УДК 004.8

А.А.Андрухова
В.О. Денисюк

ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ СОРТУВАННЯ HYPER QUICK SORT

Вінницький національний технічний університет

Анотація

Розглянуто розробку паралельного алгоритму сортування Hyper Quick Sort за допомогою технології MPI. Проаналізовано існуючі методи паралельного сортування з метою досягнення високої продуктивності, обґрунтовано вибір засобів розробки програмного модуля, розроблено графи, що відображають архітектуру та логіку роботи алгоритму, а також обґрунтовано вибір програмного середовища реалізації. У роботі створено програмну реалізацію оптимізованого паралельного алгоритму сортування з використанням MPI та проведено тестування його продуктивності на масивах різного обсягу. Використання отриманих результатів дозволить підвищити швидкодію та ефективність обробки великих обсягів даних у системах, що вимагають високопродуктивного паралельного сортування.

Ключові слова: *паралельний алгоритм, паралельне сортування, гіпершвидке сортування, MPI.*

Abstract

The development of the parallel sorting algorithm Hyper Quick Sort using MPI technology is considered. Existing methods of parallel sorting are analyzed in order to achieve high performance, the choice of software module development tools is justified, graphs are developed that reflect the architecture and logic of the algorithm, and the choice of software implementation environment is justified. The paper creates a software implementation of the optimized parallel sorting algorithm using MPI and tests its performance on arrays of various sizes. Using the results obtained will allow to increase the speed and efficiency of processing large amounts of data in systems that require high-performance parallel sorting.

Keywords: *parallel algorithm, parallel sorting, Hyper Quick Sort, MPI.*

Вступ

Актуальність реалізації паралельного алгоритму сортування Hyper Quick Sort полягає в тому, що сучасні задачі обробки великих обсягів даних — зокрема в галузях машинного навчання, наукових обчислень, бізнес- аналітики та інженерного моделювання — вимагають високопродуктивних методів упорядкування інформації. Особливістю алгоритму Hyper Quick Sort є його ефективна адаптація до паралельних обчислювальних середовищ, що дозволяє суттєво прискорити процес сортування за рахунок одночасної обробки різних частин масиву на окремих вузлах системи. Використання технології MPI забезпечує ефективний розподіл даних між процесами та координацію їх взаємодії, що є критично важливим для досягнення масштабованості й високої продуктивності.

Паралельні архітектури дозволяють розподіляти обчислювальне навантаження між кількома процесорами або вузлами мережі, а результати їх роботи об’єднуються в єдиний відсортований масив. Такий підхід є особливо ефективним при роботі з великими обсягами даних, де класичні послідовні алгоритми сортування стають обчислювально надто витратними.

Метою роботи є розробка паралельного алгоритму сортування Hyper Quick Sort з використанням MPI для покращення швидкодії та продуктивності обробки великих масивів даних.

Постановка задачі дослідження

Задачі дослідження полягають у вирішенні наступних питань:

- проаналізувати відомі алгоритми сортування та визначити переваги Hyper Quick Sort у контексті паралельних обчислень;

- розробити алгоритмічну модель паралельного Hyper Quick Sort та обґрунтувати вибір технології MPI;
- виконати тестування та провести аналіз отриманих результатів щодо швидкодії та масштабованості.

Виклад основного матеріалу

Паралельне сортування широко застосовується у задачах високопродуктивних обчислень, де необхідно швидко обробляти великі обсяги даних — зокрема в аналізі наукових даних, фінансовому моделюванні, обробці логів систем, а також у підготовці даних для машинного навчання [1]. Серед існуючих методів сортування виділяють класичні алгоритми, такі як сортування злиттям (Merge Sort) та швидке сортування (Quick Sort), які мають середню обчислювальну складність O(nlogn) у послідовному варіанті [2-5]. Однак для досягнення високої продуктивності в умовах розподілених систем використовуються спеціалізовані паралельні алгоритми, зокрема Hyper Quick Sort, який адаптований до архітектури гіперкуба та забезпечує ефективну роботу в середовищах з розподіленою пам'яттю [6].

Для реалізації паралельного сортування було проведено дослідження методів розподілених обчислень, зокрема технології MPI (Message Passing Interface), що дозволяє організувати взаємодію між процесами через передачу повідомлень у системах із розподіленою пам'яттю. Це стало основою для розробки програмної реалізації алгоритму Hyper Quick Sort [1, 6].

Hyper Quick Sort — це паралельна модифікація класичного Quick Sort, яка виконується на логічній топології гіперкуба. Алгоритм передбачає розподіл вхідного масиву між процесами, локальне сортування кожної частини, вибір глобального опорного елемента (pivot), поділ даних на «нижні» та «верхні» підмасиви, а також обмін відповідними частинами між парами процесів-партнерів у кожному вимірі гіперкуба. У результаті кожен процес отримує діапазон значень, який не перетинається з діапазонами інших процесів, що дозволяє завершити сортування локально, не вдаючись до подальшого обміну даними [6, 7].

На відміну від інших паралельних алгоритмів сортування, Hyper Quick Sort забезпечує мінімальний комунікаційний трафік завдяки організованому обміну лише необхідними частинами даних, а також рівномірне завантаження процесів, що є критичним для масштабованості на великих кластерах.

П'ять основних переваг та ідей застосування Hyper Quick Sort у поєднанні з MPI такі.

1. Ефективний розподіл обчислень: масив поділяється між процесами, що дозволяє кожному з них працювати незалежно та паралельно обробляти власну частину даних.
2. Мінімізація комунікаційних накладних витрат: обмін відбувається лише між парами процесів у кожному вимірі гіперкуба, що зменшує кількість передач і синхронізацій.
3. Логічна топологія гіперкуба: забезпечує чітку структуру взаємодії процесів, що спрощує реалізацію та прогнозується продуктивністю.
4. Масштабованість: алгоритм добре масштабується зі збільшенням кількості процесів, особливо при обробці великих масивів, де обчислювальне навантаження переважає над затратами на передачу даних.
5. Гарантована коректність: після завершення обмінів всі елементи на процесах з нижчим рангом завжди менші за елементи на процесах з вищим рангом, що забезпечує глобальну впорядкованість після локального сортування.

У дослідженнях розглянуто та використано чотири основних підходи до оптимізації паралельного алгоритму Hyper Quick Sort.

1. За рахунок паралелізації обчислень. Вхідний масив рівномірно розподіляється між процесами за допомогою колективної операції MPI_Scatter. Кожен процес виконує локальне сортування та подальші етапи розбиття незалежно, що максимально використовує паралелізм обчислень.
2. За рахунок архітектури логічного гіперкуба. Логічна топологія процесів моделюється шляхом поділу комунікатора (MPI_Comm_split) на підгрупи, що відповідають

підгіперкубам. Це дозволяє організувати обмін даними лише між партнерами у кожному вимірі, що зменшує час передачі та підвищує ефективність.

3. За рахунок використання оптимізованих процедур сортування та розбиття. Реалізовано функції hyper_partition та quicksort, які забезпечують ефективне розбиття локальних підмасивів навколо глобального опорного елемента, а також локальне завершення сортування з мінімальними затратами.
4. За рахунок взаємодії між процесами через MPI. Обмін даними здійснюється за допомогою точкових операцій (MPI_Send/MPI_Recv) та колективних операцій (MPI_Bcast, MPI_Gatherv). Синхронізація процесів досягається неявно через послідовність комунікацій, що забезпечує коректність та уникнення гонок даних.

Програмно реалізовано задану задачу та описано всі компоненти коду [8-9].

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

Таблиця 1 – Час виконання програми на різних вхідних даних

Кількість процесів	Час для16 елементів, с	Час для 1 000 000 елементів, с
2	2.13×10^{-5}	0.842
4	3.75×10^{-5}	0.467
8	59.80×10^{-5}	0.291

Таблиця 2 – Коефіцієнти прискорення

Розмір масиву	16 елементів	1 000 000 елементів
Коефіцієнт прискорення (S)	0.36	2.89

Таблиця 3 – Коефіцієнти ефективності

Кількість процесів	2	4	8
Коефіцієнт ефективності (E), 16 елементів	0.18	0.09	0.0075
Коефіцієнт ефективності (E), 1 000 000 елементів	0.72	0.36	0.1800

Проаналізувавши отримані значення часу виконання, коефіцієнти прискорення та ефективності при різних обсягах вхідних даних і різній кількості процесів (табл. 1–3), можна зробити такі висновки.

1. При невеликих вхідних даних (наприклад, 16 елементів) значення коефіцієнтів прискорення та ефективності є значно нижчими за 1, а зі збільшенням кількості процесів час виконання навіть зростає. Це свідчить про те, що накладні витрати на комунікацію між процесами (ініціалізація MPI, обмін повідомленнями, синхронізація) перевищують виграш від паралельного обчислення. Тому застосування багатопроцесного підходу для малих масивів є неефективним.
2. Навпаки, при великих вхідних даних (1 000 000 елементів) спостерігається значне зменшення часу виконання зі збільшенням кількості процесів. Коефіцієнт прискорення становить 2.89 при використанні 8 процесів порівняно з 2 процесами, що підтверджує ефективну паралелізацію. Хоча коефіцієнт ефективності зменшується зі зростанням кількості процесів (з 0.72 до 0.18), це є типовим для багатопроцесних систем через зростання комунікаційних витрат.

Отже, основними чинниками, які впливають на зміну таких показників, як час виконання, коефіцієнт прискорення та коефіцієнт ефективності, є:

- розмір вхідного масиву: лише при достатньо великому обсязі даних переваги паралельних обчислень переважають комунікаційні накладні витрати;

- кількість процесів - оптимальна кількість процесів залежить від обсягу даних, для малих масивів краще використовувати мінімальну кількість процесів (наприклад, 2), а для великих збільшувати їх кількість;
- організація обміну даними в Hyper Quick Sort - алгоритм передбачає регулярний обмін між процесами-партнерами, що при малих даних стає «вузьким місцем», але при великих добре масштабується.

Таким чином, паралельний алгоритм Hyper Quick Sort є доцільним лише для обробки великих масивів, де обчислювальне навантаження значно перевищує витрати на взаємодію процесів, що підтверджується експериментальними даними.

Висновки

Досліджено паралельне сортування як одну з ключових операцій у високопродуктивних обчисленнях та визначено основні сфери його застосування. Розглянуто базові алгоритми сортування й проаналізовано принципи їх паралельної реалізації, зокрема алгоритм Hyper Quick Sort, описано його структуру, особливості роботи та сфери використання. На основі літературних джерел досліджено технологію MPI як складову розподілених систем та сформульовано підхід до її застосування для паралельних обчислень.

Програмно реалізовано задану задачу та описано основні компоненти коду, наведено UML-діаграми. Проведено тестування розробленої програми, проаналізовано результати та досліджено коефіцієнти прискорення й ефективності.

Встановлено, що збільшення кількості процесів забезпечує прискорення лише для великих обсягів даних, тоді як для малих масивів ефективність знижується через накладні витрати на обмін повідомленнями.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Жуков І.А. Паралельні та розподілені обчислення. Лабораторнийпрактикум / І.А. Жуков, О.В. Корочкін. К. : Корнейчук, 2008. 224 с. URL: https://comsys.kpi.ua/wp-content/uploads/2025/02/paralelkorochkin.pdf
2. Бульбашкове сортування URL: https://campus.epam.ua/ua/blog/434
3. Сортуваня вставкою URL: https://www.guru99.com/uk/insertion-sort-algorithm.html
4. Merge Sort URL: https://www.geeksforgeeks.org/dsa/merge-sort/
5. Quick Sort URL: https://phm.cuspu.edu.ua/nauka/naukovo-populiarni-publikatsii/824-quicksort-istoriia-vynyknennia-ta-rozvytku-naishvydshoho-alhorytmu-sortuvannia.html
6. Hyper Quick Sort URL: https://journals.khnu.km.ua/vestnik/wp-content/uploads/2023/09/323-95-105.pdf
7. Hyper Quick Sort URL: https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Mrunal-Narendra-Inge-Spring-2021.pdf
8. Prinz, Peter. [C++ Lernen und professionell anwenden. English] A complete guide to programming in C++ / Peter Prinz, Ulla Kirch-Prinz; translated by Ian Travis. p. cm. URL: https://www.idpoisson.fr/volkov/C++.pdf
9. C++ URL: https://cplusplus.com/doc/tutorial/#google_vignette

Андрухова Анастасія Анатоліївна – студентка кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: anastasiiaandrukhova8@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.

Andrukhova Anastasiia Anatoliivna – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: anastasiiaandrukhova8@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