УДК 004.7

О.О. Бєрнікова
В.О. Денисюк

РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ СОРТУВАННЯ ЗА РОЗРЯДАМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ OPENCL

Вінницький національний технічний університет

Анотація

Розглянуто розробку паралельного алгоритму сортування за розрядами. Проаналізовано основні підходи до побудови паралельних обчислень, архітектуру OpenCL та можливості бібліотеки PyOpenCL. Виконано математичне моделювання алгоритму, побудовано потоковий граф алгоритму, розроблено UML-діаграми структури програмного модуля, створено програмну реалізацію та проведено тестування ефективності паралельного сортування. Створений програмний модуль дозволяє суттєво пришвидшити сортування великих масивів даних завдяки обчислювальній потужності графічного процесора. Це рішення доцільно використовувати для оптимізації систем, що потребують швидкої обробки значних обсягів інформації.

Ключові слова: *паралельний алгоритм, сортування за розрядами, OpenCL, PyOpenCL, GPGPU.*

Abstract

The development of a parallel radix sort algorithm is considered. The main approaches to constructing parallel computations, the OpenCL architecture, and the capabilities of the PyOpenCL library are analyzed. Mathematical modeling of the algorithm is performed, the algorithm's data flow graph is constructed, UML diagrams of the software module structure are developed, a software implementation is created, and the efficiency of parallel sorting is tested. The created software module allows for significantly accelerating the sorting of large data arrays due to the computational power of the graphics processor. This solution is advisable to use for optimizing systems that require rapid processing of significant volumes of information.

Keywords: *parallel algorithm, radix sort, OpenCL, PyOpenCL, GPGPU.*

Вступ

Актуальність реалізації сортування за розрядами за допомогою технології OpenCL полягає у тому, що сортування є базовим етапом для роботи пошукових систем, баз даних та наукових обчислень. Для ефективної обробки великих обсягів інформації доцільно використовувати відкритий стандарт OpenCL, який дозволяє організувати паралельні обчислення на гетерогенних платформах, забезпечуючи уніфікований доступ до ресурсів як центральних, так і графічних процесорів. Зокрема, перенесення обчислень на графічний процесор є ефективним завдяки його архітектурі, яка містить тисячі обчислювальних ядер, здатних виконувати однакові операції над великими масивами даних одночасно. Це надає можливість застосовувати архітектуру SIMD, що суттєво підвищує швидкодію алгоритмів порівняно з послідовним виконанням.

Метою роботи є дослідження та реалізація паралельного алгоритму сортування за розрядами з використанням мови програмування Python та бібліотеки PyOpenCL для покращення швидкодії обробки даних.

Постановка задач дослідження

Відповідно до мети роботи визначено такі основні задачі дослідження: дослідження предметної області паралельних обчислень та архітектури OpenCL; математичне моделювання алгоритму та побудова потокового графу; розробка оптимізованої програми для сортування за розрядами на GPU; тестування програми та аналіз залежності часу виконання від обсягу даних.

Виклад основного матеріалу

Тривалий час підвищення продуктивності обчислювальних систем забезпечувалося збільшенням тактової частоти процесорів, проте цей підхід вичерпав себе через надмірне

енергоспоживання та перегрів. Це стало передумовою переходу до багатоядерних архітектур та розвитку методів паралельної обробки даних.

Паралельні обчислення – це спосіб організації виконання програми, за якого завдання поділяється на окремі підзадачі, що можуть виконуватися одночасно на різних обчислювальних елементах. Для опису структури таких систем зазвичай застосовують класифікацію Флінна. Для реалізації алгоритмів на графічних процесорах важливою є архітектура SIMD, оскільки вона дозволяє виконувати одну й ту ж операцію над величезним масивом даних одночасно [1]. Основним показником ефективності є прискорення, проте на практиці воно обмежується законом Амдала, який враховує ту частину програми, яку не можна розпаралелити [2].

Для програмної реалізації паралельного алгоритму обрано відкритий стандарт OpenCL, який дозволяє організувати ефективні обчислення на гетерогенних системах. Архітектура OpenCL базується на чотирьох моделях: платформи, виконання, пам’яті та програмування, що забезпечує уніфікований підхід до керування ресурсами [3, 4].

Для реалізації обрано бібліотеку PyOpenCL, яка інтегрує можливості OpenCL у середовище Python. Це дозволяє уникнути громіздкості коду, характерної для реалізації керуючої частини мовами C/C++, завдяки автоматичному управлінню ресурсами та пам'яттю. Такий підхід поєднує зручність високорівневого програмування (легка компіляція ядер, зрозумілий синтаксис) із повним доступом до апаратних можливостей графічного процесора [5].

У роботі реалізовано алгоритм сортування за розрядами (Radix Sort), принцип якого ґрунтується на обробці чисел по розрядах. Для програмної реалізації обрано метод LSD (Least Significant Digit), суть якого полягає у тому, що масив чисел сортується поетапно: від молодшого розряду до старшого. Важливою умовою є стабільність сортування. Ця властивість гарантує, що якщо два числа мають однакове значення у поточному розряді, їхній відносний порядок, встановлений на попередніх етапах, залишається незмінним [6, 7].

Математична модель паралельного алгоритму базується на векторному представленні чисел у двійковій системі. Вхідний масив складається з n елементів, де кожен елемент a_i розглядається як набір бітів. Головною проблемою розпаралелювання є визначення унікального місця для кожного числа у новому масиві без конфліктів доступу до пам'яті. Для цього застосовано метод обчислення індексів у три етапи

1) Побудова вектора предикатів. Для поточного розряду формується допоміжний масив, який містить одиниці для чисел з бітом «0» і нулі для чисел з бітом «1». Це дозволяє виділити групу елементів, які мають переміститися у початок масиву.
2) Обчислення префіксних сум. Над вектором предикатів виконується операція сканування (scan). Результат цієї операції для кожного елемента показує кількість нулів, що знаходяться перед ним. Це значення фактично є майбутнім індексом для елементів першої групи.
3) Розрахунок глобальних адрес. Фінальна позиція кожного елемента (indexi) визначається за формулою:
   - для чисел з бітом «0»: позиція дорівнює значенню префіксної суми f_i;
   - для чисел з бітом «1»: позиція обчислюється як сума загальної кількості нулів (TotalFalses) та порядкового номера елемента серед одиниць $(i - f_i)$.

Така модель дозволяє кожному обчислювальному потоку GPU незалежно розрахувати свою адресу запису, забезпечуючи коректне розміщення даних [10].

Оцінка часової складності алгоритму для сортування n чисел з розрядністю b у загальному вигляді описується формулою $T(n) = \Theta(\frac{b}{r}(n + 2^r))$, де r – кількість бітів, що обробляються за одну ітерацію. Оскільки для програмної реалізації на GPU обрано метод побітового сортування (r=1), а розрядність b є фіксованою константою, складність трансформується у вигляд $T(n) = \Theta$. Це свідчить про лінійну залежність часу виконання від кількості вхідних даних, що є значною перевагою над алгоритмами, які базуються на порівнянні елементів [7].

Взаємодію компонентів алгоритму та рух даних наочно представлено на потоковому графі. Керування всім процесом здійснює макрооператор управління $\Phi_{МУ}$ (хост-програма на CPU), який відповідає за запуск обчислень. Центральне місце займає макрооператор пам'яті $\Phi_{МПЗ}$ (глобальна пам'ять відеокарти), який виступає сховищем для вхідних даних, результатів та проміжних значень. Безпосередні обчислення виконують три оператори, що відповідають ядрам OpenCL:

– Φ_{Pred} (Predicate) – визначає, чи є біт нульовим;
– Φ_{Scan} (Scan) – розраховує позиції для переміщення елементів;
– Φ_{Scat} (Scatter) – виконує фінальну перестановку чисел у пам'яті.

Для оптимізації алгоритму сортування за розрядами на GPU покращено такі аспекти.

1) Мінімізація передачі даних. Проміжні масиви зберігаються безпосередньо в пам’яті GPU (використання буферів), що зменшує кількість пересилань даних на CPU [3].

2) Швидкий доступ до глобальної пам’яті. Організовано дані так, щоб сусідні потоки обробляли сусідні елементи масиву (коалесценція пам’яті) [8].

3) Ефективне використання локальної пам’яті. Застосовано техніку відступів (padding) для уникнення конфліктів банків пам'яті, щоб кілька потоків не звертались до одного блоку одночасно [3, 8].

4) Оптимізація циклів. Використано розгортання циклів, при якому параметри передаються як константи під час компіляції [3].

5) Оптимізація умовних команд. Мінімізовано розгалуження потоків шляхом використання ефективних примітивів для сканування [3].

Програмно реалізовано паралельне сортування за розрядами та описано всі компоненти коду. Для забезпечення коректності порівняльного аналізу додатково розроблено функцію послідовного сортування, яка виконується на центральному процесорі та реалізує ідентичну логіку. Ефективність розробленого алгоритму перевірено шляхом порівняльного тестування обох версій (CPU та GPU) на масивах різної розмірності. Результати наведено у таблиці 1.

Таблиця 1 – Часові характеристики роботи алгоритму сортування за розрядами (с)

Розмір масиву (N)	CPU (с)	GPU (с)	Прискорення
10 000	0.0040	0.0165	0.25x
50 000	0.0165	0.0172	0.96x
100 000	0.0322	0.0157	2.05x
1 000 000	0.4282	0.0507	8.44x
10 000 000	4.4832	0.1775	25.25x
50 000 000	22.5551	0.7623	29.59x
100 000 000	45.5667	1.3189	34.55x
150 000 000	68.0333	1.9985	34.04x
200 000 000	95.0978	2.4788	38.36x
210 000 000	108.0587	3.9190	27.57x

Аналіз результатів показав, що на малих обсягах даних (до 50 тисяч) ефективнішим залишається центральний процесор через накладні витрати на ініціалізацію контексту та передачу даних. Проте зі зростанням обсягу даних ефективність паралельного алгоритму стрімко зростає, досягаючи максимального прискорення у 38,36 разів на масиві з 200 мільйонів елементів. При досягненні граничних обсягів (210 мільйонів) продуктивність знижується через вичерпання фізичної пам'яті відеокарти та перехід на використання оперативної пам'яті.

Висновки

Проведено аналіз предметної області паралельних обчислень, архітектури стандарту OpenCL та можливостей бібліотеки PyOpenCL. Надалі виконано математичне моделювання алгоритму сортування за розрядами та побудовано потоковий граф, що відображає структурну організацію обчислювального процесу на GPU. На етапі проєктування розроблено UML-діаграми та проведено оптимізацію алгоритму, яка полягала у мінімізації обміну даними та налаштуванні ефективного

доступу до пам’яті, після чого створено програмну реалізацію паралельного алгоритму мовою Python та його послідовну версію для центрального процесора. Тестування проводилось на масивах від 10 тисяч до 210 мільйонів елементів і показало, що за невеликих обсягів даних ефективнішим залишається центральний процесор, тоді як для великих масивів графічний процесор забезпечує значне прискорення. Встановлено, що при обробці надвеликих масивів даних швидкість роботи обмежується обсягом відеопам’яті.

Таким чином, результати тестування засвідчили, що використання технології OpenCL на графічному процесорі забезпечує високу продуктивність та суттєве прискорення обчислень під час опрацювання великих обсягів даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Trobec R., Slivnik B., Bulić P., Robič B. Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art Platforms. Springer, 2018.
2. Семеренко В. П. Технології паралельних обчислень: навчальний посібник. Вінниця: ВНТУ, 2018. 104 с.
3. NVIDIA Corporation. OpenCL Programming Guide for the CUDA Architecture. Version 4.0. NVIDIA Corporation, 2011.
4. Munshi A., Gaster B., Mattson T. G. OpenCL Programming Guide. Addison-Wesley Professional, 2011. 648 p.
5. Klöckner A. PyOpenCL Documentation. Version 2024.1. 2024. URL: https://documen.tician.de/pyopencl/
6. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 3rd ed. MIT Press, 2009. 1312 p.
7. Sedgewick R., Wayne K. Algorithms. 4th ed. Addison-Wesley Professional, 2011. 992 p.
8. Kirk D. B., Hwu W. M. W. Programming Massively Parallel Processors: A Hands-on Approach. 3rd ed. Morgan Kaufmann, 2016. 576 p.

Бєрнікова Олена Олексіївна – студентка кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м. Вінниця, e-mail: olebersever@gmail.com;

Денисюк Валерій Олександрович – канд. техн. наук, доцент, доцент кафедри комп’ютерних наук, Вінницький національний технічний університет, м. Вінниця, e-mail: vad64@i.ua.

Bernikova Olena Oleksiivna – student of the Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: olebersever@gmail.com;

Denysiuk Valerii Oleksandrovych – Ph.D., Associate Professor, Associate Professor of the Computer Science Department, Vinnytsia National Technical University, Vinnytsia, e-mail: vad64@i.ua.