

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 злиттям та його паралельна реалізація мовою програмування Java

|  |  |
| --- | --- |
| **Керівник**:  ст. викладач  Дифучин Антон Юрійович  «Допущено до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 р.  Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Виконавець**:  Бондаренко Максим Вікторович  студент групи ІП-13  залікова книжка № ІП-1304  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «26» травня 2024 р.  Інна СТЕЦЕНКО  Антон ДИФУЧИН |

**Київ – 2024**

**ЗАВДАННЯ**

1. Виконати огляд існуючих реалізацій алгоритму, послідовних та паралельних, з відповідними посиланнями на джерела інформації (статті, книги, електронні ресурси). Зробити висновок про актуальність дослідження.
2. Виконати розробку послідовного алгоритму у відповідності до варіанту завдання та обраного програмного забезпечення для реалізації. Опис алгоритму забезпечити у вигляді псевдокоду. Провести тестування алгоритму та зробити висновок про коректність розробленого алгоритму. Дослідити швидкодію алгоритму при зростанні складності обчислень та зробити висновки про необхідність паралельної реалізації алгоритму.
3. Виконати розробку паралельного алгоритму у відповідності до обраного завдання та обраного програмного забезпечення для реалізації. Опис алгоритму забезпечити у вигляді псевдокоду. Забезпечити ініціалізацію даних при будь-якому великому заданому параметрі кількості даних.
4. Виконати тестування алгоритму, що доводить коректність результатів обчислень. Тестування алгоритму обов’язково проводити на великій кількості даних. Коректність перевіряти порівнянням з результатами послідовного алгоритму.
5. Виконати дослідження швидкодії алгоритму при зростанні кількості даних для обчислень.
6. Виконати експериментальне дослідження прискорення розробленого алгоритму при зростанні кількості даних для обчислень. Реалізація алгоритму вважається успішною, якщо прискорення не менше 1,2.
7. Дослідити вплив параметрів паралельного алгоритму на отримуване прискорення. Один з таких параметрів – це кількість підзадач, на які поділена задача при розпаралелюванні її виконання.
8. Зробити висновки про переваги паралельної реалізації обчислень для алгоритму, що розглядається у курсовій роботі, та програмних засобів, які використовувались.

**АНОТАЦІЯ**

**Структура та обсяг роботи.** Пояснювальна записка курсової роботи складається з N сторінок, 5 розділів, містить 21 рисунків, 44 таблиць, 4 додатки, 17 джерел.

**Мета.** Реалізувати та проаналізувати паралельну реалізацію алгоритму сортування злиттям на мові Java з використанням ForkJoin Framework, що допоможе оптимізувати роботу алгоритму у виконанні задачі сортування.

Робота складається з п’яти розділів, в яких описується алгоритм та його відомі паралельні реалізації, розробляється послідовний алгоритм та аналізується його швидкодія, обирається та описується програмне забезпечення для розробки паралельних обчислень, розробляється паралельний алгоритм з використанням обраного ПЗ, досліджується ефективність паралельних обчислень алгоритму.

КЛЮЧОВІ СЛОВА: СОРТУВАННЯ ЗЛИТТЯМ, FORKJOIN FRAMEWORK, JAVA, ПАРАЛЕЛЬНИЙ АЛГОРИТМ

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

Алгоритми сортування лежать в основі багатьох операцій з обробки даних. Одним із найпоширеніших алгоритмів є сортування злиттям, який характеризується своєю стабільністю та ефективністю. Даний алгоритм базується на концепції «розділяй і володарюй», бо він розділяє масиви на менші підмасиви, сортує їх окремо, а потім зливає в один відсортований масив. Стабільним та ефективним його робить часова складність *O*(*n*log*n*), яка зберігається у найгіршому, середньому та найкращому випадках.

Однак, навіть ефективні алгоритми потребують оптимізації, коли мова йде про серйозне збільшення розмірів даних та суворі вимоги до продуктивності. Ефективним рішенням даної проблеми є розпаралелення алгоритмів. Паралельні обчислення дозволяють розподіляти навантаження між кількома потоками, які виконують задачу одночасно, що сильно зменшує загальний час виконання задачі. У контексті нашої задачі розпаралелення може значно прискорити процес виконання, оскільки різні частини масиву можуть сортуватись одночасно.

У сучасному світі все більше комп’ютерів мають багатоядерні процесори, які здатні виконувати багато задач одночасно. Також великі обсяги даних стають дедалі частішою реальністю в сучасних додатках. Усім цим обумовлена доцільність розпаралелення алгоритму сортування злиттям, оскільки це допоможе використовувати ресурси більш ефективно, зменшуючи при цьому час виконання.

# 1 ОПИС АЛГОРИТМУ ТА ЙОГО ВІДОМИХ ПАРАЛЕЛЬНИХ РЕАЛІЗАЦІЙ

## 1.1 Опис алгоритму

Алгоритм сортування злиттям (merge sort) є класичним алгоритм, що працює за принципом «розділяй і володарюй». Основна ідея полягає в рекурсивному поділі масиву на менші підмасиви до тих пір, поки кожен з них не буде уявляти з себе масив довжиною 1, після ці підмасиви зливаються назад в один відсортований масив.

Алгоритм сортування складається з трьох основних етапів:

1. Розділення. Розбиття масиву на дві частини.
2. Рекурсивне сортування. Рекурсивно застосовуємо алгоритм до розділених частин.
3. Злиття. Зливаємо дві частини в один відсортований масив.

Нижче наведено псевдокод алгоритму сортування злиттям. Функція mergeSort(arr) – головна функція алгоритму, яка виконує всю його логіку. В якості вхідного параметру вона приймає масив arr. Якщо довжина масиву менша або дорівнює 1, то він повертається без змін, оскільки це є базовим випадком рекурсії. В іншому випадку масив розбивається на дві частини за індексом, який визначає середину масиву: , де mid – індекс середнього елементу, len – довжина масиву. Обидві частини сортуються рекурсивно за допомогою викликів відповідних методів mergeSort(arr[0:середина]) та mergeSort(arr[середина:довжина arr]). Після сортування частин вони зливаються в один відсортований масив за допомогою функції merge(left, right), яка приймає обидві частини в якості параметрів. Функція merge(left, right) – є допоміжною функцією для злиття двох відсортованих масивів в один масив. Вона почергово порівнює елементи обох масивів і додає до результуючого масиву менший елемент. Коли елементи одного масиву закінчуються, до результуючого масиву додаються всі елементи іншого, які залишились нерозподіленими.

Псевдокод алгоритму:

**функція** mergeSort(arr):

**якщо** довжина arr ≤ 1:

**повернути** arr

mid = довжина arr / 2

left = mergeSort(arr[0:середина])

right = mergeSort(arr[середина:довжина arr])

**повернути** merge(left, right)

**функція** merge(left, right):

result = []

i = 0, j = 0

**поки** i < довжина left і j < довжина right:

**якщо** left[i] ≤ right[j]:

додати left[i] до result

i = i + 1

**інакше**:

додати right[j] до result

j = j + 1

**поки** i < довжина left:

додати left [i] до result

i = i + 1

**поки** j < довжина right:

додати right[j] до result

j = j + 1

**повернути** result

Часова складність алгоритму становить *O*(*n*log*n*), що робить його достатньо ефективним для великих обсягів даних. Просторова складність становить *O*(*n*), оскільки потрібен простір для злиття підмасивів.

## 1.2 Назва підрозділу

Текст….Рисунок завжди ПІСЛЯ посилання на нього в тексті. Посилання оформлюємо в тексті або «…наведено на рисунку 1.1», або «…(рис.1.1)».

Рисунок. 1.1. – Назва рисунку

Текст….у формулі (1.1)

(1.1)

# 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