

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту

«Технології паралельних обчислень. Курсова робота»

Тема: Алгоритм сортування (bucket sort) та його паралельна реалізація на Node.js

|  |  |
| --- | --- |
| **Керівник**:  ст. викладач  Дифучин Антон Юрійович  «Допущено до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 р.  Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Виконавець**:  Петров Ігор Ярославович  студент групи ІП-13  залікова книжка № ІП-1327  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «26» травня 2024 р.  Інна СТЕЦЕНКО  Антон ДИФУЧИН |

**Київ – 2024**

**ЗАВДАННЯ**

**АНОТАЦІЯ**

ЗМІСТ

[**ВСТУП** 5](#_Toc158713962)

[**1 ОПИС АЛГОРИТМУ ТА ЙОГО ВІДОМИХ ПАРАЛЕЛЬНИХ РЕАЛІЗАЦІЙ** 6](#_Toc158713963)

[**1.1 Назва підрозділу** 6](#_Toc158713964)

[**1.2 Назва підрозділу** 6](#_Toc158713965)

[**2 РОЗРОБКА ПОСЛІДОВНОГО АЛГОРИТМУ ТА АНАЛІЗ ЙОГО ШВИДКОДІЇ** 7](#_Toc158713966)

[**3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ ТА ЙОГО КОРОТКИЙ ОПИС** 8](#_Toc158713967)

[**4 РОЗРОБКА ПАРАЛЕЛЬНОГО АЛГОРИТМУ З ВИКОРИСТАННЯМ ОБРАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ПРОЕКТУВАННЯ, РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ** 9](#_Toc158713968)

[**5 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ АЛГОРИТМУ** 10](#_Toc158713969)

[**ВИСНОВКИ** 11](#_Toc158713970)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 12](#_Toc158713971)

[**ДОДАТКИ** 13](#_Toc158713972)

[**Додаток А. Нзава додатку** 13](#_Toc158713973)

[**Додаток Б. Назва додатку** 14](#_Toc158713974)

[**Додаток В. Назва додатку** 15](#_Toc158713975)

# ВСТУП

У сучасному світі, де обробка великих обсягів даних стає все більш актуальною, ефективні алгоритми сортування набувають великого значення. Одним із таких алгоритмів є "алгоритм сортування корзинами" або "bucket sort". Цей алгоритм відноситься до категорії "розподільних" сортувань, які використовуються для сортування даних, розподілення їх у певні групи (корзини) та подальшого сортування кожної групи окремо.

Однак, існують випадки, коли потрібно обробляти великі обсяги даних на багатоядерних або розподілених системах. У таких випадках паралельна реалізація алгоритмів стає ключовою для забезпечення швидкодії та ефективності обробки даних.

Ця курсова робота спрямована на дослідження алгоритму сортування корзинами та розробку його паралельної реалізації на платформі Node.js. Реалізація на Node.js може бути нетривіальною задачею через однопоточність самої мови, однак для покращення ефективності використання ресурсів системи можна використовувати мультипроцесинг.

В цілому, мета цієї роботи полягає в дослідженні ефективності паралельної реалізації алгоритму сортування корзинами на платформі Node.js порівняно з його послідовною реалізацією.

# 1 ОПИС АЛГОРИТМУ ТА ЙОГО ВІДОМИХ ПАРАЛЕЛЬНИХ РЕАЛІЗАЦІЙ

## 1.1 Опис основного алгоритму сортування

Алгоритм сортування корзинами (Bucket Sort) - це алгоритм сортування, який використовує ідею розподілу елементів масиву у "корзини" на основі їх значень, а потім сортує кожну корзину окремо, використовуючи інший алгоритм сортування.

Основні кроки алгоритму сортування корзинами:

* Розподіл у відповідні корзини: Спочатку створюються певна кількість корзин, а потім кожен елемент вихідного масиву розподіляється до відповідної корзини згідно зі значенням.
* Сортування кожної корзини: Після розподілу всі елементи, що потрапили в одну корзину, сортуються за допомогою будь-якого іншого алгоритму.
* Об'єднання корзин: Нарешті, всі відсортовані елементи кожної корзини об'єднуються в один великий відсортований масив.

Псевдокод алгоритму:

function bucketSort(array, num\_buckets):

minValue = array[0]

maxValue = array[0]

for i from 1 to length(array) - 1:

if array[i] < minValue:

minValue = array[i]

else if array[i] > maxValue:

maxValue = array[i]

interval = (maxValue - minValue + 1) / num\_buckets

buckets = Array(num\_buckets, [])

for i from 0 to length(array) - 1:

bucket\_index = floor((array[i] - minValue) / interval)

index = bucket\_index == num\_buckets ? num\_buckets - 1 : bucket\_index

push array[i] to buckets[index]

sortedArray = []

for each bucket in buckets:

bubbleSort(bucket)

append bucket to sortedArray

return sortedArray

## 1.2 Опис алгоритму сортування корзин

Основним алгоритмом для сортування вже розбитих корзин було обрано сортування бульбашкою. Це простий метод сортування, що порівнює пари сусідніх елементів масиву та, якщо вони знаходяться в неправильному порядку, обмінює їх. Цей процес повторюється доти, доки масив не буде відсортований. Асимптотична складність алгоритму складає O(n2), де n - кількість елементів у масиві.

Основні кроки алгоритму сортування бульбашкою:

* Проходи по масиву: Починаючи з початку масиву, алгоритм порівнює кожну пару сусідніх елементів.
* Порівняння і обмін елементів: Якщо елементи знаходяться в неправильному порядку (наприклад, елемент i більший за елемент i+1 у випадку сортування за зростанням), то вони обмінюються місцями.
* Повторення ітерації: Ці кроки повторюються для всього масиву до тих пір, поки всі елементи не будуть розташовані в правильному порядку.
* Кінець сортування: Алгоритм завершує роботу, коли під час одного повного проходу по масиву не відбувається жодного обміну елементів.

Сортування бульбашкою було обрано для забезпечення більшого навантаження на процеси, що дозволить наочно побачити плюси паралельної реалізації для основного алгоритму на великих обсягах даних.

Псевдокод алгоритму:

function bubbleSort(arr):

len = length(arr)

for i from 0 to len - 2:

for j from 0 to len - 2 - i:

if arr[j] > arr[j + 1]:

temp = arr[j]

arr[j] = arr[j + 1]

arr[j + 1] = temp

return arr

## 1.3 Опис паралельних реалізацій алгоритму

Паралельна реалізація алгоритму сортування корзинами передбачає розділення завдання на декілька підзадач, які можна виконувати паралельно. У контексті сортування корзинами, це означає, що розділення елементів у вихідному масиві на корзини та сортування кожної корзини можуть відбуватися паралельно.

Паралельне розділення і заповнення корзин не є вигідним по часу і ресурсам, тому найкращим рішенням буде зосередити увагу на обчисленнях, які вимагають найбільших обчислювальних потужностей, а саме сортування корзин.

Для реалізації паралельних обчисленнь у Node.js існує три відомих варіанти: Child Processes, Worker Threads, Clustering. Детальніше розглянемо кожен з них.

Child Processes дозволяють запускати окремі процеси для виконання важких обчислень, але вони можуть мати значні затримки через міжпроцесовий зв'язок і вимагають додаткової пам'яті та ресурсів.

Worker Threads забезпечують можливість створювати багатопоточні обчислення без міжпроцесового зв'язку, що дозволяє ефективніше використовувати ресурси обчислювальної системи.

Clustering дозволяє використовувати всю потужність багатоядерних систем для обробки багатьох запитів одночасно, але потребує ретельного керування станом додатку та обміном станом між процесами через міжпроцесовий зв'язок.

З усіх цих методів, Worker Threads є найбільш швидким і ефективним для Bucket Sort. Вони надають зручний інтерфейс для створення багатопоточних обчислень без зайвого для конкретної задачі міжпроцесового зв'язку. Крім того, вони можуть ефективно використовувати всі ресурси багатоядерних систем, що робить їх ідеальним вибором для паралельного виконання великих обчислень або обробки великих обсягів даних у Node.js додатках.

# 2 РОЗРОБКА ПОСЛІДОВНОГО АЛГОРИТМУ ТА АНАЛІЗ ЙОГО ШВИДКОДІЇ

# 3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ ТА ЙОГО КОРОТКИЙ ОПИС

# 4 РОЗРОБКА ПАРАЛЕЛЬНОГО АЛГОРИТМУ З ВИКОРИСТАННЯМ ОБРАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ

# 5 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ АЛГОРИТМУ

…представлені у таблиці 5.1…..

Таблиця 5.1. – Назва таблиці

|  |  |  |
| --- | --- | --- |
| Кількість елементів | Час послідовного алгоритму, мікросекунд | Час паралельного алгоритму, мікросекунд |
| 1000 | 2477 | 475 |
| 3000 | 2338 | 837 |

На рисунку 5.1….

…..(рис.5.3).

Рисунок 5.1. – Назва рисунку

Рисунок 5.3. – Результат виконання програми у консолі

# ВИСНОВКИ

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. …...
2. …..
3. …..
4. ….
5. …..

# ДОДАТКИ

## Додаток А. Назва додатку

## Додаток Б. Назва додатку

## Додаток В. Назва додатку