УДК 004.8

А. Р. Пащенко
В.О. Денисюк

РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ ШВИДКОГО СОРТУВАННЯ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ C++ AMP

Вінницький національний технічний університет

Анотація

Розглянуто розробку та реалізацію гібридного паралельного алгоритму швидкого сортування (Quick Sort), призначений для ефективного виконання на гетерогенних обчислювальних системах із залученням потужностей графічних процесорів (GPU) за допомогою технології C++ AMP. Проаналізовано основні алгоритми сортування, принципи паралельних алгоритмів, розроблено та реалізовано паралельний алгоритм Quick Sort, а також проведено тестування на коректність та продуктивність. Отримані результати свідчать про доцільність використання технології C++ AMP для оптимізації процесу сортування.

Ключові слова: *паралельний алгоритм, швидке сортування, Quick Sort, C++ AMP, оптимізація.*

Abstract

The development and implementation of a hybrid parallel algorithm for quick sorting (Quick Sort), designed for efficient execution on heterogeneous computing systems with the use of graphics processing units (GPU) using C++ AMP technology, is considered. The main sorting algorithms and the principles of parallel algorithms are analyzed, the parallel algorithm Quick Sort is developed and implemented, and testing for correctness and performance is carried out. The results obtained indicate the feasibility of using C++ AMP technology to optimize the sorting process.

Keywords: *parallel algorithm, quick sorting, Quick Sort, C++ AMP, optimization.*

Вступ

Актуальність реалізації паралельного алгоритму швидкого сортування полягає у потребі скорочення часу виконання операцій упорядкування на великих наборах даних. Застосування технології C++ AMP відкриває можливості для істотного прискорення процесу за рахунок масивного паралелізму, що притаманний сучасним графічним процесорам. Дослідження ефективності паралельної реалізації QuickSort дозволяє оцінити потенціал GPU у задачах сортування та визначити оптимальні методи організації паралельних обчислень.Алгоритм швидкого сортування (Quick Sort) є одним з найбільш ефективних алгоритмів сортування завдяки середній часовій складності O(nlogn) та використанню принципу «розділяй і володарюй». Його структура дозволяє виконувати операції поділу масиву та обробки підмасивів незалежно, що робить алгоритм придатним для паралельної реалізації. Використання графічних процесорів (GPU) у поєднанні з центральним процесором (CPU) дає змогу значно прискорити найбільш ресурсоємні етапи алгоритму, зокрема масові операції порівняння елементів.

Метою даної роботи є дослідження та реалізація паралельного алгоритму швидкого сортування з використанням технології C++ AMP для підвищення продуктивності процесу упорядкування даних.

Постановка задачі дослідження

Задачі дослідження полягають у вирішенні наступних питань:

- розробка паралельного алгоритму швидкого сортування з урахуванням особливостей гетерогенної архітектури обчислювальних систем, що поєднують центральний процесор та графічний процесор;
- мінімізація накладних витрат, пов’язаних із передаванням даних між CPU та GPU, шляхом ефективної організації доступу до пам’яті та синхронізації обчислень;
- створення програмної реалізації гібридного паралельного алгоритму швидкого сортування мовою C++ з використанням бібліотеки C++ AMP;
- тестування програми та аналіз отриманих результатів.

Виклад основного матеріалу

Сортування є однією з фундаментальних операцій у комп'ютерних науках, що полягає у впорядкуванні елементів за визначеним критерієм — найчастіше за зростанням або спаданням ключового значення. Від ефективності виконання сортування залежить продуктивність численних алгоритмів, структур даних та інформаційних систем. Сортування використовується в організації великих обсягів даних, у базах даних, пошукових системах, методах оптимізації, аналітичних задачах та у підготовці даних для подальших обчислень [1-3].

Особливе місце серед алгоритмів займає швидке сортування (Quick Sort) — один із найефективніших та найпоширеніших методів сортування для практичних застосувань. Робота алгоритму базується на рекурсивному розділенні масиву на дві частини відносно опорного елемента (pivot): елементи, менші за pivot, формують одну підмножину, елементи, більші за pivot, — іншу. Після розбиття алгоритм застосовується до кожної підмножини окремо [2].

Quick Sort має низку суттєвих переваг: висока середня швидкодія, локальність доступу до пам'яті, низькі накладні витрати та природна придатність до паралельної обробки. Середня складність становить O(nlogn), що робить алгоритм одним із найшвидших у реальних умовах. Недоліком є можливість деградації до O(n2) при невдалому виборі опорного елемента, що вирішується використанням стратегій вибору pivot, таких як «медіана трьох» та випадковий вибір. Також після операції розбиття обидві частини масиву є незалежними, що дозволяє ефективно розпаралелювати обчислення, зокрема на графічних процесорах за допомогою таких технологій, як C++ AMP [4].

Технологія C++ AMP (Accelerated Massive Parallelism) є сучасним інструментом для організації паралельних обчислень на гетерогенних системах, що включають центральний процесор (CPU) і графічний процесор (GPU). Використання C++ AMP у задачах сортування є перспективним, оскільки алгоритми обробки масивів добре поєднуються з паралельною природою GPU. Зокрема, швидке сортування отримує додаткові переваги через незалежність підмасивів після етапу розбиття, що дозволяє реалізувати ефективну паралельну обробку. Це робить C++ AMP доцільним вибором для реалізації високопродуктивного варіанту Quick Sort у сучасних системах масових паралельних обчислень [4-6].

Оптимізація алгоритму швидкого сортування має ключове значення для підвищення його ефективності, особливо при обробці великих масивів даних. Розглянемо підходи до оптимізації, що дозволяють покращити швидкодію, мінімізувати використання пам'яті та уникнути найгірших сценаріїв.

1. Балансування навантаження через оптимальний вибір півота: для алгоритмів, що базуються на стратегії «Розділяй і володарюй» (наприклад, Quick Sort), критично важливим є статистично надійний вибір опорного елемента (півота) (наприклад, "медіана трьох" або випадковий вибір). Це запобігає дисбалансу навантаження, нівелює простої обчислювальних ресурсів та уникнення деградації часової складності до $O(n^2)$.
2. Гібридні алгоритмічні стратегії та динамічний розподіл роботи: необхідно застосовувати гібридні підходи, доповнюючи складні паралельні алгоритми (як Quick Sort) швидшими послідовними (як Insertion Sort) для обробки малих підмасивів (менше 16–32 елементів). Це знижує накладні витрати CPU на керування рекурсією. Додатково використовується динамічний розподіл роботи для завдань із непередбачуваною тривалістю.
3. Мінімізація трансферу даних CPU–GPU: у гетерогенних системах (CPU/GPU) продуктивність критично залежить від максимального утримання даних у пам'яті GPU (VRAM). Операції копіювання (copyToGPU, copyFromGPU) є блокуючими та становлять головне "вузьке місце", тому їхня кількість має бути зведена до мінімуму.
4. Оптимізація кеш-пам'яті CPU та уникнення псевдоспільного доступу (false sharing): для ефективної обробки на CPU забезпечується локальність даних (дані, що обробляються разом, зберігаються близько) за допомогою методів блокового розбиття (тайлінгу) для використання кеш-пам'яті L1/L2. Необхідно уникати псевдоспільного доступу, використовуючи вирівнювання даних (padding), щоб запобігти непотрібній інвалідації кеш-ліній між ядрами.
5. Стратегічне використання надшвидкої спільної пам'яті (Shared Memory): надшвидка спільна

пам'ять всередині тайла використовується для проміжного зберігання та обміну даними між потоками, що дозволяє уникнути повільних звернень до глобальної відеопам'яті (VRAM) у критичних інтенсивних операціях.

6. Зменшення синхронізаційних витрат та атомарні операції: для мінімізації затримок, спричинених традиційними блокуваннями, пріоритет надається безблокувальним структурам даних та атомарним операціям. Це гарантує, що дія буде виконана як єдина неподільна операція. Додатково ефективно використовується механізм редукції для безпечного паралельного злиття результатів без необхідності в ручному керуванні замками.

Програмно реалізовано задачу паралельного сортування масивів із використанням гібридного алгоритму Quick Sort, який поєднує послідовне виконання на CPU та паралельні операції порівняння на GPU за допомогою технології C++ AMP. У ході реалізації описано всі основні компоненти програмного коду, зокрема класичний алгоритм швидкого сортування, GPU-модуль паралельних порівнянь та механізм синхронізації даних між центральним і графічним процесорами.

Аналіз результатів тестування програми проведено для різної кількості елементів вхідного масиву. Для кожного набору даних вимірювався час виконання послідовного Quick Sort на CPU та гібридного Quick Sort із використанням GPU. На основі отриманих значень обчислювався коефіцієнт прискорення (табл. 1, 2).

Таблиця 1 – Час виконання сортування для різних розмірів масиву

Кількість елементів	Час CPU, мс	Час Hybrid (GPU), мс
1 000	0,002	0,060
5 000	0,012	0,065
10 000	0,035	0,070
50 000	0,290	0,180
100 000	0,650	0,320
500 000	4,120	1,050
1 000 000	9,870	2,100

Таблиця 2 – Коефіцієнти прискорення

Кількість елементів	Прискорення (разів)
1 000	0,03
5 000	0,18
10 000	0,50
50 000	1,61
100 000	2,03
500 000	3,92
1 000 000	4,70

Проаналізувавши отримані результати (табл.1, 2), можна зробити такі висновки.

- При невеликій кількості елементів масиву використання графічного процесора є недоцільним, оскільки витрати часу на ініціалізацію GPU та передачу даних між CPU і GPU перевищують виграш від паралельних обчислень. У цьому випадку послідовний Quick Sort демонструє кращу продуктивність.
- Зі збільшенням розміру вхідного масиву коефіцієнт прискорення поступово зростає, що

свідчить про ефективне використання паралельних обчислень на GPU. Починаючи з масивів середнього розміру, гібридна реалізація перевершує послідовний алгоритм за швидкодією.

- Найбільший ефект від застосування паралельного Quick Sort досягається при обробці великих масивів даних, де витрати на передачу даних компенсуються значним зменшенням часу виконання операцій порівняння за рахунок масового паралелізму GPU.

Отже, основними чинниками, що впливають на час виконання програми та коефіцієнт прискорення, є розмір вхідного масиву, накладні витрати на обмін даними між CPU та GPU, а також особливості гібридної реалізації алгоритму швидкого сортування. Використання технології C++ AMP у поєднанні з класичним Quick Sort дозволяє суттєво підвищити продуктивність сортування великих обсягів даних у гетерогенних обчислювальних системах.

Висновки

Досліджено алгоритм швидкого сортування як один із найефективніших алгоритмів упорядкування даних, що базується на принципі «розділяй і володарюй». Розглянуто особливості його послідовної та паралельної реалізацій, а також проаналізовано можливості застосування алгоритму Quick Sort у гетерогенних обчислювальних системах.

Програмно реалізовано гібридний алгоритм швидкого сортування з використанням технології C++ AMP, у якому поєднано послідовну обробку на CPU та паралельні операції порівняння на GPU. Описано основні компоненти програмного коду та виконано тестування розробленої програми для масивів різної розмірності.

За результатами експериментів встановлено, що зі збільшенням кількості елементів масиву ефективність паралельної реалізації зростає, а використання GPU дозволяє суттєво підвищити продуктивність сортування порівняно з послідовним виконанням на CPU.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. About sorting algorithms. URL: https://foxminded.ua/alhorytmy-sortuvannia/
2. Quick Sort Algorithm. URL: https://www.enjoyalgorithms.com/blog/quick-sort-algorithm
3. Kubliy L. I. “Algorithms and data structures” Kyiv: Igor Sikorsky Kyiv Polytechnic Institute, 2022. 318 p.
4. Microsoft. C++ AMP Overview. URL: C++ AMP Overview | Microsoft Learn
5. C++ AMP: Accelerated Massive Parallelism in Visual C++. URL: PowerPoint Presentation
6. GPU Computing. NVIDIA Developer Zone. URL: https://developer.nvidia.com/gpu-computing

Пащенко Анна Романівна – студентка кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м. Вінниця, e-mail: paschenkoanna2005@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.

Paschenko Anna Romanivna – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: paschenkoanna2005@gmail.com;

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