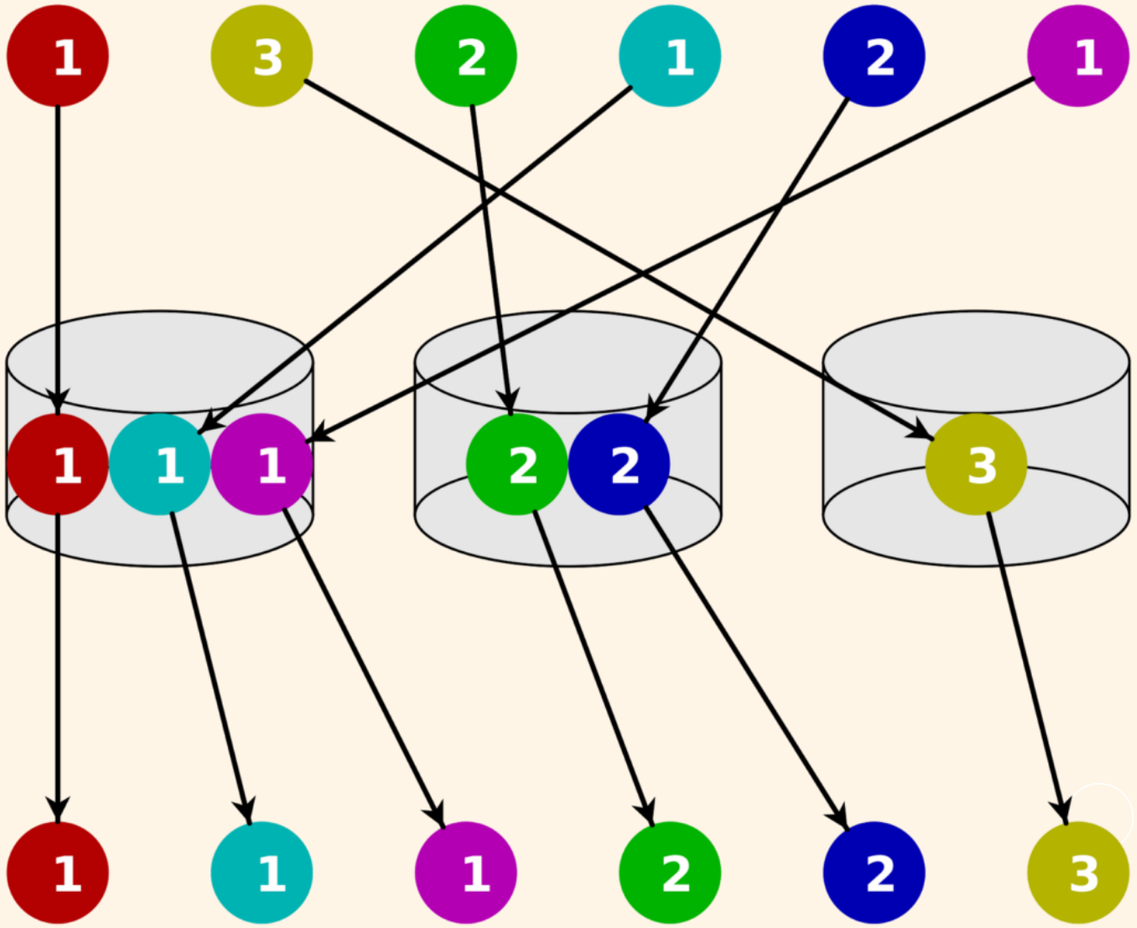


Паралельна реалізація алгоритму bucket sort мовою C++ з використанням OpenMP

Виконала: Скрипець Ольга, ІП-21

Керівник: Нестеренко К.П.

Київ, 2024

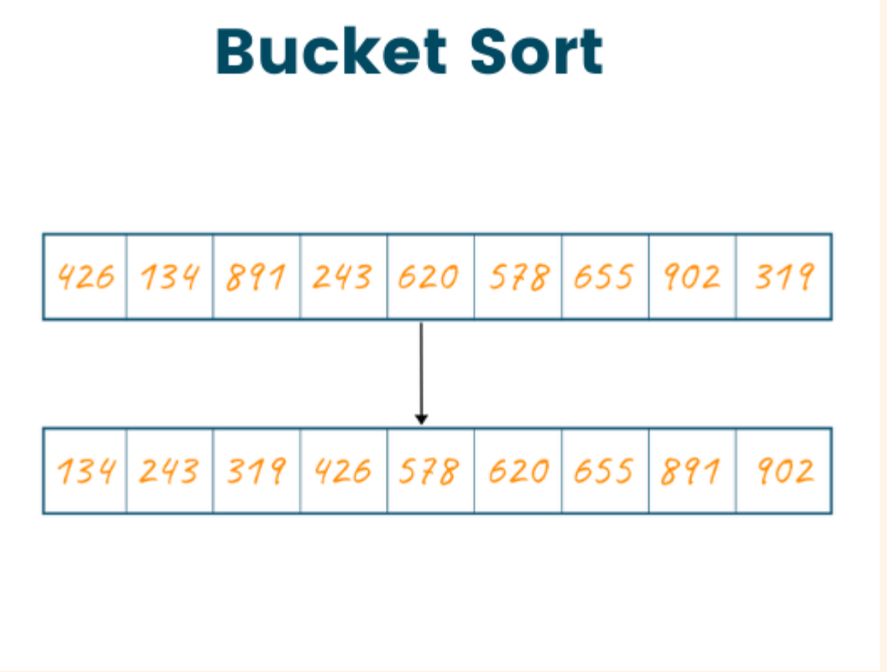
Актуальність теми

Велика кількість даних

Велика кількість даних потребує ефективної обробки, з чим послідовні алгоритми не завжди справляються.

Багатоядерні процесори

Багатоядерні процесори дозволяють використовувати паралельні обчислення для прискорення обробки даних.



Алгоритм bucket sort

Алгоритм bucket sort підходить для розпаралелювання, що робить його актуальним для оптимізації.

Мета та завдання курсової

Мета

Розробити паралельну версію алгоритму bucket sort мовою C++

Завдання 1

Провести огляд теорії та існуючих реалізацій алгоритму bucket sort.

Завдання 2

Розробити послідовний та паралельний алгоритми bucket sort.

Завдання 3

Провести тестування та порівняльний аналіз розроблених алгоритмів.

Суть алгоритму bucket sort

Розподіл

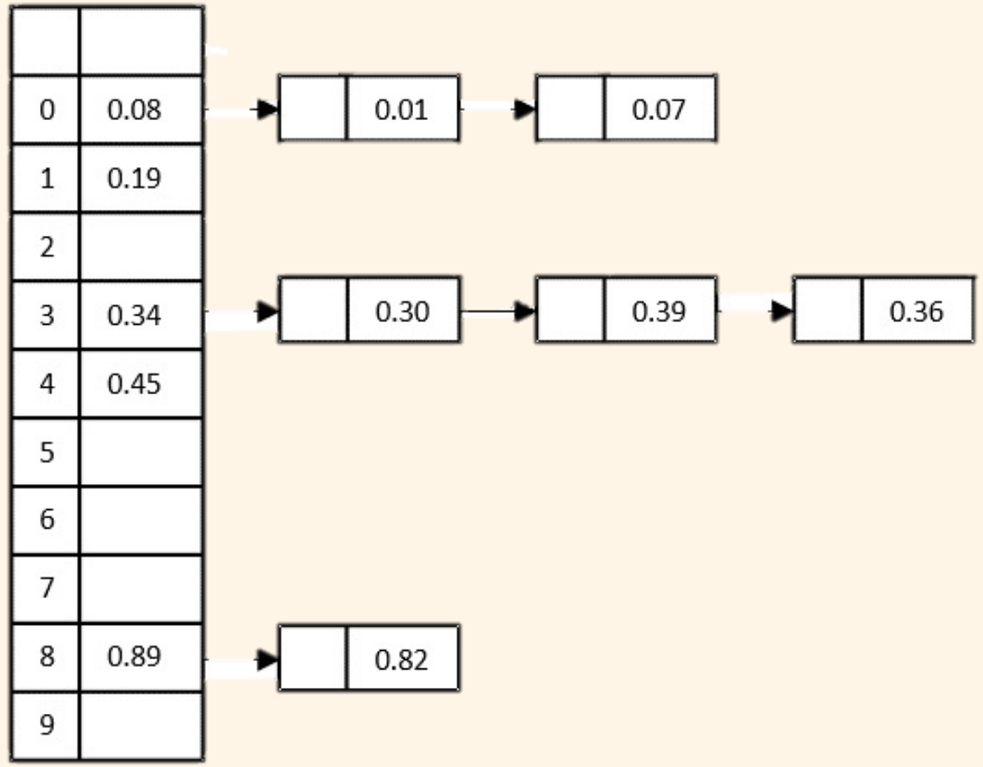
Розподіл елементів по корзинах відповідно до їх діапазону.

Сортування

Сортування елементів у кожній окремій корзині.

Об'єднання

Об'єднання відсортованих корзин в один відсортований масив.



Складність алгоритму: O(n + k). Підходить для рівномірно розподілених числових даних.

Послідовна реалізація

Особливості

- Написана мовою C++.
- Використано insertion sort для сортування корзин.
- Проведено тестування на 50 млн елементів.

Результат

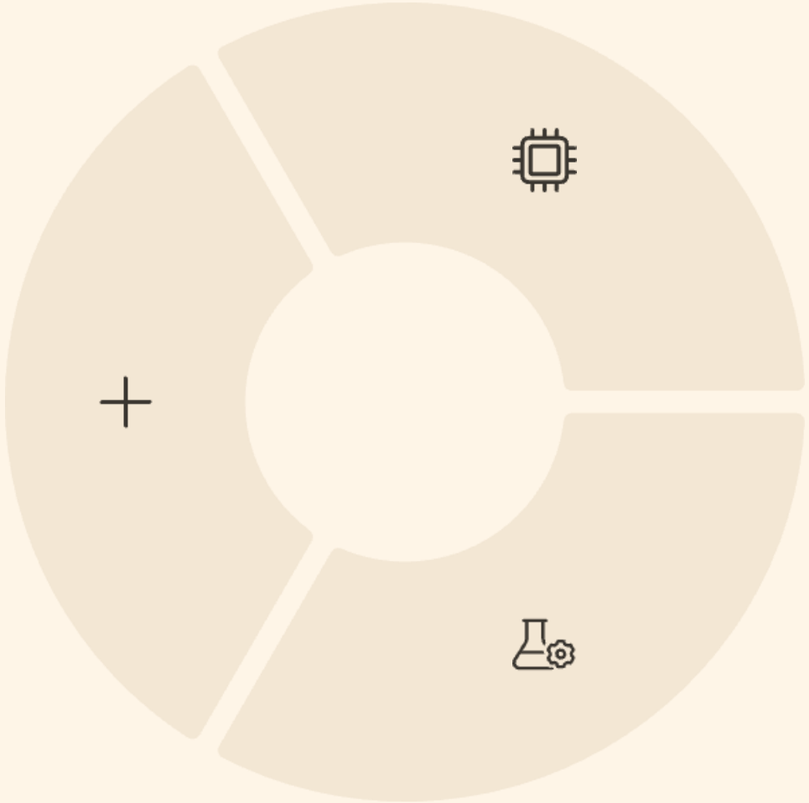
Середній час виконання: 6,45 сек.

Послідовна версія є ефективною лише для малих обсягів даних.

```
PS C:\Users\olya\Desktop\cursova> ./bucket_sort
Sorting 50000000 elements took 7.67947 seconds.
Array successfully sorted.
```

Вибір технології OpenMP

Просте впровадження у C++ проекти.



Підтримка багатоядерних процесорів для паралельних обчислень.

Гнучке налаштування кількості потоків для оптимізації продуктивності.

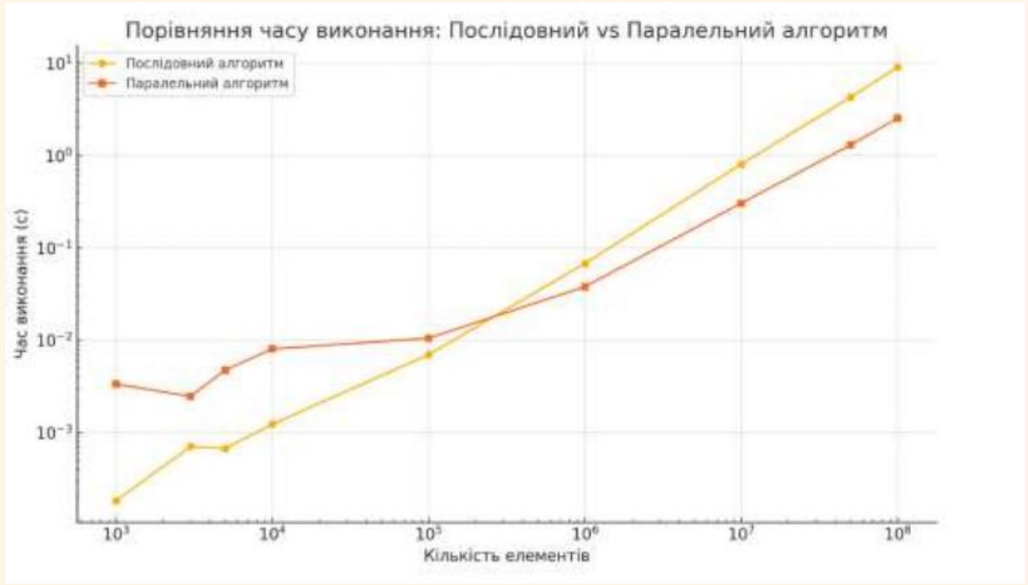
Директиви #pragma omp забезпечують простоту підключення та використання OpenMP.

Паралельна реалізація

1	Паралельне сортування елементів у кожній корзині.
2	Розподіл елементів по корзинах (через синхронізацію).
3	Об'єднання відсортованих корзин в один масив.

Середній час виконання: 2,35 сек. Прискорення у 2,74 рази порівняно з послідовною версією.

Дослідження ефективності



1K-100M

Обсяг даних

Тестування на масивах від 1 тис. до 100 млн елементів.

2.74x

Прискорення

Прискорення у 2,74 рази порівняно з послідовною версією.

3.6x

Максимальне прискорення

Максимальне прискорення: 3,6 рази (на 100 млн елементів).

При малих обсягах даних паралельна версія менш ефективна. При обсягах > 1 млн елементів — суттєве прискорення.

Кількість елементів	Час послідовного алгоритму, секунд	Час паралельного алгоритму, секунд
1000	0.0001843	0.0033672
3000	0.0007100	0.0024788
5000	0.0006806	0.0047882
10000	0.0012390	0.0080901
100000	0.0070218	0.0105631
1000000	0.0679281	0.0381365
10000000	0.8099790	0.3056880
50000000	4.2837900	1.3038600
100000000	9.0674800	2.5379000

Висновки

Розробка

Розроблено послідовну та паралельну версії алгоритму bucket sort.

Продуктивність

Проведено тестування та порівняння продуктивності алгоритмів.

Ефективність

Доведено ефективність паралельного підходу для великих обсягів даних.

Суттєве зниження часу виконання при великих обсягах даних. Простота інтеграції OpenMP у C++. Паралельні обчислення — ефективний інструмент для обробки великих даних.