УДК 004.8

М.І. Мельничук
В.О. Денисюк

РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ ШВИДКОГО СОРТУВАННЯ

Вінницький національний технічний університет

Анотація

Розглянуто паралельний алгоритм швидкого сортування, реалізацію потокового графу та проведено математичне моделювання цього паралельного підходу. У процесі розробки було побудовано UML-діаграми класів та активності, що описують архітектуру та логіку роботи паралельного алгоритму швидкого сортування. Проведено оптимізацію програмного модуля та реалізовано його програмну частину. Виконано тестування реалізації з метою аналізу ефективності та швидкодії. Отримані результати можуть бути використані в різних програмних системах та алгоритмах для прискорення швидкого сортування.

Ключові слова: *паралельний алгоритм, паралельне сортування, швидке сортування, алгоритми сортування.*

Abstract

The parallel quick sort algorithm, the implementation of the flow graph, and the mathematical modeling of this parallel approach were considered. During the development process, UML class and activity diagrams were constructed that describe the architecture and logic of the parallel quick sort algorithm. The software module was optimized and its software part was implemented. The implementation was tested to analyze the efficiency and speed. The results obtained can be used in various software systems and algorithms to accelerate quick sort.

Keywords: *parallel algorithm, parallel sort, quick sort, sorting algorithms.*

Вступ

Сортування є основоположною операцією в обчислювальній техніці, яка знаходить застосування в різноманітних сферах, включаючи обробку великих даних, організацію баз даних, машинне навчання, пошук та аналіз інформації. З огляду на стрімке зростання обсягів даних, традиційні алгоритми сортування потребують удосконалення для забезпечення швидкості та ефективності обробки.

Паралельні алгоритми сортування, зокрема паралельний алгоритм швидкого сортування, дозволяють значно зменшити час обробки великих наборів даних за рахунок одночасної роботи на декількох процесорах чи ядрах. Це підходить для сучасних багатоядерних систем і відповідає вимогам до високопродуктивних обчислювальних середовищ.

Актуальність даної теми обумовлена необхідністю розробки та оптимізації алгоритмів сортування, які можуть ефективно працювати в умовах паралельних обчислень і відповідати сучасним вимогам до продуктивності. Результати цього дослідження можуть знайти застосування в таких сферах, як великі обчислення, фінансові технології, системи керування базами даних та штучний інтелект, підвищуючи ефективність обробки даних та продуктивність загальних систем.

Постановка задачі дослідження

Розв’язання задачі паралельної реалізації швидкого сортування вимагає:

- розробки ефективної архітектури паралельного алгоритму швидкого сортування для оптимального розподілу обчислень та зменшення часу виконання;
- розробки програми для реалізації паралельного швидкого сортування, проведення тестування програми та аналіз отриманих результатів.

Виклад основного матеріалу

Сортування є фундаментальною операцією, яку виконують більшість комп'ютерів [1]. Це базовий обчислювальний блок, що має важливе значення, і є однією з найбільш досліджених проблем в алгоритміці. Відсортованими даними легше маніпулювати, ніж випадково впорядкованими, тому багато алгоритмів потребують їх попереднього сортування. Сортування широко використовується в різноманітних програмах. Усі програми для роботи з електронними таблицями містять певний вид коду сортування. Програми баз даних, які використовуються в страхових компаніях, банках та інших установах, також мають вбудовані алгоритми сортування. Через важливість сортування у цих застосунках розроблено багато алгоритмів із різною обчислювальною складністю. Серед найпоширеніших алгоритмів варто виділити сортування вибором, бульбашкове сортування, сортування вставками, сортування злиттям, сортування купою та швидке сортування.

Швидке сортування або Quicksort — це алгоритм швидкого сортування, який розбиває великий масив даних на менші підмасиви [2, 3]. Це означає, що кожна ітерація розбиває вхідні дані на два компоненти, сортує їх і потім повторно комбінує. Цей метод дуже ефективний для великих наборів даних, оскільки його середня та найкраща складність становить $O(n \cdot \log n)$.

Алгоритм швидкого сортування може мати кілька недоліків, але на практиці він добре працює та може перевершувати інші алгоритми сортування за швидкістю та ефективністю простору.

Швидке сортування працює шляхом рекурсивного сортування підсписків по обидва боки від певної опорної точки та динамічного переміщення елементів у списку навколо цієї опорної точки.

З появою паралельної обробки паралельне сортування стало важливою областю дослідження. Більшість алгоритмів паралельного сортування поділяються на дві категорії: сортування на основі злиття та сортування на основі розділів. Сортування на основі злиття добре працює з невеликою кількістю процесорів, але при їх збільшенні накладні витрати на синхронізацію зростають, що зменшує прискорення. Сортування на основі розділу складається з розподілу набору даних на підмножини, у яких усі елементи в одній підмножині не більші за будь-який елемент в іншій, і подальшого паралельного сортування. Ефективність такого підходу залежить від рівномірності розподілу даних між підмножинами. Важко знайти опорні точки для рівномірного розбиття без попереднього сортування, що обмежує прискорення незалежно від кількості процесорів. Паралельна реалізація алгоритму швидкого сортування за підходом «розділяй і володарюй» підвищує швидкість, але її ефективність залежить від балансування навантаження. Швидке сортування не лише вважається одним із найефективніших алгоритмів сортування, але й є надійним алгоритмом, який можна паралелізувати. Ключовою особливістю паралельного швидкого сортування є паралельне розбиття даних. Паралельну генералізацію алгоритму швидкого сорту найпростіше реалізувати в мережі оброблювальних елементів, використовуючи топологію D-вимірного гіперкуба (тобто кількість оброблювальних елементів $p=2^D$).

Псевдокод паралельного алгоритму швидкого сортування такий [4, 5].

1. Розділити n значень даних на p рівних частин, які призначені для кожного процесора.
2. Обрати опорний елемент випадковим чином на першому процесорі P0 та передайте його всім процесорам.
3. Виконати глобальне сортування:
   3.1) локально в кожному процесорі розділити дані на два набори відповідно до опорного елемента (менші або більші);
   3.2) розділити процесори на дві групи та обміняти даними парно, так аби усі процесори в одній групі отримали дані, менші за опорний елемент, а інші — більші.
5. Повторити пункти 3.1 і 3.2 рекурсивно для кожної половини.
6. Кожен процесор сортує елементи, які у нього є, використовуючи алгоритм швидкого сортування.

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. *Вибір оптимального опорного елемента*. Вибір опорного елемента впливає на розподіл даних на підмасиви. Щоб уникнути найгіршого випадку, коли масив ділиться нерівномірно (наприклад, один підмасив містить майже всі елементи), застосовують наступні методи:

- медіана трьох - обирається середнє значення серед першого, середнього та останнього елементів масиву, це зменшує ймовірність вибору невдалого опорного елемента;
- рандомізований вибір - випадковий вибір опорного елемента зменшує ймовірність виникнення найгірших випадків.

2. *Використання гібридних алгоритмів*. У масивах невеликого розміру накладні витрати рекурсії можуть перевищувати вигоди від швидкого сортування. Тому в підмасивах, розмір яких не перевищує 10-15 елементів, доцільно використовувати сортування вставками, яке працює швидше на малих обсягах даних.

3. *Зменшення глибини рекурсії*. Рекурсивна природа швидкого сортування може призвести до надмірного використання пам’яті та викликати переповнення стека. Для уникнення цього застосовують такі підходи:

- лімітування глибини рекурсії - після досягнення певної глибини алгоритм переключається на інший метод сортування, наприклад, сортування злиттям;
- хвостова рекурсія - переписування алгоритму так, щоб зменшити кількість рекурсивних викликів.

4. *Паралельне виконання*. Швидке сортування добре підходить для паралелізації, оскільки підмасиви можна сортувати незалежно. Оптимізація паралельної версії включає:

- розподіл завдань між потоками або ядрами - після розбиття масиву на підмасиви вони обробляються одночасно;
- збалансований розподіл навантаження - забезпечується рівномірний розмір підмасивів для кожного ядра, щоб уникнути простоїв.

5. *Використання кеш-пам’яті*. Ефективність алгоритму можна підвищити, оптимізувавши його взаємодію з кеш-пам’яттю процесора:

- блокування доступу до даних - поділ масиву на блоки, які поміщаються в кеш, зменшує кількість звернень до повільнішої оперативної пам’яті;
- сортування блоків - обробка блоків цілком може значно покращити продуктивність на великих обсягах даних.

6. *Уникнення зайвих копіювань*. Для підвищення продуктивності варто мінімізувати кількість операцій копіювання. Це досягається шляхом реалізації сортування на місці (in-place), де обмін елементами відбувається без створення додаткових масивів.

7. *Оптимізація для розподілених систем*. У розподілених середовищах, таких як хмарні обчислення, оптимізація передбачає:

- попереднє розбиття даних - дані поділяються на підмасиви ще до початку виконання сортування;
- мінімізація міжпроцесорних комунікацій - зменшення кількості обміну даними між вузлами.

Результати дослідження.

Впроваджено такі методи оптимізації для паралельного швидкого сортування: вибір оптимального опорного елемента за допомогою методу медіани трьох; використання гібридних алгоритмів для малих масивів через сортування вставками; обмеження глибини рекурсії з переходом на стандартне швидке сортування; реалізація паралельного виконання за допомогою ProcessPoolExecutor; уникнення зайвих копіювань через сортування на місці; а також оптимізація для розподілених систем шляхом попереднього розбиття даних та мінімізації міжпроцесорних комунікацій.

Аналіз результатів тестування програми. Тестування програми проводили з вхідними даними різного розміру, фіксуючи при цьому час виконання. Це дозволило порівняти результати та оцінити ефективність розподілу задачі серед паралельних процесів (табл.1).

Таблиця 1 – Результати тестування програми

Кількість елементів масива	200	500	1 000	10 000	20 000	100 000
Час виконання, с	0.43352	0.37968	0.40678	0.48882	0.48687	0.76943

Аналізу результатів (табл.1) свідчить про те, що: час виконання програми залишається майже на одному рівні при зростанні кількості елементів у масиві до 100000 елементів; обраний метод обробки добре оптимізований для невеликих обсягів даних і не потребує значних покращень; основним фактором впливу на час виконання програми, є початкова ефективність алгоритму.

Висновки

Розглянуто алгоритм швидкого сортування, а також особливості паралельних алгоритмів і їх реалізацію. Детально проаналізовано математичне моделювання паралельного алгоритму, побудовано потоковий граф для його реалізації.

Програмно реалізовано паралельний алгоритм швидкого сортування з використанням модифікації, що включає обмеження глибини рекурсії, а також зміну модулю для обробки даних через кілька процесів з використанням ProcessPoolExecutor. Програмний модуль був оптимізований для покращення ефективності сортування великих обсягів даних.

Тестування програми довело стабільну швидкість при обробці масивів різного розміру, що свідчить про його ефективність у використанні для визначених обсягів даних.

Результати дослідження можуть бути використані для оптимізації інших паралельних алгоритмів сортування та в навчальних цілях для демонстрації практичного застосування паралельних обчислень.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Mdy M. Basic sorting. URL: https://dev.to/m__mdy__m/basic-sorting-5h20 .
2. Built In. Quicksort. URL: https://builtin.com/articles/quicksort .
3. Enjoy Algorithms. Quick Sort Algorithm. URL: https://www.enjoyalgorithms.com/blog/quick-sort-algorithm .
4. Singh T. Performance Comparison of Sequential Quick Sort and Parallel Quick Sort Algorithms. URL: https://www.researchgate.net/profile/Tinku-Singh/publication/366138697_Performance_comparison_of_sequential_quick_sort_and_parallel_quick_sort_algorithms/links/6392fd7c484e65005bf85f6c/Performance-comparison-of-sequential-quick-sort-and-parallel-quick-sort-algorithms.pdf
5. Antas. Parallel Quicksort Algorithm. URL: https://243-antas.medium.com/parallel-quicksort-algorithm-991cbfc94adc .

Мельничук Марина Іванівна – студентка кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: marinamelnicuk71@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.

Melnychuk Maryna Ivanovna – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: marinamelnicuk71@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