УДК 004.274

С.Д. Погорілий, д-р техн. наук, проф.,
Б.О. Семьонов, студент
Київський національний університет ім. Тараса Шевченка
sdp77@i.ua
bohdan.semonov@gmail.com

Дослідження програмної бібліотеки Linpack на архітектурі CUDA мовою програмування Python

Реалізовано послідовну версію методу Гауса (розв'язання систем лінійних алгебраїчних рівнянь) та її паралельну версію на ядрах архітектури CUDA на мові програмування Python. Інстальовано пакет Linpack від розробників компанії NVidia та проведено ряд досліджень: порівняння швидкості виконання різних реалізацій методу Гауса та пакету Linpack.

Ключові слова: GPGPU, CUDA, SIMD, SIMT, Python, PyCUDA, Anaconda, Linpack, метод Гауса.

Вступ

Основною особливістю сучасних графічних відеоадаптерів, які використовують графічні процесори (GPU), є наявність набору потокових мультипроцесорів (SM), що використовувалися раніше лише в алгоритмах і задачах, пов'язаних з обробкою графічних зображень. Програмні технології (інтерфейси), що застосовуються програмістами для створення програм такого напрямку, використовують пам'ять відеоадаптера для розміщення структур даних, таких як текстури, буфери, визначають конвеєр обробки, кожен етап якого відповідає за свої специфічні дії: растеризацію, інтерполяцію, блендинг, теселяцію [4, 5] тощо.

Технологія обчислень загального призначення на графічних процесорах (GPGPU) ґрунтується на використанні великої кількості процесорів GPU, що працюють паралельно, для обробки даних за допомогою алгоритмів загального призначення (наукових чи інших, але не обов'язково пов'язаних з обробкою зображень).

Потоковий процесор на GPU має простішу структуру, ніж вузол CPU. Тобто такі вузли менш універсальні і виконують менший набір функцій, аніж вузли процесора. Проте, оскільки їхня кількість велика, то для певного набору задач можна досягти суттєвого приросту в швидкодії. Найкращого прискорення вдається досягти для алгоритмів, що підтримують концепцію паралелізму за даними (один паралельний потік обробляє свою область у пам'яті). Тому зазвичай GPGPU застосовують у наступних сферах: обробка зображень (Reduction, Histogram, Fast Fourier Transform, Summed Area Table); обробка відеоданих (Transcode, Digital Effects, Analysis); лінійна алгебра; моделювання (Technical, Finance, Academic, Some Databases) тощо.

Кожен із програмних інтерфейсів створення GPGPU застосувань має свій рівень абстракції по відношенню до апаратної реалізації цільових архітектур. Так, NVidia CUDA позиціонується фірмою-розробником як програмна модель і платформа паралельних обчислень загального призначення. Первісно для цієї технології було створено середовище розробки CUDA SDK, в основі якого лежить мова програмування C з деякими розширеннями [2, 12, 15]. Сьогодні архітектура CUDA набула значної популярності в комп'ютерних системах при обчисленні великих об'ємів даних. Тому є доцільним розширення переліку мов програмування, здатних підтримувати дану програмну модель.

Наразі таку можливість має мова програмування Python, яка набула широкого застосування у програмуванні чисельних методів, і є популярною через порівняну простоту синтаксису та зручність читання написаного коду.

Python – об'єктно-орієнтована мова надвисокого рівня, яка підтримує множинне успадкування, перевизначення інфіксних операторів, причому можна перевизначити операцію як для лівого операнда, так і для правого; починаючи із версії 2.1 є повне перевизначення операторів порівняння (механізм rich comparison для об'єктів, що підтримують часткове упорядкування, наприклад, матриць). У Python є обробка виняткових ситуацій і механізм їх перехоплення; таким чином програміст може побудувати правильну обробку помилок і створити надійну програму. Вбудовані механізми інтроспекції дозволяють опитувати інтерфейси об'єктів під час виконання програми. Наприклад, можна дізнатися кількість та імена параметрів функції; цю інтроспекцію використовує Zope, щоб підготувати правильний список параметрів функції при виклику її з web. Python портований та працює майже на всіх відомих платформах – від мобільних платформ до мейнфреймів.

Існують порти під Microsoft Windows, всі варіанти UNIX (включаючи FreeBSD та GNU/Linux), Plan 9, Mac OS та Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, AS/400 та навіть OS/390, Symbian та Android.

Подальші таблиці 2 – 5 ілюструють переваги Python у порівнянні із іншими інтерпретованими мовами програмування. Введемо наступні позначення:

Таблиця 1. Умовні позначення

+	Вказана можливість присутня
-	Вказана можливість відсутня
+/-	Можливість підтримується не повністю
-/+	Можливість підтримується дуже обмежено
?	Немає даних
N/A	Постановка питання не прийнятна до мови

Для мови Python відомо такі програмні інтерфейси: PyCUDA та Anaconda, що дозволяють виконати розпаралелене застосування на ядрах CUDA.

Метою дослідження роботи є програмні інтерфейси для архітектури CUDA, а саме: для мови програмування Python. Предметом дослідження є «паралельний» на ядрах CUDA метод розв'язування систем лінійних алгебраїчних рівнянь (метод Гауса) та пакет Linpack від розробників фірми NVidia, в основі якого лежить цей метод, і порівняння їхньої швидкодії.

Таблиця 2. Порівняння за парадигмами

	Python	Perl	Ruby	Java	JavaScript
Імперативна	+	+	+	+	+
Об'єктно-орієнтована	+	+	+	+	+
Функціональна	+	+	+	-	+
Рефлексивна	+	+	+	-/+	+
Узагальнене програмування	-	+	-	+	+
Логічна	-	-	-	-	-
Процедурна	+	+	+	-	+
Розподілена	-/+	-	-/+	-	-

Таблиця 3. Порівняння за типами та структурами даних

Можливість	**Мова**				
	Python	Perl	Ruby	Java	JavaScript
Кортежі	+	+	+	-	-
Алгебраїчні типи даних	N/A	N/A	N/A	-	N/A
Багатомірні масиви	+/-	+/-	+/-	+/-	+/-
Динамічні масиви	+/-	+/-	+/-	+/-	+/-
Асоціативні масиви	+	+	+	+/-	+
Контроль границь масивів	+	N/A	?	+	N/A
Цикл foreach	+	+	+	+	+
Спискові включення	+	?	?	-	-
Цілі числа довільної довжини	+	+	+	+	-+

Таблиця 4. Порівняння за об'єктно-орієнтованими можливостями

Можливість	**Мова**				
	Python	Perl	Ruby	Java	JavaScript
Інтерфейси	+	+/-	?	+	?
Mixins	+	?	+	+	?
Перейменування членів при спадкуванні	-	-/+	?	-	?
Множинне спадкування	+	+	-	-	?
Рішення конфлікту імен при множинному спадкуванні	+	+	N/A	N/A	?

Таблиця 5. Порівняння за функціональними можливостями

Можливість	**Мова**				
	Python	Perl	Ruby	Java	JavaScript
First class functions	+	+	+	-	+
Анонімні функції	+/-	+	+	-	+
Лексичні замикання	+	+	+	+	+
Часткове застосування	+	-	+	-	-
Карирування функцій	+	+	+	-	+

Аналіз архітектури паралельних обчислень CUDA

Архітектура CUDA [2, 3] – це програмна модель, яка включає опис обчислювального паралелізму та ієрархічної структури пам'яті безпосередньо в мову програмування. З точки зору програмного забезпечення, реалізація CUDA є багатоплатформовою системою компіляції та виконання програм, частини яких працюють на CPU і GPU. CUDA призначена для розробки GPGPU додатків без прив'язки до графічних API і підтримується всіма GPU NVidia, починаючи з серії GeForce 8.

У ряді можливостей нових версій CUDA простежується тенденція до поступового перетворення GPU в самодостатній пристрій, який повністю може замінити звичайний CPU за рахунок реалізації деяких системних викликів (в термінології GPU системними викликами є, наприклад, виділення пам'яті та звільнення, реалізоване в CUDA 3.2) і додавання полегшеного енергоефективного CPU-ядра в сам GPU (архітектура Maxwell та Pascal) [3].

Важливою перевагою CUDA є використання для програмування GPU мов високого рівня. На даний момент існують компілятори C++ і Fortran, спеціальний прискорювач для мови програмування Python. Ці мови розширюються невеликою кількістю нових конструкцій: атрибути функцій і змінних, вбудовані змінні і типи даних, оператор запуску ядра.

Програмні інтерфейси для архітектури CUDA мовою програмування Python

PyCUDA – це бібліотека для роботи з технологією CUDA мовою програмування Python, яка поширюється під вільною ліцензією MIT.

Python – це мова програмування з динамічною типізацією і підтримкою об'єктно-орієнтованої, структурної, функціональної, імперативної і аспектно-орієнтованої парадигм програмування. Python є популярним засобом розробки додатків для веб (web), зважаючи на такі властивості, як, наприклад, хороша читабельність коду, інтерактивний режим розробки (без компіляції), динамічна типізація.

Однак множина сфер застосування значно ширша. Python використовується в графіці (PyOgre, PyOpenGL), розпізнаванні зображень (Python для OpenCV, Python для OpenVIDIA), робототехніці (в Robotics Operating System), чисельних методах та інших областях [1, 6, 8, 14]. Наявність Python-інтерфейсів у безлічі прикладних бібліотек робить цю мову відмінним засобом організації взаємодії компонентів складених програм.

Існують готові бібліотеки для наукових застосувань: для побудови графіків на площині і в просторі, для чисельних обчислень, MPI тощо.

PyCUDA можна інсталювати як додатковий пакет до існуючої інсталяції Python 2.7, 3.0 або 3.5. PyCUDA доступний на всіх операційних системах, де доступна розробка для CUDA: Linux, Windows і OS X. Програми на PyCUDA можна налагоджувати за допомогою CUDA-GDB:

cuda-gdb -args python -m pycuda.debug demo.py [3]

Математичні бібліотеки зі складу CUDA Toolkit також мають готові інтерфейси в оточенні Python – *scikits.cuda*.

PyCUDA використовує ту ж програмну модель, що і CUDA, внаслідок чого в застосуванні відбувається явне виділення пам'яті на GPU і копіювання в неї даних. Потім створюється об'єкт типу Module, який асоційований з модулем CUBIN. Як аргумент SourceModule в потрійних лапках у вигляді символьного рядка передається програмний код ядра на CUDA C. Вже під час виконання цей код передається компілятору nvcc, який і створює CUBIN (якщо його ще немає в кеші). Функції з модуля можна отримати по символьному імені та потім викликати. Оброблені на GPU дані копіюються назад на хост і виводяться в консоль.

Запустити цю програму можна або безпосередньо введенням коду в інтерпретатор Python, або, помістивши в окремий файл, командою: *python example1.py*.

Таким чином, на відміну від звичайного компільованого застосування, програма на PyCUDA складається з двох частин: GPU-частина, як і раніше компілюється, а для «склеювання» низькорівневих високопродуктивних блоків застосовується інтерпретована скриптова мова. Разом ці дві частини засобами Python об'єднані в гібридну програму (рис. 1) (де nvcc – компілятор CUDA; cubin – бінарний файл), що може спрости-

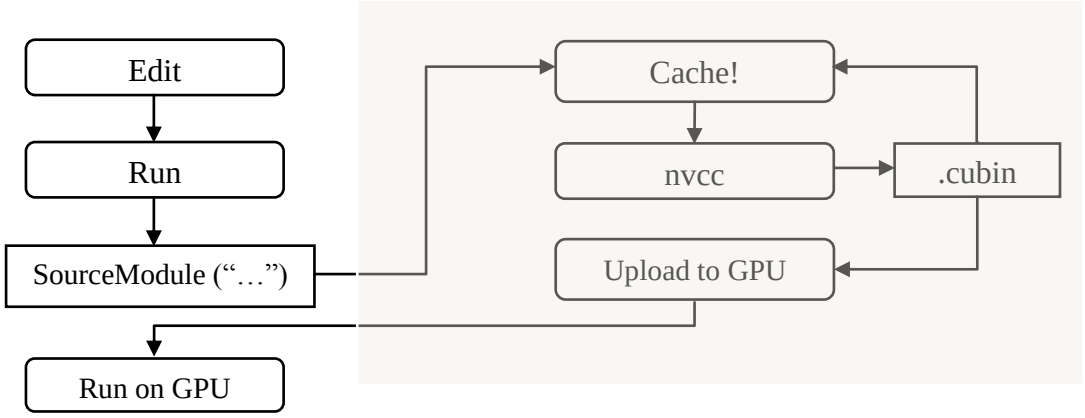

Рисунок 1 – Процес розробки на PyCUDA

ти розробку, не погіршивши при цьому продуктивність ядер. За рахунок додаткової інтеграції API CUDA, всі помилки, які виникають у ході роботи застосування, перетворюються у виключення Python.

Anaconda. CUDA Python – це деякий «гібрид» інтерпретованої мови програмування Python та «прискорювача» Numba, що є частиною платформи Anaconda, яка розкриває всі можливості NVidia відеоадаптерів. Numba надає можливість прискорити застосування за допомогою функцій високої продуктивності, написаних безпосередньо мовою Python. Програми на Python, метою яких є складні математичні обрахунки, можуть бути виконані з порівняною продуктивністю до застосунків, написаних мовами C, C++ і Fortran [10].

Numba працює шляхом створення оптимізованого машинного коду з використанням інфраструктури компілятора LLVM під час виконання або статично (за допомогою доданого інструменту pycc). Numba підтримує компіляцію Python, що дозволяє застосуванню працювати на будь-яких апаратних засобах GPU, і призначений для інтеграції з Python.

Платформа Anaconda є провідною платформою відкритих наукових даних, що працює на Python. Версія платформи Anaconda з відкритим вихідним кодом – це високопродуктивний дистрибутив Python та R і включає в себе більше ста найпопулярніших пакетів для наукових досліджень, написаних мовами програмування Python, R і Scala. Крім того, платформа надає доступ до більш ніж 720 пакетів, які можуть бути легко встановлені за допомогою системи управління пакетами conda. Anaconda має BSD ліцензію, яка дає дозвіл на використання даної платформи на

комерційній основі [9, 10].

CUDA JIT компілятор є низькорівневою точкою входу в множину функцій NumbaPro. Він переводить функції Python в PTX код, який виконується на обладнанні CUDA. JIT декоратор застосовується до функцій Python, написаних на діалекті Python для CUDA. NumbaPro взаємодіє з драйвером API CUDA, щоб завантажити PTX на пристрій CUDA і виконати [11].

Більшість різних відкритих функцій та методів CUDA API знаходяться в модулі numba.cuda.

CUDA-ядра (kernel) та функції, які повинні виконатися на GPU, компілюються за допомогою jit- або autojit декораторів:
@jit(argtypes=[float32[:,:], float32[:,:], float32[:,:]], target='gpu').

Набір вбудованих CUDA-функцій використовується для ідентифікації поточного потоку виконання. Ці вбудовані функції мають сенс тільки всередині CUDA-ядра.

Далі варто відправити всі необхідні дані на відеоадаптер. Обмін інформацією між хостом (host) та пристроєм (device) є асинхронним, тому необхідно його синхронізувати. Для цього варто застосовувати CUDA-потік (stream). Він є чергою команд для пристрою CUDA. Після створення потоку виклики CUDA API стають автоматично синхронними.

Для синхронізації роботи ниток в середині CUDA-ядра використовується метод cuda.syncthreads(). Його принцип дії: наступна інструкція виконається тільки тоді, коли всі нитки «прийдуть» в дану точку, де викликаний даний метод.

Пакет Linpack для тестування продуктивності комп'ютерних систем

Пакет Linpack – програмна бібліотека, написана на мові програмування Фортран, яка містить набір підпрограм для розв'язування систем лінійних алгебраїчних рівнянь. Linpack була розроблена для роботи на суперкомп'ютерах, які використовувались в 1970-х – на початку 1980-х років. Існують версії бібліотеки для чисел з плаваючою крапкою з різною точністю і для комплексних чисел [13].

Також під назвою Linpack часто розуміють тести продуктивності Linpack. Спочатку тест був опублікований в додатку «B» у документації бібліотеки і призначався для грубої екстраполяції часу роботи бібліотеки. Існують варіанти тесту: linpack100 (матриця 100 на 100 – 1977 рік), linpack1000 (матриця збільшена до 1000 елементів в кожному вимірі – 1986 рік), linpack parallel (1000 елементів, паралельна обробка) і HPL (довільні розміри, перші версії випущені в 1991-1993 роках) – популярний тест продуктивності, призначений для оцінки продуктивності паралельних обчислювальних систем і створений на базі деяких функцій з бібліотеки Linpack.

Реалізація методу Гауса засобами мови Python

Найбільш класичним прямим методом розв'язування систем лінійних рівнянь є метод виключення невідомих, який пов'язують з ім'ям Гауса. Ідея алгоритму Гауса при розв'язуванні системи полягає в тому, що система зводиться до еквівалентної системи з верхньою трикутною матрицею (прямий хід). Із перетвореної таким чином системи невідомі знаходяться послідовними підстановками, починаючи з останнього рівняння перетвореної системи (зворотний хід).

Крім аналітичного розв'язку СЛАР, метод Гауса також застосовується для: знаходження матриці, оберненої до даної; визначення рангу матриці [7].

До переваги методу можна віднести наступне: для матриць обмеженого розміру менш трудомісткий в порівнянні з іншими методами; дозволяє однозначно встановити, сумісна система чи ні, і якщо сумісна, знайти її розв'язок; дозволяє знайти максимальне число лінійно незалежних рівнянь – ранг матриці системи.

Тест № 1. Реалізація методу Гауса на одному ядрі CPU. На рис. 2 наведено графік залежності часу роботи послідовної (тільки на одному ядрі процесора) реалізації алгоритму Гауса від розмірності матриці.

Тестування проводилося на комп'ютерній системі з наступними характеристиками:

- Linux RHE 7.2 Workstation;
- Intel Core i7-3612QM, 2.10 GHz;
- RAM 6GB.

Тест № 2. Реалізація методу Гауса на ядрах CUDA. Процес розв'язання СЛАР методом Гауса зводиться до послідовності однотипних обчислювальних операцій множення і додавання над рядками матриці, тому в основу паралельної реалізації алгоритму може бути покладено наступний принцип розпаралелювання: вибирається перший рядок матриці і від наступних рядків віднімається цей перший рядок, домножений на коефіцієнти так, щоб перші елементи кожного наступного рядка при відніманні перетворювалися на нулі. При цьому за віднімання кожної пари елементів відповідає окремий CUDA-потік. Після запуску функції-ядра всі перші елементи рядків матриці (крім першого) міститимуть нулі. Надалі вибирається підматриця, яка отримується виключенням першого рядка та першого стовпчика і дії повторюються (рис. 3).

Параметри відеоадаптера, на якому проводилося тестування:

- GPU – NVidia GeForce GT 640M;
- архітектура Kepler;
- кількість ядер CUDA – 384;
- тактова частота графічної підсистеми –

624 МГц;

• швидкість передачі даних пам'яті – 1800 МГц;

• відеопам'ять – 2048 МБ DDR3.

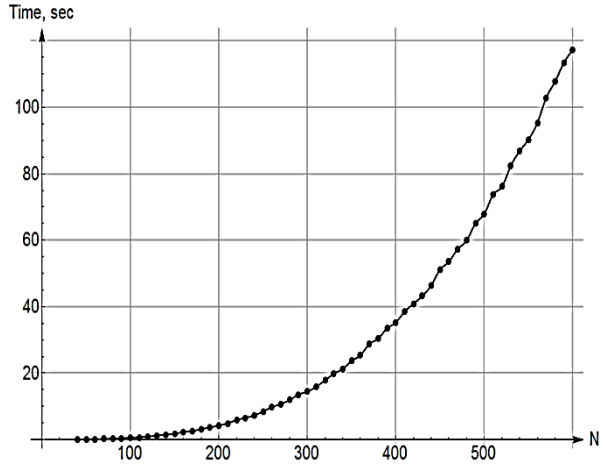

Рисунок 2 – Графік результатів тесту № 1

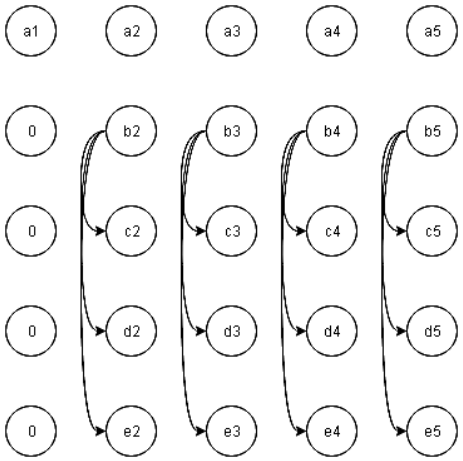

Рисунок 3 – Підхід до розпаралелювання алгоритму Гауса

На рис. 4 наведено графік залежності часу роботи алгоритму від розмірності матриці.

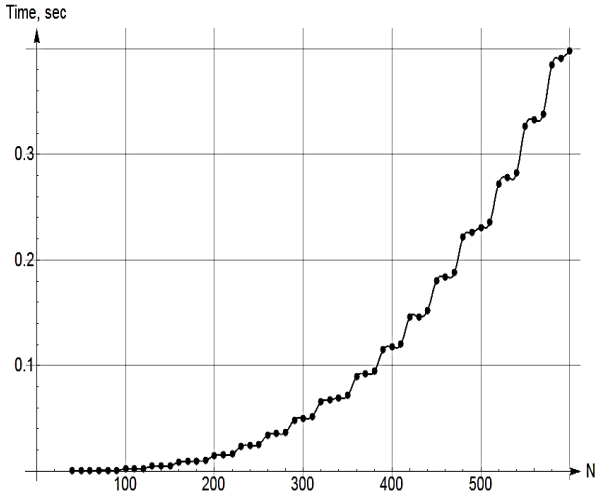

Рисунок 4 – Графік результатів тесту № 2

Методика вимірювань. Для вимірювання часу виконання алгоритму використовується клас default_timer з модуля timeit.

Вимірювання швидкодії «паралельного» алгоритму проводиться з урахуванням обміну даними між хостом та пристроєм та виклику ядра на GPU.

Прискорення визначається за допомогою формули: t_послідовний/t_паралельний .

Результати вимірювань записуються у файл за допомогою вбудованих засобів мови програмування Python: f = open('data.txt', 'r'),

де 'data.txt' – текстовий файл, з якого необхідно прочитати або куди треба записати;

'r' – режим роботи з файлом (запис/читання).

Порівняльна характеристика № 1. На рис. 5 зображений графік співвідношення часу виконання різних (послідовної та паралельної) версій методу Гауса в залежності від розмірності матриці.

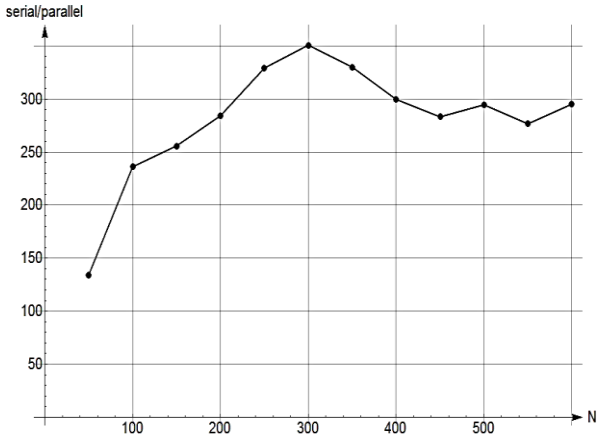

Рисунок 5 – Графік порівняльної характеристики № 1

Як можна побачити з графіку було досягнуто швидкого виконання «паралельного» методу Гауса у порівнянні з послідовною версією. Прискорення паралельної версії коливається від 100 до 350 разів.

Пакет Linpack від компанії NVidia для архітектури CUDA

Для інсталяції пакету Linpack від розробників компанії NVidia необхідна наявність, проінстальованих заздалегідь, пакетів:

• Операційна система Linux RHE або CentOS;

• Intel компілятор;

• Intel MKL;

• Openmpi;

• CUDA Toolkit;

• NVidia драйвер.

Тест № 3. Пакет Linpack від NVidia. На рис. 6 наведено графік залежності тестування, яке проведено за допомогою Linpack від NVidia.

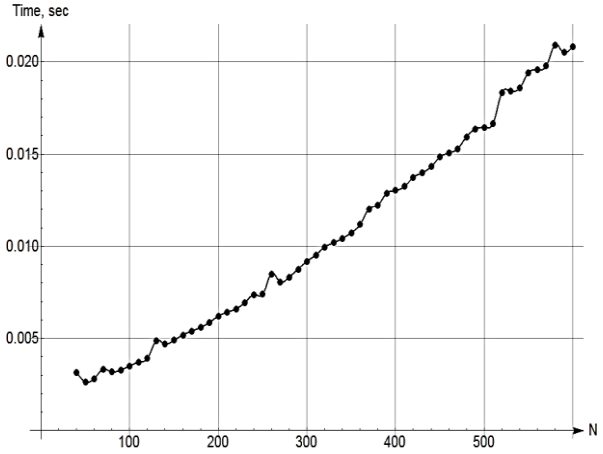

Рисунок 6 – Графік результатів тесту № 3

Порівняльна характеристика № 2. На рис. 7 зображений графік співвідношення часу виконання паралельної версії методу Гауса та пакету Linpack компанії NVidia від розмірності матриці.

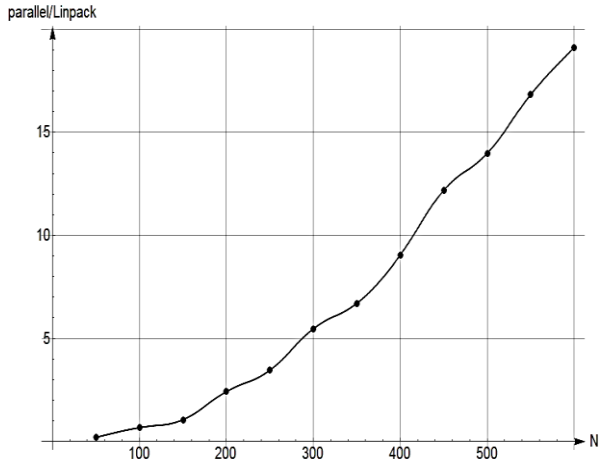

Рисунок 7 – Графік порівняльної характеристики № 2

З графіка видно, що прискорення Linpack у порівнянні з паралельною версією – приблизно порядок. Цей факт свідчить про доцільність використання мови програмування Python та платформи Anaconda для роботи з архітектурою CUDA.

Висновки

Тенденція розвитку GPGPU-технологій, свідчить про той факт, що GPU стає все більш незалежним пристроєм по відношенню до CPU. Це демонструють такі технології як динамічний паралелізм та динамічне виділення пам'яті. Результатом цього процесу є очікування появи найближчим часом універсальної архітектури GPU-CPU, що об'єднає функціональні можливості цих пристроїв.

У роботі розглянуто реалізацію інструментальних засобів технології GPGPU мовою програмування Python: програмні інтерфейси PyCUDA та Anaconda. Засоби CUDA, що створені фірмою NVidia, підтримують відеоадаптерами лише цієї фірми. З іншого боку, цей факт є і перевагою, оскільки, порівняно з іншими реалізаціями, інструментальні засоби CUDA підтримують специфічні для NVidia GPU інструкції, що в багатьох випадках може суттєво покращити швидкодію виконання алгоритму.

Проведений аналіз швидкодії виконання «паралельного» методу Гауса на ядрах CUDA мовою програмування Python, яка є, між іншим, інтерпретованою мовою, показав доцільність її використання на архітектурі CUDA. У ході дослідження було встановлено максимальне прискорення паралельної версії методу Гауса відносно послідовної – приблизно у 350 разів, що є хорошим показником, зважаючи на вищенаведені характеристики. Проте співвідношення швидкодії такого ж алгоритму, реалізованого в пакеті Linpack розробниками фірми NVidia є не таким оптимістичним. Тут «перемагає» Linpack від NVidia: близько порядку при матрицях великих розмірів (400 ÷ 600), а при невеликих (40 ÷ 200) – швидкість є порівняною. Це можна пояснити тим, що Linpack є відкомпільованим пакетом, в той час як Тест № 2 виконується в режимі інтерпретації.

Список літератури

1. А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий. Програмування числових методів мовою PYTHON. За редакцією чл.-кор. НАН України А. В. Анісімова. Підручник із грифом МОН України. ВПЦ «Київський університет», 2015 р. 640 с.

2. Боресков А. В. Основы работы с технологией CUDA / А. В. Боресков, А. А. Харламов. – ДМК-Пресс, 2010. – 232 с.

3. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учеб. пособие / А. В. Боресков и др. Предисл.: В. А. Садовничий. – М.: Издательство Московского университета, 2012. – 336 с.

4. Погорілий С. Д., Вітель Д. Ю., Верещинський О. А. Новітні архітектури відеоадаптерів. Технологія GPGPU. Частина 1. – К.: ВПЦ «Київський університет», 2012. – 13 с.

5. Погорілий С. Д., Вітель Д. Ю., Верещинський О. А. Новітні архітектури відеоадаптерів. Технологія GPGPU. Частина 2. – К.: ВПЦ «Київський університет», 2013. – 11 с.

6. Погорілий С.Д. Програмне конструювання. Підручник серії Автоматизація наукових досліджень за редакцією академіка АПН України Третяка О.В. ВПЦ Київський університет., 2-е видання, 2007, 438 с.

7. Програмування числових методів мовою Python: навч. посіб. / А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий, Є. В. Глушко; за ред. А. В. Анісімова. – К.: ВПЦ «Київський університет», 2013. – 463 с.

8. С.Д. Погорілий, Р.В. Білоус. Генетичний алгоритм розв'язання задачі маршрутизації в мережах. Матеріали сьомої міжнародної науково-практичної конференції з програмування УкрПРОГ'2010. Проблеми програмування, 2010, №2-3, с. 171-177.

9. Anaconda Accelerate [Електронний ресурс]. – Режим доступу: https://developer.nvidia.com/anaconda-accelerate.

10. Continuum analytics [Електронний ресурс]. – Режим доступу: https://www.continuum.io.

11. GPU Accelerated Computing with Python [Електронний ресурс]. – Режим доступу: https://developer.nvidia.com/how-to-cuda-python.

12. Kaufmann Morgan. CUDA Application Design and Development, Rob Farber / Morgan Kauf-mann. – Waltham, Massachusetts (USA). – November 14, 2011. – 336 p.

13. LINPACK [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/LINPACK.

14. Pogorilyy S.D. Y, Gusarov A.D. Paralleling Of Edmonds-Karp Net Flow Algorithm. Appl. Comput. Math. 5 (2006), no.2, pp.121-130.

15. Sanders Jason. CUDA by Example: An Introduction to General-Purpose GPU Programming / Jason Sanders, Edward Kandrot. – Addison-Wesley Professional. Ann Arbor, Michigan (USA). – July 2010. – 312 p.

Надійшла до редакції 20.09.2016

С.Д. ПОГОРЕЛЫЙ, Б.А. СЕМЁНОВ
Киевский национальный университет им. Тараса Шевченко

ИССЛЕДОВАНИЕ ПРОГРАММНОЙ БИБЛИОТЕКИ LINPACK НА АРХИТЕКТУРЕ CUDA НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

Реализована последовательная версия метода Гаусса (решение систем линейных алгебраических уравнений) и ее параллельная версия на ядрах архитектуры CUDA на языке программирования Python. Установлен пакет Linpack от разработчиков компании NVidia и проведен ряд исследований: сравнение скорости выполнения различных реализаций метода Гаусса и пакета Linpack.

Ключевые слова: GPGPU, CUDA, SIMD, SIMT, Python, PyCUDA, Anaconda, Linpack, метод Гаусса.

S.D. POGORILYY, B.A. SEMONOV
Taras Shevchenko National University of Kyiv

RESEARCH OF LINPACK PROGRAMMING LIBRARY ON CUDA ARCHITECTURE WITH PYTHON PROGRAMMING LANGUAGE

Python programming language was the first main object of the research of the article. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. Some tables were created for comparison of Python with other languages.

General-purpose computing on graphics processing units (GPGPU) was considered in the paper, which typically handles computation only for computer graphics and is used to perform computation in applications traditionally handled by the central processing unit (CPU). The article examines the implementation of GPGPU technology tools with programming language Python APIs: PyCUDA and Anaconda.

NVidia CUDA technology was introduced, which was the second main object of the research, and was given its advantages in the article. Possibilities of this technology were shown. Fast facts about NVIDIA were given. Then it was considered its production and achievements. It could be concluded that this technology made a great impact on the market and is developing very actively.

Then an algorithm of solution was given, based on matrix factorization, called HPL (High-performance Linpack). It was shown advantages of this package in linear system solution and compared with other packages. Short manual for this package was described. An example of linear system solution by this package was also given. Parallel version of this program was shown that can accelerate solution. Such technology can be very useful in modeling different situations where linear system is used.

Serial version of the Gaussian elimination (solving systems of linear algebraic equations) and its parallel version on the CUDA architecture cores in the Python programming language were implemented. Linpack package from NVidia company developers was installed and a number of researches was done: a comparison of the speed of the various implementations of the Gaussian elimination and Linpack package.

Keywords: GPGPU, CUDA, SIMD, SIMT, Python, PyCUDA, Anaconda, Linpack, Gaussian elimination.