УДК 004.8

А.О. Алексішін
В.О. Денисюк

ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ СОРТУВАННЯ PARALLEL MERGE SORTNET

Вінницький національний технічний університет

Анотація

Досліджено та реалізовано паралельний алгоритм сортування злиттям. Проаналізовано основні алгоритми сортування, принципи паралельних алгоритмів, розроблено та реалізовано паралельний алгоритм сортування злиттям, а також проведено тестування на продуктивність. Отримані результати свідчать про доцільність використання паралельності для оптимізації процесу сортування.

Ключові слова: *паралельне сортування злиттям, порівняння алгоритмів сортування, тестування алгоритму сортування.*

Abstract

Researched and implemented parallel merge sort algorithm. analyzed the basic sorting algorithms, principles of parallel algorithms, developed and implemented parallel merge sort algorithm, and conducted performance testing. The results obtained indicate the feasibility of using parallelism to optimize the sorting process.

Keywords: *: parallel merge sort, sorting algorithm comparison, sorting algorithm testing.*

Вступ

Актуальність теми дослідження зумовлена стрімким зростанням обсягів інформації у сучасному цифровому світі, що вимагає від програмного забезпечення максимально високої швидкості обробки даних. Сортування масивів залишається однією з найбільш фундаментальних та часто використовуваних операцій у програмуванні, ефективність якої безпосередньо впливає на продуктивність цілих програмних систем [1].

Зі зростанням обсягів інформації все більшого значення набувають методи, здатні ефективно використовувати багатоядерні та багатопотокові середовища. Сортування злиттям традиційно розглядається як надійний алгоритм із передбачуваною складністю $O(n\log n)$ та високою стабільністю, а його рекурсивна структура поділу даних на підмасиви робить метод ідеальним кандидатом для розпаралелювання [1, 2]. Проте пряме використання паралельності не завжди дає очікуваний ефект через накладні витрати на координацію потоків, що створює потребу в оптимізованих підходах, які враховують особливості сучасного апаратного забезпечення.

Дослідження можливостей підвищення швидкодії через комбінацію паралельного та послідовного режимів дозволяє суттєво зменшити системні витрати.

Постановка задачі дослідження

Метою роботи є дослідження, всебічний аналіз та розробка паралельного алгоритму сортування злиттям, а також оцінка впливу різних оптимізацій на реальну продуктивність обчислень. Для досягнення цієї мети було поставлено завдання провести теоретичний огляд існуючих алгоритмів сортування, обґрунтувати вибір технологічного стека та реалізувати три варіації методу:

- класичну послідовну;
- базову паралельну;
- оптимізовану гібридну версію.

Важливим аспектом дослідження є порівняння ефективності розробленого рішення з класичним підходом на різних обсягах даних - від малих списків до масивів розміром у сто мільйонів елементів.

Виклад основного матеріалу

Сортування масивів є однією із найбільш фундаментальних операцій у програмуванні, ефективність якої безпосередньо впливає на загальну продуктивність складних систем. При виборі конкретного алгоритму зазвичай оцінюють часову складність у найкращому, середньому та найгіршому випадках, стабільність, що дозволяє зберігати первинний порядок рівних елементів, та адаптивність до частково впорядкованих даних. Сортування злиттям (Merge Sort) традиційно розглядається як надійний метод із передбачуваною складністю у всіх сценаріях виконання. Завдяки своїй рекурсивній структурі, заснованій на принципі «розділяй і володарюй», цей алгоритм є ідеальним кандидатом для розпаралелювання, оскільки поділ даних на незалежні підмасиви дозволяє виконувати їх обробку паралельно [1].

Паралельне сортування злиттям являє собою адаптовану версію класичного методу для багатопотокових обчислювальних середовищ, де обробка кожної гілки рекурсивного дерева здійснюється в окремих незалежних потоках. Це дозволяє ефективно задіяти потужність кількох ядер сучасних процесорів, значно скорочуючи час обробки великих обсягів інформації. Однак пряме використання паралельності не завжди дає очікуваний ефект через значні накладні витрати на ініціалізацію підпроцесів, системні виклики та міжпотокову синхронізацію. Практика показує, що на глибоких рівнях рекурсії або при роботі з масивами малого розміру час, витрачений на керування потоками, стає більшим за користь від самої паралельної роботи, що нівелює переваги багатопотоковості.

Для вирішення проблеми системних витрат доцільно застосовувати оптимізований гібридний підхід, який поєднує паралельний та послідовний алгоритми. Основою такої оптимізації є впровадження механізму визначення порогового значення для поділу масиву. Якщо розмір підмасиву перевищує встановлений поріг, виконується паралельний поділ у нових потоках, але як тільки обсяг даних стає меншим за критичне значення, алгоритм автоматично перемикається на стандартне послідовне сортування. Це гарантує, що паралельні потужності використовуватимуться лише для масштабних завдань, де економія часу на обчислення є значно вищою за витрати на контроль підпроцесів [3-5].

Додатковим напрямком оптимізації є вдосконалення стратегії керування потоками та мінімізація операцій у пам'яті. Замість створення двох нових потоків для правої та лівої гілок на кожному кроці рекурсії, раціональніше ініціювати створення лише одного потоку для правої гілки, продовжуючи виконання лівої в межах поточного оригінального потоку. Такий підхід підтримує всі ядра процесора в активному стані та вдвічі зменшує загальну кількість створених потоків і витрати на контекстне перемикання. Крім того, модифікація алгоритму за принципом Ping-pong Merge Sort дозволяє скоротити кількість переміщень елементів майже вдвічі шляхом почергового використання оригінального масиву та допоміжного буфера як джерела і результату на різних рівнях рекурсії.

Програмна реалізація дослідженого алгоритму була виконана мовою Java з використанням спеціалізованого фреймворку Fork/Join. Даний інструмент нативно підтримує механізм «викрадання роботи» (work-stealing), який дозволяє потокам, що звільнилися, автоматично забирати завдання у більш навантажених підпроцесів, забезпечуючи рівномірне балансування навантаження між ядрами. Використання JIT-компіляції у середовищі виконання Java дозволяє досягти високої продуктивності обробки великих даних, що наближається до мов із прямою компіляцією, за рахунок агресивних оптимізацій байт-коду в машинний код під час виконання програми [5-7].

Реалізація програми виконано для різної кількості елементів масиву (табл.1), де час виконання є сумарним часом всіх пробігів.

Аналіз результатів тестування програми. Аналіз експериментальних результатів тестування на масивах розміром від тисячі до ста мільйонів елементів чітко підтвердив переваги комбінованого методу (рис.1). На малих обсягах даних (1 000 елементів) класичний послідовний алгоритм продемонстрував кращий час виконання (0,562 с) порівняно зі звичайним паралельним методом (0,843 с), що пояснюється надмірними витратами на керування потоками для малих задач. Однак при обробці 100 мільйонів елементів ситуація кардинально змінилася: оптимізована паралельна версія з граничним значенням впоралася за 6,2 секунди, що майже у 6 разів швидше за послідовний варіант

(38 секунд) та у 1.2 рази швидше за звичайний паралельний варіант (7,3 секунди). Таким чином, саме гібридний метод сортування злиттям виявився найбільш універсальним рішенням, оскільки він забезпечує високу швидкість обробки великих масивів завдяки паралелізму та відсутність затримок на малих підзадачах завдяки переходу до послідовного виконання.

Таблиця 1 – Результати тестування методів сортування

Розмір списку (кількість пробігів)	Merge Sort, с	Parallel Merge Sort, с	Parallel Merge Sort With Cutoff, с
1 000 (x10 000)	0,562	0,843	0,523
100 000 (x 500)	4,108	1,139	0,930
10 000 000 (x 15)	16,899	3,315	2,898
100 000 000 (x 3)	38,001	7,332	6,231

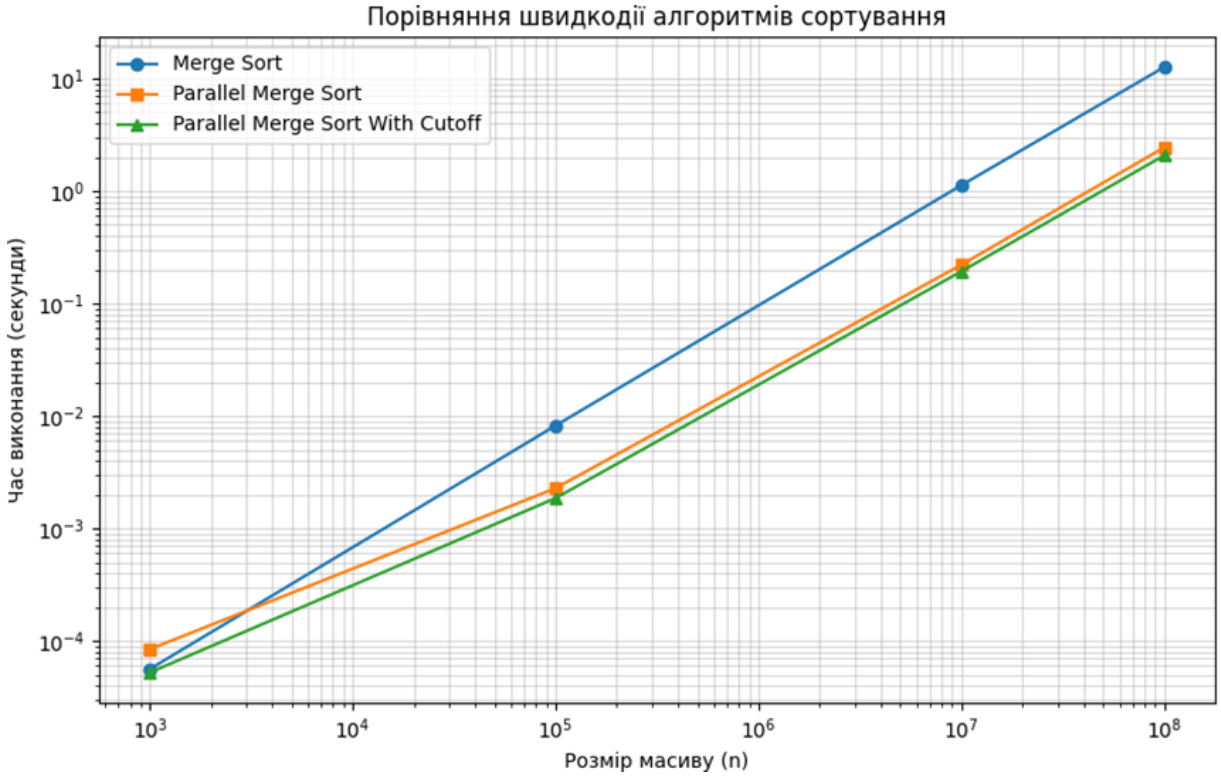

Рисунок 1 – Середній час виконання сортування на логарифмічному графіку

Висновки

У ході роботи досліджено та реалізовано гібридний паралельний алгоритм сортування злиттям на мові Java. Теоретичний аналіз підтвердив, що структура Merge Sort ідеально підходить для розпаралелювання завдяки рекурсивному поділу задач. Експериментально доведено, що пряма паралелізація на малих масивах є неефективною через системні витрати на керування потоками.

Найкращі результати показала оптимізована версія з пороговим значенням (Cutoff), яка при досягненні 1000 елементів перемикається на послідовний режим. Використання фреймворку Fork/Join із механізмом «викрадання роботи» забезпечило рівномірне навантаження ядер процесора. На масивах у 100 мільйонів елементів вдалося досягти шестикратного прискорення порівняно з

класичним методом. Таким чином, комбінований підхід є найбільш продуктивним рішенням для сучасних систем обробки великих даних

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Рогушина Ю. В. Алгоритм сортування // Велика українська енциклопедія. URL: https://vue.gov.ua/Алгоритм_сортування
2. Java Sorting Algorithm: Exercises, Practice, Solution. URL: https://www.w3resource.com/java-exercises/sorting/index.php
3. Changing std::sort at Google's Scale and Beyond. URL: https://danlark.org/2022/04/20/changing-stdsort-at-googles-scale-and-beyond/
4. Parallel Merge Sort. URL: https://redixhumayun.github.io/systems/2023/12/29/parallel-merge-sort.html
5. Parallel mergesort. URL: https://tarjotin.cs.aalto.fi/CS-A1140/2020Summer/notes/par-mergesort.html
6. scandum/quadsort. URL: https://github.com/scandum/quadsort?tab=readme-ov-file
7. Create a Recursive Solution Using Fork/Join. URL: https://openclassrooms.com/en/courses/5684021-scale-up-your-code-with-java-concurrency/6660941-create-a-recursive-solution-using-fork-join

Алексішін Андрій Олександрович – студент кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: andrii.antidot@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.

Aleksishin Andrii Olexandrovich – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: andrii.antidot@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 .