УДК 004.8

Ю.О.Загон
В.О. Денисюк

РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ БУЛЬБАШКОВОГО СОРТУВАННЯ

Вінницький національний технічний університет

Анотація

Розглянуто дослідження алгоритму бульбашкового сортування та можливості його паралельної реалізації з використанням бібліотеки OpenMP і стандартних засобів багатопотоковості мови C++ (std::thread). Проаналізовано основні поняття сортування, принципи роботи бульбашкового алгоритму та особливості паралельних обчислень. Обґрунтовано вибір засобів розробки та підходів до реалізації паралельного алгоритму. У роботі створено послідовну та паралельні реалізації алгоритму бульбашкового сортування та проведено експериментальне дослідження їх продуктивності на масивах різного розміру. Результати експериментів показали, що використання багатопоточності дозволяє суттєво підвищити швидкодію сортування великих масивів даних при оптимальній кількості потоків, тоді як надмірна кількість потоків призводить до зниження ефективності через накладні витрати на синхронізацію та обмеження пропускної здатності пам'яті. Отримані результати підтверджують доцільність використання паралельних обчислень для прискорення алгоритмів сортування.

Ключові слова: *бульбашкове сортування, OpenMP, багатопоточність, C++, продуктивність, паралельні обчислення.*

Abstract

The study of the bubble sort algorithm and the possibilities of its parallel implementation using the OpenMP library and standard C++ multithreading support (std::thread) is considered. The basic concepts of sorting, the principles of the bubble sort algorithm, and the features of parallel computing are analyzed. The choice of development tools and approaches for implementing the parallel algorithm is justified. A sequential and parallel implementations of the bubble sort algorithm are developed, and their performance is experimentally evaluated on data sets of various sizes. The experimental results show that the use of multithreading significantly improves performance when sorting large data arrays with an optimal number of threads, while further increasing the number of threads does not always lead to performance gains due to synchronization overhead and memory bandwidth limitations. The obtained results confirm the effectiveness of parallel computing for accelerating sorting algorithms.

Keywords: *bubble sort, OpenMP, multithreading, C++, performance, parallel computing.*

Вступ

Актуальність роботи зумовлена зростанням обсягів даних, що потребують швидкої та ефективної обробки в сучасних обчислювальних системах. Сортування великих масивів даних є важливою складовою багатьох прикладних задач у науці, техніці та інформаційних технологіях. Одним із базових алгоритмів сортування є бульбашкове сортування, яке наочно демонструє принципи впорядкування даних шляхом обміну сусідніх елементів [1-6].

Класичне послідовне бульбашкове сортування неефективно використовує можливості багатоядерних процесорів. Тому застосування паралельних обчислень із використанням OpenMP та стандартної багатопотоковості C++ (std::thread) дозволяє підвищити продуктивність сортування великих масивів даних [4, 7, 8].

Метою роботи є дослідження та реалізація паралельного алгоритму бульбашкового сортування з метою зменшення часу виконання алгоритму та підвищення швидкодії обчислювальних систем.

Постановка задачі дослідження

Завданнями, що ставляться в ході роботи, є:

- провести аналіз класичного алгоритму бульбашкового сортування та принципів паралельних обчислень;
- розробити блок-схему паралельного алгоритму сортування та обґрунтувати вибір технології реалізації (OpenMP / std::thread);
- реалізувати паралельний алгоритм бульбашкового сортування на мові C++ у середовищі Visual Studio Code;
- перевірити розроблений алгоритм на коректність сортування та виконати тестування продуктивності для різної кількості потоків;
- проаналізувати результати експериментальної частини та визначити можливості оптимізації алгоритму.

Виклад основного матеріалу

Сортування є однією з фундаментальних операцій у комп'ютерних науках та використовується в широкому колі прикладних задач: від попередньої обробки даних і оптимізації пошуку до побудови ефективних структур даних, роботи з базами даних та реалізації паралельних обчислень. Упорядкування даних дозволяє значно зменшити обчислювальну складність подальших операцій, зокрема пошуку, фільтрації та агрегації інформації, що робить алгоритми сортування ключовим елементом сучасних інформаційних систем.

Ефективність алгоритмів сортування оцінюється за кількома основними критеріями:

- часовою складністю;
- просторовою складністю;
- стабільністю;
- адаптивністю;
- можливістю використання внутрішнього або зовнішнього сортування.

Залежно від підходу до реалізації алгоритми сортування поділяються на прості (базові) та високоефективні, а також на алгоритми, що базуються на порівнянні елементів, і алгоритми без порівнянь, які працюють із розрядами або ключами даних.

До простих алгоритмів сортування належать сортування вибором, сортування вставками та бульбашкове сортування. Ці алгоритми характеризуються простою логікою реалізації та наочністю, проте мають квадратичну часову складність, що обмежує їх ефективність при роботі з великими масивами даних. Більш складні та продуктивні алгоритми, такі як швидке сортування (Quick Sort), сортування злиттям (Merge Sort) та сортування за розрядами (Radix Sort), забезпечують значно кращу продуктивність, зокрема часову складність порядку $O(n \log n)$ або навіть $O(n)$ для окремих випадків [1-7]. Особливе місце серед простих алгоритмів займає бульбашкове сортування. Його принцип роботи полягає у багаторазовому проході масиву з попарним порівнянням сусідніх елементів та їх обміном у разі порушення порядку. Після кожного проходу найбільший елемент «спливає» до кінця масиву, що і зумовило назву алгоритму. Незважаючи на простоту та стабільність, основним недоліком бульбашкового сортування є його низька продуктивність при обробці великих обсягів даних через часову складність $O(n^2)$ [1, 4, 7].

Крім того, класичне бульбашкове сортування є внутрішньо послідовним алгоритмом. Кожна ітерація залежить від результатів попередньої, що унеможливлює ефективну паралелізацію без виникнення гонок даних або значних накладних витрат на синхронізацію. Це робить стандартну реалізацію алгоритму непридатною для повного використання обчислювальних можливостей багатоядерних процесорів.

З розвитком багатоядерних архітектур та технологій паралельного програмування зросла актуальність розробки та аналізу паралельних алгоритмів. Основними принципами побудови

паралельних алгоритмів є декомпозиція задачі, паралельне виконання незалежних підзадач, синхронізація результатів, балансування навантаження та уникнення блокувань і гонок даних. Дотримання цих принципів дозволяє суттєво підвищити продуктивність обчислень і ефективно використовувати апаратні ресурси системи.

Для паралельної реалізації бульбашкового сортування доцільно використовувати його модифікацію - алгоритм парно-непарного сортування (Odd-Even Transposition Sort). Цей алгоритм зберігає базову ідею попарних обмінів сусідніх елементів, проте організовує обчислення у дві чергуючі фази: парну та непарну. У межах кожної фази порівнюються незалежні пари елементів, що дозволяє виконувати обчислення паралельно без конфліктів доступу до даних.

Завдяки такій структурі Odd-Even Transposition Sort є практично єдиним ефективним паралельним аналогом бульбашкового сортування, який широко використовується в задачах паралельних обчислень. Його застосування дозволяє значно скоротити час сортування для середніх і великих масивів даних за умови правильного вибору кількості потоків та оптимального використання ресурсів багатоядерних процесорів.

Таким чином, аналіз класичних алгоритмів сортування та принципів паралельних обчислень підтверджує доцільність використання модифікованих алгоритмів, таких як Odd-Even Transposition Sort, для реалізації паралельного сортування. Це створює теоретичне підґрунтя для програмної реалізації та експериментального дослідження продуктивності паралельного бульбашкового сортування [9, 10].

Програмно реалізовано задану задачу та описано всі компоненти коду.

Аналіз результатів тестування програми виконано для різної кількості розмірів масиву (табл.1-3).

Таблиця 1 – Час виконання програми на різних кількостях потоків (с)

Кількість потоків	Розмір масиву			
	100	1 000	10 000	100 000
1	0.04196	1.10507	71.213	6 896.89
2	0.08202	1.25480	37.151	3 492.73
4	0.09703	1.34277	24.946	2 359.77
8	2.92301	3.99399	22.224	1 691.52
16	6.43492	68.65910	274.523	3 857.74
32	11.74500	83.49490	864.805	9 762.10

Таблиця 2 – Коефіцієнти прискорення

Кількість потоків	Розмір масиву			
	100	1 000	10 000	100 000
1	1.00000	1.0000	1.000	1.000
2	0.06100	0.398	1.917	1.975
4	0.05200	0.372	2.855	2.923
8	0.00170	0.125	3.204	4.077
16	0.00008	0.007	0.259	1.788
32	0.00040	0.006	0.082	0.706

Таблиця 3 – Коефіцієнти ефективності

Кількість потоків	Розмір масиву			
	100	1 000	10 000	100 000
1	1	1	1	1
2	0.031	0.199	0.958	0.988
4	0.013	0.093	0.714	0.731
8	0.0002	0.016	0.401	0.510
16	0.00005	0.00004	0.016	0.112
32	0.00001	0.0002	0.003	0.022

Проаналізувавши результати експериментальних досліджень (табл.1-3) часу виконання, коефіцієнтів прискорення та ефективності паралельного бульбашкового сортування для різних розмірів вхідних масивів і різної кількості потоків, можна зробити такі висновки:

- для малих вхідних даних значення коефіцієнта прискорення є меншими за одиницю, а коефіцієнт ефективності наближається до нуля, що свідчить про недоцільність використання паралельного підходу через значні накладні витрати на створення потоків, ініціалізацію OpenMP-середовища та синхронізацію між потоками;
- зі збільшенням розміру вхідного масиву ефективність паралельного сортування зростає, і для середніх та великих масивів досягається суттєве прискорення порівняно з послідовною реалізацією, що підтверджує доцільність використання багатопоточності для обробки великих обсягів даних;
- експериментально встановлено наявність оптимальної кількості потоків, при якій досягається максимальне прискорення та найкращий баланс між обчислювальним навантаженням і накладними витратами, при цьому у даній реалізації оптимальною є кількість потоків, рівна 8;
- подальше збільшення кількості потоків понад оптимальне значення не приводить до зростання продуктивності та, навпаки, знижує коефіцієнт ефективності через конкуренцію за ресурси оперативної пам’яті, погіршення кеш-локальності та зростання витрат на синхронізацію.

Отже, основними чинниками, що впливають на зміну показників продуктивності, таких як час виконання, коефіцієнт прискорення та коефіцієнт ефективності, є розмір вхідного масиву, кількість потоків у програмній реалізації, архітектура обчислювальної системи та особливості використаного алгоритму сортування. Отримані результати підтверджують, що паралельне бульбашкове сортування є доцільним лише для достатньо великих обсягів даних та за умови раціонального вибору кількості потоків, що дозволяє мінімізувати накладні витрати і максимально використати потенціал багатоядерних процесорів.

Висновки

У роботі досліджено алгоритм бульбашкового сортування як один із базових алгоритмів впорядкування даних та проаналізовано його обмеження щодо продуктивності. Визначено, що класичне послідовне бульбашкове сортування має квадратичну часову складність та не піддається ефективній паралелізації через залежність ітерацій і необхідність синхронізації. Для розв’язання цієї проблеми обґрунтовано вибір алгоритму Odd-Even Transposition Sort як паралельного аналога бульбашкового сортування, що дозволяє безпечний розподіл обчислень між потоками.

Програмно реалізовано паралельний алгоритм сортування з використанням директив OpenMP. Проведено тестування коректності роботи алгоритму та експериментальне дослідження продуктивності для різної кількості потоків. Отримані результати показали, що для малих масивів паралелізація є недоцільною через накладні витрати, тоді як для середніх і великих масивів

досягається суттєве прискорення. Встановлено, що оптимальна продуктивність забезпечується при використанні обмеженої кількості потоків, а подальше їх збільшення призводить до зниження ефективності.

Результати роботи підтверджують доцільність застосування раціональної паралелізації для підвищення швидкодії алгоритмів сортування та демонструють ефективність використання OpenMP у задачах паралельних обчислень.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Про алгоритми сортування. URL: https://foxminded.ua/alhorytmy-sortuvannia/
2. Сортування вибором. URL: https://aliev.me/runestone/SortSearch /TheSelectionSort.html
3. Quicksort: історія виникнення та розвитку "найшвидшого" алгоритму сортування. URL: https://phm.cuspu.edu.ua/nauka/naukovo-populiarni-publikatsii/824-quicksort-istoriia-vynyknennia-ta-rozvytku-naishvydshoho-alhorytmu-sortuvannia.html
4. Bubble Sort: https://www.geeksforgeeks.org/dsa/bubble-sort-algorithm/
5. Сортування злиттям: алгоритм, переваги і особливості. URL: https://javarush.com/ua/groups/posts/uk.2202.sortuvannja-zlittjam-merge-sort-v-java
6. Radix Sort: https://www.ritambhara.in/radix-sort/
7. Коцовський В. М. Теорія паралельних обчислень: навчальний посібник. Ужгород: ПП «АУТДОР-Шарк», 2021. 188 с.
8. OpenMP Architecture Review Board.Доступно: https://www.openmp.org/specifications/
9. Паралельні алгоритми та їх складність. URL: https://uk.wikipedia.org/wiki/Паралельний_алгоритм
10. GeeksforGeeks. Breadth First Search (BFS) Algorithm for Graphs. Доступно: https://www.geeksforgeeks.org/breadth-first-search-bfs-for-graphs/

Загон Юлія Олексіївна – студентка кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: juliazagon@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.

Zahon Juliia Oleksiyivna – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: juliazagon@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 .