УДК 004.3,004.421.43

Погорілий С. Д., Семьонов Б. О.

ДОСЛІДЖЕННЯ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ МОВОЮ PYTHON З ВИКОРИСТАННЯМ РІЗНИХ ПЛАТФОРМ

Реалізовано послідовну версію методу Гауса (розв'язання систем лінійних алгебраїчних рівнянь) та її паралельну версію на ядрах архітектури CUDA з використанням різних програмних бібліотек та платформ на мові програмування Python, а саме: Anaconda (Numba), PyCUDA, KappaCUDA та PyOpenCL. Проведено низку досліджень: порівняння швидкості виконання різних реалізацій методу Гауса.

Ключові слова: GPGPU, CUDA, SIMD, SIMT, Python, PyCUDA, Anaconda, PyOpenCL, KappaCUDA, 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 напряму, використовують пам'ять відеоадаптера для розміщення структур даних, як-от текстури, буфери, визначають конвеєр обробки, кожен етап якого відповідає за свої специфічні дії: растеризацію, інтерполяцію, блендинг, теселяцію [4, с. 63] 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) [5, с. 76].

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 у 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 таких 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 [11, с. 214]. Сьогодні архітектура CUDA набула значної популярності в комп'ютерних системах при обчисленні великих обсягів даних. Тому є доцільним розширення переліку мов програмування, здатних підтримувати цю програмну модель.

Нині таку можливість має мова програмування Python, яка набула широкого застосування в 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. 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 дають змогу 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, усі 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 [16, с. 57].

Таблиця 1. **Умовні позначення**

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

Таблиці 2–5 ілюструють переваги Python у порівнянні з іншими інтерпретованими мовами програмування. Введемо такі позначення (табл. 1).

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 (Numba, PyCUDA, KappaCUDA та PyOpenCL). Предметом дослідження є «паралельний» на ядрах CUDA метод розв'язування систем лінійних алгебраїчних рівнянь (метод Гауса).

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 [1, с. 137] – 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 завдяки реалізації деяких системних викликів (у 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) [2, с. 301].

Важливою перевагою CUDA є використання для програмування GPU мов високого рівня. На сьогодні 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 [8, с. 51].

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 галузях [3, с. 64]. 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 [6, с. 120].

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.debugdemo.py* [2, с. 232].

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

PyCUDA використовує ту саму програмну модель, що й 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 [13, с. 127].

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 спростити розробку, не погіршивши при цьому продуктивність ядер. За рахунок додаткової

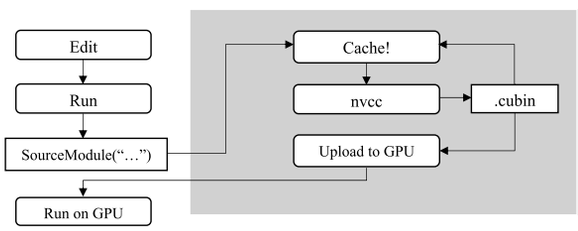

Рис. 1. Процес розробки на PyCUDA

інтеграції API CUDA всі помилки, які виникають у процесі роботи застосування, перетворюються у виключення Python.

Anaconda (Numba). 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, с. 1].

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 дає змогу 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 понад сто 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 цієї платформи на комерційній основі [9, с. 1].

CUDA JIT компілятор є низькорівневою точкою входу в множину функцій NumbaPro. Він переводить функції Python у 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 [12, с. 1].

Більшість різних відкритих функцій та методів CUDA API містяться 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 всередині CUDA-ядра використовується метод cuda.syncthreads(). Його принцип дії: наступна інструкція виконається тільки тоді, коли всі нитки «прийдуть» у дану точку, де викликаний цей метод.

KappaCUDA. Kappa є передовим програмним забезпеченням, яке дає змогу легко отримати максимальну продуктивність при обробці даних за допомогою відносно недорогого високопаралельного CPU і GPU обладнання. Kappa працює із сучасними мовами програмування і джерелами даних. Платформа Kappa готова підвищити TCO (Total Cost of Ownership, сукупну вартість володіння) за рахунок розподілення робочого навантаження кластерів сотень серверів усього лише на кілька серверів.

Kappa-фреймворк підтримує інтерпретовану мову програмування Python. Вихідний код для пакету KappaCUDAPython доступний під ліцензією MIT, тому зацікавлені сторони можуть взяти на себе обслуговування і підтримку [14, с. 1].

Модуль KappaCUDA дає змогу отримати доступ до архітектури NVidia CUDA через бібліотеку Kappa, а ключові слова мови програмування Python дозволяють запускати Python-підпрограми на виконання потоків, які мають контекст CUDA GPU (з доступом до контексту CUDA, змінні і т. п.).

PyOpenCL. OpenCL (від англ. Open Computing Language) – фреймворк для створення комп'ютерних програм, пов'язаних із паралельними обчисленнями на різних графічних GPU і центральних процесорах CPU. У фреймворк OpenCL входять мова програмування, яка базується на стандарті C99, та інтерфейс програмування комп'ютерних програм. OpenCL забезпечує паралельність на рівні інструкцій та на рівні даних і є реалізацією техніки GPGPU. OpenCL – повністю відкритий стандарт, його використання доступне на базі вільних ліцензій.

Мета OpenCL полягає в тому, щоб доповнити OpenGL і OpenAL, які є відкритими галузевими стандартами для тривимірної комп'ютерної графіки і звуку, користуючись можливостями GPU. OpenCL розробляв і підтримує некомерційний консорціум Khronos Group, у який входять

багато великих компаній, включаючи Apple, AMD, ARM, Intel, NVidia, Qualcomm, Sun Microsystems, Sony Computer Entertainment та ін.

Бібліотека PyOpenCL надає можливість використання фреймворку OpenCL з мовою програмування Python.

Деякі особливості PyOpenCL:

• знищення об'єктів прив'язане до життєвого циклу об'єктів. Ця ідіома, яку часто називають RAII в C++, робить легшим написання коду (менше ризику написати програму, що «тече»);

• повнота. PyOpenCL пропонує всю міць API OpenCL;

• автоматична перевірка помилок. Усі помилки автоматично перетворюються у виключення (exceptions) Python;

• швидкість. Базовий шар PyOpenCL написаний на C ++;

• відкритість. Бібліотека PyOpenCL має відкритий вихідний код та розповсюджується під ліцензією MIT і є безкоштовною для комерційного, академічного і приватного використання [15, с. 1].

Порівняльна характеристика застосованих бібліотек

У таблиці 6 наведено порівняння бібліотек, які були застосовані для реалізації розпаралеленого (архітектура NVidia CUDA) алгоритму розв'язання СЛАР (метод Гауса) мовою програмування Python.

Таблиця 6. **Порівняльна характеристика бібліотек для Python CUDA**

Характеристики	**Anaconda** (тест № 2)	**PyCUDA** (тест № 3)	**Kappa** (тест № 4)	**PyOpenCL** (тест № 5)
Ліцензія	BSD	MIT	MIT	MIT
Мова CUDA-ядра	Python	CUDA C	CUDA C	C
Автоматичне очищення пам'яті	Так	Так	Так	Так
Вигляд CUDA-ниток	tx = numba.threadIdx.x	inttx = threadIdx.x	inttx = threadIdx.x	inttx = get_global_id(0)
CUDA-ядро в одному файлі	Так	Так	Ні	Так
Мова Python є основною мовою	Так	Ні	Ні	Ні
Підтримка чисел із плаваючою комою (тип float)	Так	Так	Так	Так
Підтримка паралелізації на CPU	Так	Ні	Так	Ні
Наявність власного SDK	Так (Spyder)	Ні	Ні	Ні
Jit-компіляція Python-коду	Так	Ні	Ні	Ні

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, с. 267].

Перевагами методу є такі: для матриць обмеженого розміру менш трудомісткий у порівнянні з іншими методами; дає змогу однозначно встановити, сумісна система чи ні, і якщо сумісна, знайти її розв'язок; дає змогу 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.

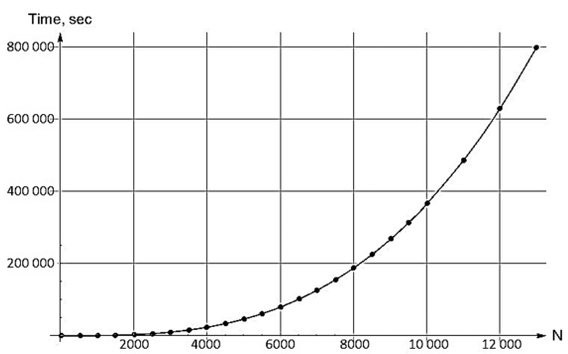

Рис. 2. Графік результатів тесту № 1

Тестування проводилося на комп'ютерній системі з такими характеристиками:

- LinuxUbuntu 16.04;
- AMD-FX 3.5 GHz;
- RAM 8GB.

Тести № 2, 3, 4, 5. 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

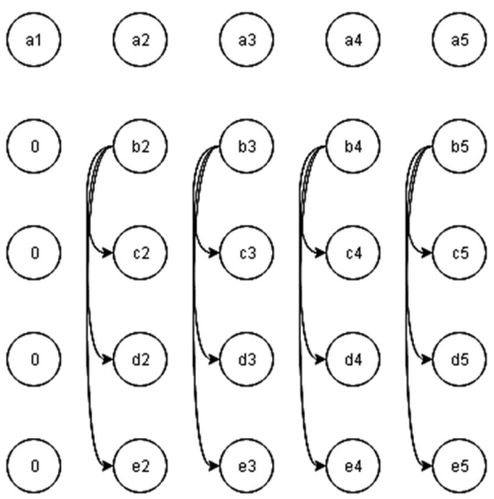

Рис. 3. Підхід до розпаралелювання алгоритму Гауса

покладено такий 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 – NVidiaGeForce GTX 550 Ti;
- архітектура Fermi;
- кількість ядер CUDA – 192;
- тактова частота графічної підсистеми – 970 МГц;
- швидкість передачі даних пам'яті – 4104 МГц;
- відеопам'ять – 1024 МБ DDR3.

Тест № 2. Паралельний алгоритм Гауса з використанням бібліотеки Numba (платформа Anaconda).

У **Тесті № 3** досліджується інтерфейс PyCUDA.

Тест № 4 – тестується бібліотека Kappa.

У наступному тесті (**Тест № 5)** розглядається фреймворк PyOpenCL.

На рис. 4 наведено графіки залежності часу виконання наведених вище тестів від розмірності матриці. Варто зауважити: N = 2000 означає, що в пам'ять графічного пристрою (GPU) було завантажено 2000^2 елементів. Як видно

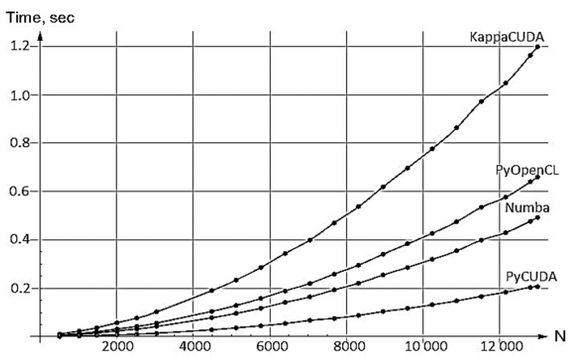

Рис. 4. Графік залежності часу роботи паралельних версій від розмірності матриці

з графіка, найшвидшим є інтерфейс PyCUDA, а найповільнішим – KappaCUDA.

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.

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).

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, Anaconda, KappaCUDA, PyOpenCL. Результати тестування наведених вище бібліотек показали майже однакові результати: матриці будь-якої (в межах пам'яті GPU) розмірності

обробляються менше двох секунд, що є хорошим результатом, тоді як на одному ядрі центрального процесора, починаючи з розмірності 2000^2 елементів, метод Гауса може працювати годинами. Максимальний розмір матриці, яку вдалося завантажити до графічного пристрою, становить приблизно 13000^2 елементів.

Проведений аналіз швидкодії виконання «паралельного» методу Гауса на ядрах CUDA мовою програмування Python показав доцільність її використання на архітектурі CUDA. Найкращим за швидкодією виявився інтерфейс PyCUDA. Але написання CUDA-ядра з використанням бібліотеки PyCUDA потребує від розробника знання мови CUDA C, тоді як менш швидка бібліотека Numba не вимагає додаткових знань, тому що CUDA-ядро використовує мову Python; проте це дуже сумнівний недолік інтерфейсу PyCUDA. Більш вагомим недоліком інтерфейсу PyCUDA є те, що CUDA-ядро передається в Python-функцію як багаторядковий коментар мови Python. Це викликає незручності при написанні CUDA-ядра, тому що в середовищах розробки для Python будуть відсутні попередження про допущені розробником помилки у синтаксисі CUDA-ядра, і про помилки можна буде дізнатися лише під час збирання Python-застосування. При використанні бібліотеки Numba така проблема не виникне.

Список літератури

1. Боресков А. В. Основы работы с технологией CUDA / А. В. Боресков, А. А. Харламов. – Москва : ДМК-Пресс, 2010. – 232 с.
2. Параллельные вычисления на GPU. Архитектура и программная модель CUDA : учеб. пособ. / А. В. Боресков и др. ; предисл. В. А. Садовничий. – Москва : Изд-во Московского ун-та, 2012. – 336 с. – (Серия «Суперкомпьютерное образование»).
3. Погорілий С. Д. Генетичний алгоритм розв'язання задачі маршрутизації в мережах / С. Д. Погорілий, Р. В. Білоус // Проблеми програмування. – 2010. – № 2–3 : Матеріали сьомої міжнародної науково-практичної конференції з програмування «УкрПРОГ'2010». – С. 171–177.
4. Погорілий С. Д. Новітні архітектури відеоадаптерів. Технологія GPGPU. Частина 1 / С. Д. Погорілий, Д. Ю. Вітель, О. А. Верещинський // Реєстрація, зберігання і обробка даних. – 2012. – Т. 14, № 4. – С. 52–64.
5. Погорілий С. Д. Новітні архітектури відеоадаптерів. Технологія GPGPU. Частина 2 / С. Д. Погорілий, Д. Ю. Вітель, О. А. Верещинський // Реєстрація, зберігання і обробка даних. – 2013. – Т. 15, № 1. – С. 71–81.
6. Погорілий С. Д. Програмне конструювання : підручник / С. Д. Погорілий ; за ред. О. В. Третяка. – 2-ге вид. – Київ : ВПЦ «Київський університет», 2007. – 438 с. – (Автоматизація наукових досліджень).
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. – Київ : ВПЦ «Київський університет», 2013. – 463 с.
8. Програмування числових методів мовою PYTHON : підручник / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – Київ : ВПЦ «Київський університет», 2015. – 640 с.
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. Farber R. CUDA Application Design and Development / Rob Farber. – Waltham, MA : Morgan Kaufmann, 2011. – 336 p.
12. GPU Accelerated Computing with Python [Електронний ресурс]. – Режим доступу: https://developer.nvidia.com/how-to-cuda-python.
13. Pogorilyy S. D. Paralleling of Edmonds-Karp Net Flow Algorithm / S. D. Pogorilyy, A. D. Gusarov // Appl. Comput. Math. – 2006. – Vol. 5, no. 2. – P. 121–130.
14. Psi Lambda LLC | Parallel computing made practical [Електронний ресурс]. – Режим доступу: http://psilambda.com/download/kappa-for-python.
15. PyOpenCL [Електронний ресурс]. – Режим доступу: https://documen.tician.de/pyopencl.
16. Sanders J. CUDA by Example: An Introduction to General-Purpose GPU Programming / Jason Sanders, Edward Kandrot. – Michigan : Addison-Wesley Professional, 2010. – 312 p.

S. Pogorilyy, B. Semonov

RESEARCH OF PARALLEL ALGORITHMS WITH PYTHON LANGUAGE WITH USING VARIOUS PLATFORMS

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

General-purpose computing on graphics processing units (GPGPU) is 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: Anaconda, PyCUDA, KappaCUDA, and PyOpenCL.

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

The article deals with modern architectures used in video adapters. The code is given as an example, which is used in working with graphical process units, and the instructions how to work with the global memory are offered. It could be concluded that such a technology can be used in a wide variety of applications and can be developed in different ways. Different patterns of thread interaction in this technology are shown. The problems of synchronization and different solutions are considered. The dynamic parallelism and its principles are defined.

A 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. A number of research procedures were done: a comparison of the speed of the various implementations of the Gaussian elimination with using of various program libraries and platforms: Anaconda (Numba), PyCUDA, KappaCUDA, and PyOpenCL.

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

Матеріал надійшов 13.10.2017

УДК 004.421.2:519.17

Глибовець М. М., Петльована М. В., Кирієнко О. В.

ЗАСТОСУВАННЯ ЕВОЛЮЦІЙНИХ АЛГОРИТМІВ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ АПРОКСИМАЦІЇ ЗОБРАЖЕНЬ МНОГОКУТНИКАМИ

У статті описано розробку та реалізацію еволюційного алгоритму розв'язку задачі апроксимації зображення многокутниками, запропоновано структуру даних для ефективного кодування зображень (зі змінною кількістю рядків та стовпців зображення, кількістю многокутників в одній клітинці, кількістю точок многокутників та ступенем їх перетинання). Було впроваджено стратегії мутації для виведення розв'язку з локального оптимуму та подальшого знаходження глобального оптимуму. Для пришвидшення роботи алгоритму етапи селекції та перевірки критерію завершення було реалізовано з використанням моделі розподілених обчислень MapReduce.

Ключові слова: еволюційні алгоритми, задачі апроксимації зображення многокутниками, MapReduce.

Вступ

Представлення зображень у вигляді множини многокутників є важливим етапом для багатьох задач аналізу зображень, як-от розпізнавання об'єкта, зіставлення зображень, супровід цілі і т. п. Актуальність та важливість цих задач у сучасній науці та техніці створює необхідність розробки ефективних алгоритмів апроксимації зображень многокутниками. Еволюційні алгоритми є перспективним напрямом вирішення поставленої проблеми.

У задачі генерації зображень за допомогою еволюційних алгоритмів можна виділити два основні напрями: автоматизована генерація зображень та оптимізація і покращення якості готових зображень. Генетичні технології ефективно допомагають у реконструкції і стилізації зображень, а саме в покращенні освітлення, контрастності, чіткості контурів тощо.

Проблемою апроксимації зображень та графічних об'єктів за допомогою многокутників займалося багато вчених світу. Зокрема, у роботах [4; 5] розглянуто ефективні алгоритми для наближення