УДК 004.8

У. О. Богуцька
В.О. Денисюк

РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ СОРТУВАНННЯ CYCLE SORT

Вінницький національний технічний університет

Анотація

Досліджено реалізацію паралельного алгоритму сортування Cycle Sort з використанням мови Python та бібліотеки multiprocessing. Проведено порівняльний аналіз швидкодії послідовної та паралельної версій. Встановлено, що через високу кількість залежних операцій та витрати на синхронізацію, паралельна реалізація алгоритму в загальних випадках є малоефективною.

Ключові слова: *паралельні обчислення, Cycle Sort, Python, прискорення, синхронізація, часова складність.*

Abstract

The implementation of the parallel sorting algorithm Cycle Sort using the Python language and the multiprocessing library was investigated. A comparative analysis of the performance of the serial and parallel versions was conducted. It was found that due to the high number of dependent operations and synchronization costs, the parallel implementation of the algorithm is inefficient in general cases.

Keywords: *parallel computing, Cycle Sort, Python, acceleration, synchronization, time complexity.*

Вступ

Актуальність розробки полягає в тому, що Cycle Sort є єдиним алгоритмом сортування порівняннями, який гарантує мінімально можливу кількість операцій перезапису в пам'ять (не більше N-1 записів). У сучасних умовах розвитку апаратного забезпечення це має критичне значення для систем із обмеженим ресурсом перезапису, таких як SSD-накопичувачі та Flash-пам’ять, де кожна зайва операція запису не лише забирає час, а й фізично зношує носій [1-2].

Використання обчислювальних потужностей багатоядерних систем дає змогу поєднати унікальну апаратну ощадливість Cycle Sort із вимогами концепції Big Data щодо швидкодії, роблячи його ефективним інструментом для обробки великих масивів даних у спеціалізованих системах [3].

Удосконалення сучасних обчислювальних потужностей перетворює результативність сортувальних алгоритмів на один із визначальних чинників загальної ефективності програмних продуктів [4]. Саме тому створення подібних методів для паралельних структур набуває особливої значущості через невпинне підвищення запитів до швидкості аналізу масивних потоків інформації. Процедури впорядкування даних залишаються фундаментальними елементами, що найчастіше застосовуються в інформаційних комплексах, сховищах даних та дослідницьких розрахунках.

Новітній поступ технологій визначається впровадженням багатоядерних і децентралізованих систем, де паралелізація стає основним інструментом нарощування потужності. Це формує перед фахівцями виклик щодо виявлення та впровадження алгоритмічних рішень, здатних максимально раціонально задіяти наявний апаратний потенціал.

Постановка задачі дослідження

Задачі дослідження полягають у вирішенні наступних питань:

- розробка ефективної архітектури алгоритму Cycle Sort для оптимального розподілу обчислень;
- мінімізація затримок передачі даних між вузлами;
- розробка програмної реалізації Cycle Sort;
- тестування програми та аналіз отриманих результатів.

Виклад основного матеріалу

Алгоритм Cycle Sort використовується у специфічних ситуаціях, коли його унікальна властивість мінімізації записів стає пріоритетнішою за швидкість. Він є незамінним для впорядкування даних на носіях з обмеженим ресурсом перезапису, таких як SSD-накопичувачі або флеш-пам'ять, а також при маніпуляціях із великими складними об’єктами, де фізичне копіювання є ресурсомістким. Методи реалізації включають: стандартний послідовний алгоритм із квадратичною часовою складністю $O(N^2)$; та паралельні підходи, що використовують багатоядерні архітектури для скорочення загального часу виконання при збереженні оптимальності за кількістю операцій запису.

Для того, щоб ефективно реалізувати паралельну версію Cycle Sort, проведено дослідження технологій паралелізму, таких як OpenMP, Pthreads та модуль multiprocessing у Python. Це допомогло визначити оптимальні механізми взаємодії між обчислювальними елементами та способи керування спільними ресурсами [5].

Паралельний Cycle Sort — це система, що базується на розподілі вхідного масиву між декількома процесами або потоками. Кожен обчислювальний елемент одночасно ідентифікує коректні позиції елементів у межах своєї частини даних. Вузли (процеси) взаємодіють через спільну пам'ять, забезпечуючи переміщення елементів на їхні фінальні місця. Така структура дозволяє одночасно виявляти та обробляти різні цикли перестановок, що теоретично підвищує продуктивність системи.

Процес паралелізації передбачає логічний поділ масиву на підмасиви, що передаються окремим робочим одиницям. Під час налаштування системи всі процеси ініціалізуються синхронно [6-7]. Навіть якщо обробка різних частин масиву має складні залежності по даним, механізми блокування гарантують, що кожен елемент буде записаний у кінцеву позицію лише один раз.

На відміну від алгоритмів, що легко піддаються декомпозиції (наприклад, Merge Sort), паралельний Cycle Sort вимагає інтенсивної синхронізації через спільний масив [8]. Це мережа обчислень, де кожен потік координує свої дії з іншими для уникнення конфліктів запису. Такий підхід допомагає досягти балансу між використанням ресурсів багатоядерного процесора та збереженням цілісності даних.

Паралельні реалізації алгоритму можуть базуватися на різних архітектурних моделях:

- моделі зі спільною пам'яттю (Shared memory);
- багатопроцесні моделі (Python multiprocessing);
- моделі з використанням м'ютексів та критичних секцій;
- структури на основі потокових графів підзадач;
- системи з асинхронним керуванням через GUI;
- архітектури з чітким розділенням логіки та інтерфейсу (UML-моделі);
- розподілені моделі (хоча вони менш ефективні для даного алгоритму) [9].

Основні переваги та ідеї застосування паралельного Cycle Sort включають:

- одночасна ідентифікація позицій, що прискорює найбільш трудомістку фазу пошуку в масиві;
- завдяки розподілу навантаження на P ядер, можливо зменшити загальний час сортування великих масивів;
- структура алгоритму дозволяє масштабувати обчислення відповідно до кількості доступних ядер CPU;
- використання спільних масивів та м'ютексів забезпечує безпечну роботу в багатопоточному середовищі;
- можливість візуалізації процесу дозволяє детально аналізувати ефективність обертання циклів у реальному часі.

Розглянуто та використано основні способи оптимізації Cycle Sort:

- ***за рахунок розподілу даних*** - відний масив поділяється на діапазони індексів, що дозволяє кожному процесу незалежно обробляти свою частину;
- ***за рахунок механізмів синхронізації*** - використання об'єктів Lock запобігає конфліктам при одночасному записі елементів у спільну пам'ять різними потоками.

- ***за рахунок математичного моделювання*** - аналіз коефіцієнтів прискорення *S(P)* та ефективності *E(P)* дозволяє визначити межі доцільності паралелізації для конкретних розмірів даних;
- ***за рахунок архітектурного проектування*** - використання UML-діаграм класів та діяльності забезпечує чітку структуру програмного модуля та взаємодію компонентів.

Програмно реалізовано задану задачу за допомогою мови Python та бібліотеки Tkinter, а також описано всі компоненти коду та результати тестування [10].

Аналіз результатів тестування програми виконано для різної кількості елементів масиву (табл.1-2).

Таблиця 1 – Час виконання програми на різних вхідних даних

Кількість ядер	Кількість елементів у масиві		
	200	500	1000
12	1.53882	10.81331	54.35227

Таблиця 2 – Коефіцієнти прискорення

Кількість елементів у масиві	200	500	1000
Коефіцієнт прискорення	0.002	0.001	0.001

Проаналізувавши обчислені коефіцієнти прискорення та коефіцієнти ефективності при різних типах вхідних даних та використанні багатоядерної архітектури, можливо зробити такі висновки:

- при обробці масивів розміром 1000 елементів показники прискорення S(P) та ефективності E(P) є критично низькими (близькими до нуля);
- паралельна реалізація демонструє значне уповільнення порівняно з послідовною версією, наприклад, для випадкового масиву час виконання зріс із 0,09 с до 77,4 с;
- залучення великої кількості ядер процесора (в даному випадку 12) не дає позитивного ефекту, оскільки накладні витрати на синхронізацію процесів та блокування м'ютексів повністю нівелюють виграш від паралелізму;
- тип вхідних даних (випадковий, відсортований чи зворотний) суттєво впливає на час виконання, проте в усіх випадках паралельна версія залишається менш продуктивною через специфіку алгоритму.

Отже, основними чинниками, які впливають на зміну розглянутих показників, таких як коефіцієнт прискорення, коефіцієнт ефективності та загальний час виконання, є розмір вхідних даних, кількість задіяних обчислювальних процесів та, що найважливіше, внутрішня природа алгоритму Cycle Sort. Висока часова складність $O(N^2)$ та значна кількість залежних операцій запису роблять цей метод малоефективним для паралелізації, оскільки витрати на управління доступом до спільної пам'яті домінують над процесом обчислень.

Висновки

У процесі виконання роботи було реалізовано та досліджено алгоритм сортування Cycle Sort у послідовному та паралельному варіантах. Було встановлено, що послідовна реалізація алгоритму працює стабільно та ефективно для масивів різного розміру, тоді як паралельна версія не демонструє очікуваного приросту продуктивності.

Експериментальні результати показали, що при використанні паралельних обчислень час виконання алгоритму суттєво збільшується порівняно з послідовною версією. Це пов’язано з особливостями алгоритму Cycle Sort, який має значну кількість залежних операцій, потребує частих звернень до спільної пам’яті та використання механізмів синхронізації, що створює додаткові накладні витрати.

Таким чином, у рамках дослідження було зроблено висновок, що Cycle Sort є малоефективним

для паралельної реалізації, а його використання доцільне переважно в послідовному режимі. Отримані результати є важливими для розуміння обмежень паралелізації алгоритмів та можуть бути використані при виборі оптимальних методів сортування для практичних задач.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Parallel Sorting – University of Porto, Faculty of Computer Science. URL: https://www.dcc.fc.up.pt/~ricroc/aulas/1516/cp/apontamentos/slides_sorting.pdf
2. Singh R., Sharma P. Parallel Sorting Algorithms: Performance Evaluation and Comparative Analysis. Electronic edition. IEEE Xplore, 2022
3. Дорошенко А. Ю., Андон П. І., Яценко О. А., Жереб К. А. Алгебро-алгоритмічні моделі та методи паралельного програмування. Київ: Видавничий дім «Академперіодика», 2018. 192 с
4. Cycle Sort Algorithm. URL: https://www.baeldung.com/cs/cycle-sort-algorithm
5. Parallelization of Cycle Sort Algorithm: Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal-576, 2024
6. Як будувати UML-діаграми? URL: https://dou.ua/forums/topic/40575/
7. Об’єктно-орієнтоване програмування: [Підручник] / В.В. Бублик. К.: ІТкнига, 2015. 624 с.
8. Python Tkinter. URL: https://www.geeksforgeeks.org/python/python-gui-tkinter
9. Downey A. Think Python: How to Think Like a Computer Scientist. Green Tea Press, 2015. 300 с.
10. TutorialsPoint. Python GUI Programming with Tkinter, 2024. URL: https://www.tutorialspoint.com/python/python_gui_programming.htm

Богуцька Уляна Олександрівна – студент кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: ul.bogytska@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.

Bogutska Uliana Oleksandrivna – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: ul.bogytska@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 .