УДК 004.8

Б.П. Янковський
В.О. Денисюк

ДОСЛІДЖЕННЯ ПАРАЛЕЛЬНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ RADIX SORT

Вінницький національний технічний університет

Анотація

Розглянуто питання аналізу відомих методів розв'язання задачі паралельного сортування та методи реалізації паралельного алгоритму сортування Radix Sort. Одержані результати можливо використовувати у різноманітних алгоритмах та програмних засобах для збільшення швидкодії сортування.

Ключові слова: *паралельний алгоритм, паралельне сортування, сортування чисел, Radix Sort.*

Abstract

The issue of analyzing known methods for solving the parallel sorting problem and methods for implementing the parallel sorting algorithm Radix Sort is considered. The results obtained can be used in various algorithms and software tools to increase the sorting speed.

Keywords: *parallel algorithm, parallel sorting, Radix Sort, integer sorting.*

Вступ

Задача паралельного сортування займає важливе місце в галузі комп'ютерних наук, особливо в контексті обробки великих обсягів даних та високопродуктивних обчислень. Її актуальність зумовлена постійно зростаючими потребами в швидкій та ефективній обробці інформації в різноманітних сферах, від наукових досліджень до бізнес-аналітики. Розв'язання цієї задачі має критичне значення для оптимізації роботи сучасних комп'ютерних систем, оскільки сортування є фундаментальною операцією, що лежить в основі багатьох складніших алгоритмів та процесів обробки даних.

Постановка задачі дослідження

При розв'язанні задачі дослідження паралельної реалізації алгоритму Radix Sort необхідно:

- провести аналіз предметної області паралельного сортування;
- розробити реалізації паралельного алгоритму сортування Radix Sort.

Виклад основного матеріалу

Галузь паралельного сортування виникла як відповідь на зростаючі обсяги даних та потребу в їх швидкій обробці. Паралельні алгоритми сортування використовуються замість лінійних для підвищення швидкості й ефективності обробки великих обсягів даних [1].

На відміну від лінійних методів, що обробляють елементи послідовно, паралельні алгоритми розподіляють навантаження між кількома процесорами або ядрами, що дозволяє одночасно виконувати різні частини завдання. Це значно скорочує час виконання, особливо при роботі з великими масивами даних, оскільки завдання сортування ділиться на менші підзадачі, які обробляються незалежно і паралельно. У результаті обробка завершується швидше, ніж у випадку лінійного підходу. Також паралельні алгоритми забезпечують кращу масштабованість: збільшуючи кількість процесорів або ядер, можна прискорити обробку без значної зміни алгоритму. Це робить їх ідеальними для застосування в суперкомп'ютерах, дата-центрах та обробці потоків даних у реальному часі. Вони також ефективні для складних операцій, таких як великі бази даних або аналіз великих даних, де обробка величезних обсягів інформації вимагає максимальної продуктивності [2].

При цьому, лінійні алгоритми сортування мають кілька переваг перед паралельними, особливо в контексті простоти реалізації та ресурсоємності. Однією з основних переваг є те, що вони не потребують додаткових обчислювальних ресурсів для синхронізації та управління потоками, що знижує ймовірність помилок, пов'язаних із паралельним доступом до даних. Це робить лінійні алгоритми простішими для реалізації, налагодження та оптимізації. Вони також не потребують розподілу та збору даних між потоками, тому зменшуються затримки, які можуть виникати через паралельний доступ до пам’яті. Важливо, що лінійні алгоритми підходять для виконання на системах із обмеженими ресурсами, де паралельне обчислення є неможливим або недоцільним через високу вартість чи низьку ефективність багатопоточності. Також, у випадку невеликих обсягів даних, лінійні алгоритми можуть працювати швидше, оскільки паралельні алгоритми мають додаткові витрати на налаштування потоків, що стає непотрібним для невеликих задач [3].

Radix Sort представляє особливий інтерес у контексті паралельного сортування. Цей алгоритм, на відміну від порівняльних методів сортування, базується на порозрядному порівнянні чисел [4]. У паралельній реалізації Radix Sort кожен розряд чисел може оброблятися окремим процесором або групою процесорів. Процес включає розподіл елементів за "блоками" відповідно до значення поточного розряду, а потім об'єднання цих "відер" у правильному порядку. Паралельний Radix Sort особливо ефективний для сортування цілих чисел та рядків фіксованої довжини.

Паралельний алгоритм сортування по розрядах має кілька важливих характеристик, які визначають його ефективність, зокрема часову та просторову складність, прискорення і ефективність. У паралельному алгоритмі сортування по розрядах дані розподіляються між процесорами таким чином, що кожен процесор обробляє один або кілька розрядів, за умови достатньої кількості процесорів. При цьому, кожна обробка розряду займає $O(n+b)$ часу, де b є сталою, що представляє основу системи числення (звичайно 10 для десяткової системи). Якщо кожен процесор обробляє один розряд, то максимальна кількість процесорів, які можуть працювати одночасно, становить d, що дорівнює числу розрядів.

Загальний час для паралельного сортування при використанні p процесорів і розподілі роботи між ними, необхідний для обробки всіх розрядів одночасно, дорівнює $O(d/p \cdot (n+b))$.

Якщо кількість процесорів дорівнює або перевищує кількість розрядів, то часова складність досягає мінімуму - $O(n)$, що вказує на теоретичну можливість значного прискорення обчислень. Прискорення паралельного алгоритму сортування по розрядах визначається як відношення часу роботи послідовного алгоритму до часу роботи паралельного алгоритму. Для послідовного сортування по розрядах цей час становить $O(d \cdot (n+b))$, що дає прискорення в $O(p)$, тобто за умови відсутності накладних витрат це є лінійне прискорення. Проте реальне прискорення завжди дещо нижче через накладні витрати на синхронізацію та комунікацію між процесорами.

Ефективність паралельного алгоритму визначається як відношення прискорення до кількості процесорів, і за оптимального розподілу обчислень між процесорами вона дорівнює $O(1)$, що означає майже ідеальне використання ресурсів системи. Проте, якщо кількість процесорів значно перевищує d, ефективність падає, оскільки деякі процесори простоюють через обмежену кількість розрядів. Щодо просторової складності, кожен процесор зберігає свою підмножину елементів і має виділений простір для розподілу в b кошиках. Просторова складність на кожен процесор становить $O(n/p+b)$, а загальна просторова складність всієї системи $O(n+b \cdot p)$, що зростає зі збільшенням кількості процесорів.

Серед підходів до паралелізації виділяють два основні: за розрядами та за блоками.

Метод паралелізації за розрядами має лінійну структуру обробки даних із чіткими етапами, що дозволяє послідовно виконувати обчислення. На етапі ініціалізації та розподілу даних вхідний масив ділиться на p частин, де p відповідає кількості процесорів. Кожен процесор отримує свій підмасив для обробки, і відбувається початкова синхронізація процесорів. Обробка розрядів включає послідовну обробку кожного з них, де для кожного розряду здійснюється локальний підрахунок частот. Далі формується глобальна префіксна сума, після чого відбувається перерозподіл елементів між процесорами. Етап синхронізації та комунікації включає обмін даними

між процесорами після обробки кожного розряду. Відбувається оновлення локальних даних процесорів і перевірка умов завершення обробки розряду.

У методі паралелізації за розрядами [5] існують ключові залежності, що визначають послідовність дій. Послідовні залежності включають необхідність обробляти розряди послідовно від молодшого до старшого, оскільки для обробки кожного наступного розряду потрібні результати обробки попереднього. Паралельні можливості в цьому методі дозволяють виконувати підрахунок частот одночасно для різних підмасивів. Перерозподіл елементів між процесорами також можна частково паралелізувати.

Метод паралелізації за блоками має деревоподібну структуру з можливістю адаптивного розподілу навантаження між процесорами, що сприяє ефективній обробці. На етапі формування блоків спочатку виконується аналіз розподілу значень у вхідному масиві. Після цього створюються b блоків, де b - основа системи числення, і здійснюється початковий розподіл елементів між ними. Етап паралельної обробки блоків передбачає розподіл блоків між процесорами, локальне сортування елементів у кожному з них і балансування навантаження між процесорами. Злиття результатів відбувається через ієрархічне з'єднання відсортованих послідовностей. Потім виконується фінальне впорядкування даних, і результати збираються на головному процесорі.

Метод паралелізації за блоками має свої особливості комунікації, що впливають на продуктивність. Комунікаційні патерни охоплюють *all-to-all* комунікацію при розподілі елементів, *point-to-point* комунікацію при балансуванні та колективні операції під час злиття результатів. Для оптимізації комунікацій використовуються асинхронні методи передачі, зменшення обсягу переданих даних та оптимізація топології комунікацій.

Метод розрядів має низку переваг: він простіший у синхронізації, передбачуваний у патернах комунікацій і забезпечує ефективне використання кеш-пам'яті. Однак метод розрядів має і недоліки, серед яких багато точок синхронізації, нижча гнучкість при балансуванні та наявність послідовної залежності між розрядами.

Метод блоків, зі свого боку, має переваги у вигляді кращої масштабованості, гнучкого балансування навантаження та меншої кількості точок глобальної синхронізації. Недоліками методу блоків є складніша реалізація, можливість виникнення нерівномірного навантаження та більші накладні витрати на комунікацію [6].

Результати дослідження.

Розроблено програмний модуль Radix sort, який використовує інші алгоритми сортування на етапі локального сортування. Для реалізації було обрано сортування підрахуванням та сортування блоками через використання ними сильних сторін умов алгоритму паралельного Radix sort.

Програма вимагає 64-бітний центральний процесор та наявність на обчислювальній машині інтерпретатора Python 3 та його стандартної бібліотеки.

Тестування програми проводилося на 4-х різних процесорах архітектури Intel x86 – Intel Core i7-13700H (14 ядер), AMD Ryzen 7 7800X3D (8 ядер) та Intel Xeon Skylake (4 ядра) (таб.1).

Можна спостерігати, що ефективність програми значно підвищується зі збільшенням кількості процесорних ядер. На низькій довжині вхідного набору Ryzen 7 показує перевагу за рахунок кращої однопоточної продуктивності, доки на більших наборах Core i7 випереджає його за рахунок більшої кількості ядер.

Висновки

Було розглянуто різні варіанти реалізації паралельного алгоритму сортування Radix Sort, а саме метод блоків та розрядів. Серед двох можливих варіантів його реалізації, сортування з паралелізацією за блоками працює від 1.5 до 2 раз швидше на однаковому об'ємі випадкових даних у порівнянні з паралелізацією за розрядами.

Розроблена реалізація може використовуватися для подальшого дослідження паралельного алгоритму сортування Radix Sort, використання його у програмному забезпеченні. Подальші

дослідження можуть бути зосереджені на порівнянні ефективності алгоритму з і іншими методологіями реалізації алгоритмів сортування.

Таблиця 1. – Результати тестування програми

Довжина вхідного масиву	Кількість розрядів	Час виконання, с		
		4 ядра	8 ядер	14 ядер
1 000	6	0.007052	0.000913	0.001297
1 000	12	0.015438	0.001704	0.002323
1 000	18	0.024564	0.002450	0.003173
10 000	6	0.039061	0.004161	0.003011
10 000	12	0.119595	0.010540	0.009372
10 000	18	0.190667	0.017132	0.017502
100 000	6	0.367426	0.035187	0.030330
100 000	12	0.941536	0.103621	0.087909
100 000	18	1.621116	0.167283	0.143100

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Casanova, H. Parallel Algorithms. URL: https://freecomputerbooks.com/Parallel-Algorithms-by-Henri-Casanova.html
2. Barney, B. Introduction to Parallel Computing. Lawrence Livermore National Laboratory. URL: https://www.lrde.epita.fr/~ricou/intro_parallel_comp.pdf
3. Parallel Programming and Algorithms. Coursera. URL: https://www.coursera.org/learn/scala-parallel-programming
4. GeeksforGeeks. Radix Sort. URL: https://www.geeksforgeeks.org/radix-sort/
5. Various. Counting Sort. GeeksforGeeks. URL: https://www.geeksforgeeks.org/counting-sort/
6. Various. Block Sort. GeeksforGeeks. URL: https://www.geeksforgeeks.org/block-sort/

Янковський Богдан Петрович – студент кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: bohdan.yankovskyi@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.

Yankovskyi Bohdan Petrovych – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: bohdan.yankovskyi@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