УДК 004.8

Д.А. Барабаш
В.О. Денисюк

ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ ПОШУКУ DIVIDE AND CONQUER

Вінницький національний технічний університет

Анотація

У роботі розглянуто розробку та дослідження паралельного k-арного алгоритму пошуку, який є адаптацією класичного методу бінарного пошуку для багатоядерних архітектур на основі стратегії «Розділяй і володарюй» (Divide and Conquer). Проаналізовано проблему «стіни пам'яті» (Memory Wall) та обмеження продуктивності послідовних алгоритмів. Розроблено програмну реалізацію алгоритму мовою C++ з використанням технології OpenMP, побудовано математичну модель, що враховує накладні витрати на синхронізацію. Проведено експериментальне тестування на масивах до 100 млн елементів. Результати показали, що хоча паралельний алгоритм значно випереджає лінійний пошук, на примітивних типах даних його ефективність обмежується системним оверхедом порівняно з бінарним пошуком

Ключові слова: *паралельний пошук, k-арний пошук, OpenMP, Divide and Conquer, багатопотоковість, швидкодія.*

Abstract

The paper addresses the development and research of a parallel k-ary search algorithm, which is an adaptation of the classical binary search method for multi-core architectures based on the "Divide and Conquer" strategy. The "Memory Wall" problem and performance limitations of sequential algorithms are analyzed. A software implementation of the algorithm was developed in C++ using OpenMP technology, and a mathematical model accounting for synchronization overhead was constructed. Experimental testing was conducted on arrays up to 100 million elements. The results demonstrated that while the parallel algorithm significantly outperforms linear search, its efficiency on primitive data types is limited by system overhead compared to binary search.

Keywords: *parallel search, k-ary search, OpenMP, Divide and Conquer, multithreading, performance.*

Вступ

Ефективний пошук даних є критично важливим завданням у сучасних інформаційних системах, що оперують великими обсягами інформації. Класичний бінарний пошук є еталоном для послідовних систем, проте він не використовує потенціал сучасних багатоядерних процесорів. Підхід Divide and Conquer дозволяє узагальнити бінарний пошук до k-арного, де простір пошуку ділиться на k частин одночасно. Використання технології OpenMP дозволяє реалізувати цей паралелізм у моделі спільної пам'яті. Актуальність дослідження полягає у визначенні меж ефективності такого підходу та аналізі впливу накладних витрат (overhead) на реальне прискорення алгоритму.

Постановка задачі дослідження

Основні задачі роботи полягали у розв'язанні таких питань:

- аналіз обмежень продуктивності класичних алгоритмів пошуку (лінійного та бінарного);
- математичне моделювання k-арного пошуку та оцінка теоретичного прискорення $S = log_2(P + 1)$;
- програмна реалізація паралельного алгоритму мовою C++ із використанням директив OpenMP;
- проведення експериментального дослідження на різних сценаріях (пошук на початку, в середині та в кінці масиву);
- аналіз отриманих метрик Speedup та визначення причин неефективності на простих типах даних.

Виклад основного матеріалу

Алгоритми пошуку є фундаментальними у комп'ютерних науках. Класичний бінарний пошук має часову складність $O(log_2 N)$, що забезпечує високу швидкодію, однак створює жорстку залежність за даними, яка унеможливлює його пряме розпаралелення [1, 2].

Було досліджено проблему «стіни пам'яті» (Memory Wall) та обмеження тактової частоти сучасних процесорів, що спонукає до пошуку алгоритмічних рішень, орієнтованих на багатоядерні архітектури. Окрему увагу приділено стратегії «Розділяй і володарюй» (Divide and Conquer), яка допускає узагальнення бінарного поділу до множинного (k-арного) розбиття простору пошуку [3, 4]. У роботі запропоновано використання k- арного пошуку, де коефіцієнт розгалуження k залежить від кількості доступних потоків P, k = P+1. Це дозволяє зменшити висоту дерева пошуку до $O(log_k N)$, що теоретично скорочує кількість звернень до оперативної пам'яті та мінімізує вплив латентності (Memory Wall) [5-9].

Програмну реалізацію виконано мовою C++ у середовищі Visual Studio [10-12]. Для організації паралелізму використано бібліотеку OpenMP. Для організації паралелізму використано директиву #pragma omp parallel, яка дозволяє потокам одночасно обчислювати індекси роздільників (pivots) та порівнювати їх з шуканим ключем. Синхронізація результатів реалізована через критичні секції (#pragma omp critical).

Експериментальне дослідження проводилося на масивах розмірністю 10^6, 10^7, 10^8 елементів. Було розглянуто три сценарії (табл.1-3).

Таблиця 1 – Порівняння часу виконання (с) та прискорення (шуканий індекс = 0)

Розмірність масиву (N)	Лінійний	Бінарний	2 потоки	4 потоки	8 потоків	16 потоків
1 000 000	0,0002	0,0012	8,2055	4,0475	4,6610	9,6278
10 000 000	0,0001	0,0050	5,9228	1,8947	3,9621	8,5784
50 000 000	0,0002	0,0027	7,5562	2,3318	4,1221	7,9735

Таблиця 2 – Порівняння часу виконання (с) та прискорення (шуканий індекс = N/2)

Розмірність масиву (N)	Лінійний	Бінарний	2 потоки	4 потоки	8 потоків	16 потоків
1 000 000	4,3412	0,0016	6,7019	2,3672	3,7863	7,9578
10 000 000	4,3412	0,0029	6,4374	2,4985	3,9345	7,4009
50 000 000	209,6641	0,0036	7,6828	2,4214	4,3354	10,5974

Таблиця 3 – Порівняння часу виконання (с) та прискорення (шуканий індекс = N-1)

Розмірність масиву (N)	Лінійний	Бінарний	2 потоки	4 потоки	8 потоків	16 потоків
1 000 000	13,5778	0,0029	8,4249	3,8148	4,8657	7,5119
10 000 000	89,1856	0,0062	14,8542	3,5397	5,3081	9,5809
50 000 000	436,6853	0,0027	7,9190	2,5437	4,1517	8,8386

1. ***Пошук на початку масиву*** (шуканий індекс = 0): Лінійний пошук демонструє миттєвий результат (O(1)), оскільки знаходить елемент на першій ітерації. Паралельний алгоритм показує значне уповільнення (час у мілісекундах проти мікросекунд). Це пояснюється фіксованими накладними витратами на створення потоків (Fork overhead), які значно перевищують час корисної роботи.

2. ***Пошук в середині масиву*** (шуканий індекс = N/2): Паралельний алгоритм (на 4-х потоках) вже значно випереджає лінійний. Однак він все ще поступається бінарному пошуку, який знаходить середину миттєво. Найкращий час серед паралельних конфігурацій стабільно показують 4 потоки.

3. ***Пошук в кінці масиву*** (шуканий індекс = N-1): Лінійний пошук стає критично повільним. Паралельний алгоритм (4 потоки) виконується швидше за лінійний. Це доводить ефективність стратегії "Розділяй і володарюй". Проте, порівняно з бінарним пошуком паралельний метод все ще повільніший.

Експериментально визначено, що прискорення з використанням розпаралелення алгоритму k-арного пошуку відносно бінарного та лінійного пошуку для 2, 4, 8 та 16 потоків становить “0”. Аналіз результатів підтвердив, що для примітивних типів даних (int) час виконання корисної роботи (порівняння) є на порядки меншим за час синхронізації потоків OpenMP ($t_{comp} \ll t_{sync}$), що призводить до відсутності прискорення. Оптимальною кількістю потоків для даної реалізації визначено 4 потоки, подальше збільшення призводить до деградації продуктивності. Це зумовлено зростанням накладних витрат на комунікацію між ядрами та конкуренцією за доступ до спільної пам'яті.

Висновки

Досліджено та реалізовано паралельний k-арний алгоритм пошуку. Встановлено, що даний метод є ефективною заміною лінійному пошуку на великих масивах даних. Однак, у порівнянні з класичним бінарним пошуком, його ефективність на простих типах даних обмежена накладними витратами на управління потоками.

Отримані результати свідчать, що застосування паралельного пошуку є доцільним для задач із високою обчислювальною складністю функції порівняння (наприклад, складні об'єкти або рядки) або в умовах, де доступ до пам'яті є критичним вузьким місцем. Робота підтверджує закон Амдала та демонструє важливість балансу між гранулярністю задач та накладними витратами паралелізму.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Leskovec J., Rajaraman A., Ullman J. D. Mining of Massive Datasets. 3rd ed. Cambridge : Cambridge University Press, 2020. 565 p.
2. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge : MIT Press, 2022. 1312 p.
3. Hager G., Wellein G. Introduction to High Performance Computing for Scientists and Engineers. Boca Raton : CRC Press, 2010. 356 p.
4. Марченко О. І., Марченко О. О. Структури даних та алгоритми : підручник. Частина 1. Київ : КПІ ім. Ігоря Сікорського, 2024. 268 с.
5. Hennessy J. L., Patterson D. A. Computer Architecture: A Quantitative Approach. 6th ed. San Francisco : Morgan Kaufmann, 2019. 936 p.
6. Мельник А. О. Архітектура комп’ютера : підручник. Луцьк : Волинська обласна друкарня, 2008. 470 с.
7. Креневич А. П. Алгоритми і структури даних : підручник. Київ : ВПЦ "Київський університет", 2021. 200 с.
8. Grama A., Gupta A., Karypis G., Kumar V. Introduction to Parallel Computing. 2nd ed. Harlow: Pearson Education,2003. 856.
9. Knuth D. E. The Art of Computer Programming. Vol. 3: Sorting and Searching. 2nd ed. Reading, Massachusetts : Addison-Wesley, 1998. 780 p.
10. The OpenMP API specification for parallel programming. URL: http://www.openmp.org.
11. Шпак З. Я. Програмування мовою С++: підручник. Львів: Видавництво Львівської політехніки, 2011. 436 с.
12. Технічна документація Microsoft. URL: https://docs.microsoft.com/uk-ua/.

Барабаш Діана Андріївна – студентка кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: 1412sutb@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.

Barabash Diana Andriyivna – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: 1412sutb@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.