	0,473 (0,2)
3072x3072	39,43 (0,38)	38,68 (0,38)	2,19 (0,38)	1,38 (0,4)
4096x4096			4,89 (0,66)	2,84 (0,7)
5120x5120			9,2 (1,02)	5,2 (1,08)
6144x6144			15,84 (1,48)	9,78 (1,54)
7168x7168			24,57 (2,0)	13,46 (2,09)

Обчислення продуктивності, прискорення та ефективності паралельної системи

Обчислимо продуктивність обчислювальних систем – це кількість операцій, які виконуються за деякий фіксований період часу. Це як правило, 1 секунда. Загальна кількість операцій з плаваючою комою при множенні матриць пропорційна $2N^3$. Тобто продуктивність оцінюють за наступним співвідношенням:

$$R = \frac{2N^3}{T},$$

де R – продуктивність, N – порядок системи, T – час множення матриць.

Продуктивність системи для множення матриць подана у таблиці 3 і приведена на рисунку 1.

Таблиця 3. Продуктивність системи

Порядок матриць, N	Продуктивність, R (GFLOPS)
512x512	7,67
1024x1024	22,37
2048x2048	36,32
3072x3072	42,02
4096x4096	48,39
5120x5120	51,62
6144x6144	47,43
7168x7168	54,72
7168x7168	245,53

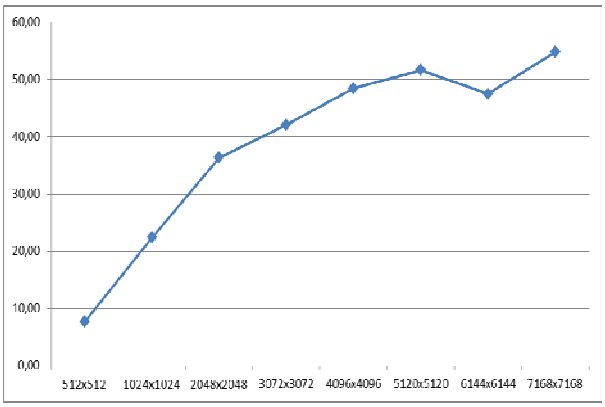

Рис. 1. Залежність продуктивності системи від порядку матриць

Для паралельного множення матриць приведемо показники отриманого прискорення часу розв'язку задачі порівняно з часом роботи послідовної програми:

$$S = \frac{t_c}{t_p},$$

де S – прискорення, t_c – час виконання послідовної програми, t_p – час виконання паралельної програми.

Прискорення паралельного алгоритму CUDA порівняно з класичним послідовним алгоритмом подане у таблиці 4 і на рисунку 2.

Таблиця 4. Прискорення системи

Порядок матриць, N	Прискорення, S
512x512	5,228571429
1024x1024	13,02083333
2048x2048	19,97885835
3072x3072	28,02898551

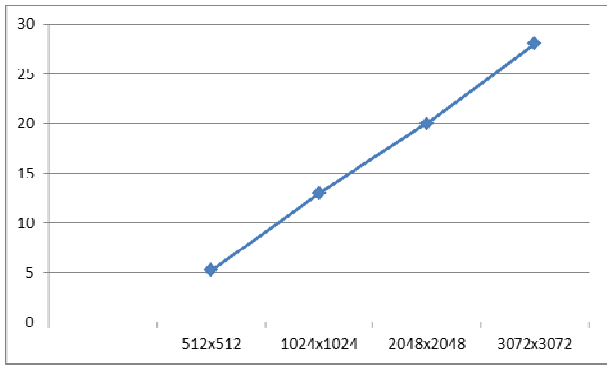

Рис. 2. Залежність прискорення системи від порядку матриць

Ефективність паралельного алгоритму – це відношення його прискорення до числа процесорів, на якому це прискорення отримане:

$$E = \frac{S}{p},$$

де E - ефективність, S - прискорення, p - число процесорів.

Ефективність системи при паралельному алгоритмі множення матриць подана у таблиці 5 і на рис. 3.

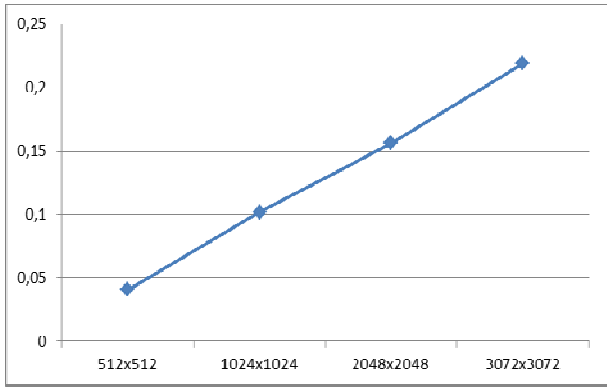

Рис. 3. Залежність ефективності системи від порядку матриць

Видно покращення продуктивності в 5 разів вже на класичному алгоритмі, ефективний ж дає приріст ефективності обчислень на порядок. Це пояснюється тим, що при послідовному алгоритмі багато часу було затрачено на звернення до глобальної пам'яті. А при паралельній програмі приблизно 80% часу припадає на самі обчислення.

Таблиця 5. Ефективність системи

Порядок матриці, *N*	Ефективність, *E*
512x512	0,040848214
1024x1024	0,10172526
2048x2048	0,156084831
3072x3072	0,218976449

Числа з плаваючою комою обчислюються швидше приблизно в 2 рази (можливо, це обумовлено архітектурою графічного процесора).

Додаткові обмеження накладає точність обчислень. Сучасні GPU оперують числами з подвійною точністю, проте продуктивність різко зменшуються. Однак, графічні процесори ще тільки розвиваються і у наступних версіях GPU може бути врахована ця проблема.

Це показує значний приріст продуктивності, значить, використання GPU для обчислень загального призначення видається перспективним при розв'язку таких задач, які використовують багато операцій з матрицями. Крім того, наукові застосування, які потребують високої інтенсивності обчислень, більше не будуть займати весь процесорний час. Обчислення в CUDA представляє платформу з високим рівнем продуктивності при вирішення задач. Наприклад, для задачі оптимізації параметрів дискретної динамічної моделі, суть якої полягає у наступному. Побудова математичної моделі у підході [6, 7] передбачає оптимізацію в сенсі мінімізації відхилення поведінки моделі від поведінки модельованого об'єкта, тобто мінімізація функції мети. З одного боку це дозволяє значно універсалізувати даний підхід, а з іншого – приводить до появи складних оптимізаційних задач, які важко розв'язати навіть з використанням сучасних засобів обчислювальної техніки. Задача знаходження мінімуму нелінійної функції багатьох змінних є доволі складним завданням у загальному випадку.

При побудові дискретних динамічних моделей функція мети має чітко виражений яровий характер з великою кількістю локальних мінімумів. З погляду придатності до вирішення таких задач найкращими характеристиками вирізняється метод напрямного конуса Растригіна [8, 9]. За допомогою цього методу можна провести цілеспрямований перебір локальних мінімумів, що прискорює знаходження глобального мінімуму функції мети. Відповідно, обчислювальна складність такої задачі буде досить велика. Для побудови якісної моделі потрібно використати значну кількість даних для обчислення значення функції мети. Отже, значними будуть і затрати машинного часу на обчислення значень відповідних функцій. Відповідно, для зменшення обчислювальних затрат доцільним є розпаралелення обчислювального процесу.

Тернопільский національний економічний університет

У задачі розпаралелення алгоритму оптимізації параметрів дискретних динамічних моделей на масивно-паралельних процесорах [5] запропоновано використати метод дрібнозернистого розпаралелення для деяких блоків алгоритму. Отже, у даному випадку можна використати паралельний алгоритм множення матриць для відповідних обчислень, оскільки він зарекомендував себе як ефективний.

Висновки

У результаті цього дослідження було:

1. Проведено порівняння класичного послідовного алгоритму множення матриць, який виконується на центральному процесорі, і паралельного алгоритму CUDA, який виконується на GPU.

2. Обчислено продуктивність, прискорення і ефективність паралельної системи.

3. Проведено чисельні експерименти, які показали, що прискорення паралельної системи для матриць 512x512 становить 5,23 рази, а для матриць 3072x307 – 28,03.

Література

1. CUDA Zone — The resource for CUDA developer. C [Електронний ресурс]. — Режим ступу: http://www.nvidia.com/object/cuda_home_new.html
2. Использование видеокарт для вычислений [Електронний ресурс]. – Режим доступу: http://GPGPU.ru
3. NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Version 2.0, – 2008. – 107 p.
4. *А.В. Боресков,* А.А. Харламов. Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010. – 232 с.
5. *Козак Ю.Я.,* Стахів П.Г., Струбицька І.П. Розпаралелення алгоритму оптимізації параметрів дискретних динамічних моделей на масивно-паралельних процесорах // Відбір і обробка інформації. - 2010. – Вип. 32 (108). – С. 126-130.
6. *Люнг Л.* Идентификация систем. Теория для пользователя - Пер. с англ.; Под ред. Я. З. Цыпкина. – М.: Наука. Гл. ред. физ.-мат. лит., 1991.- 432 с.
7. *Семенов А.Д.*, Артамонов Д.В., Брюхачев А.В. Идентификация объектов управления: Учебн. пособие. - Пенза: Изд-во Пенз. гос. ун-та, 2003.-211 с.
8. *Растригин Л.А*. Адаптация сложных систем.- Рига: Зинатне, 1981.- 375 с.
9. *Стахів П.Г.,* Козак Ю.П. Побудова макромоделей електромеханічних компонент з використанням оптимізації // Технічна електродинаміка. – 2001. – №4. – С.33–36.