МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

**ЗВІТ**

**з дисципліни:**

**“** **Технології розподілених систем та паралельних обчислень”**

**по лабораторній роботі №3**

Виконав

студент групи ІПЗ-22011бск

Бурцев В.М

Київ – 2025

**Завдання:**

Розширити функції програми, що зроблена при виконанні лабораторних робіт № 1 і  № 2 та сформувати порівняльну таблицю результатів роботи програми при різних параметрах кластерізації вихідних масивів.

**Конкретизація завдання:**

1.     Масив максимальної розмірності (1 000 000 елементів або більше) розподіляється на N блоків (наприклад, N = 1,..., 50). Обчислення виконати мінімум три рази для кожного варіанту розподілу, і для подальшого аналізу порахувати середнє значення часу роботи програми.

2.     Фіксуючи час роботи процесора і програми для всіх прорахованих варіантів побудувати відповідну таблицю та графічно представити результати виконання завдання

3.     Зробити аналіз даних порівняльної таблиці і обгрунтувати висновки ефективності і доцільності використання розподіленої мережі та організації паралельних обчислень відповідно до задачі, яка вирішується.

4. Сформувати звіт.

**Опис та код програми**

const { Worker } = require("worker\_threads");

const path = require("path");

const fs = require("fs");

*//that calculates the sum for subarray*

function runWorker(*data*, *partName*) {

return new Promise((*resolve*, *reject*) => {

const startTime = process.hrtime.bigint();

const worker = new Worker(path.resolve(\_\_dirname, "worker.js"), {

workerData: { array: *data*, partName },

});

worker.on("message", (*message*) => {

const endTime = process.hrtime.bigint();

const totalTime = Number(endTime - startTime) / 1e6;

resolve({

result: *message*.result,

workerTime: *message*.time,

overallTime: totalTime,

partName,

array: *data*,

});

});

worker.on("error", reject);

worker.on("exit", (*code*) => {

if (*code* !== 0)

reject(new Error(`Worker stopped with exit code ${*code*}`));

});

});

}

*//that calculates the sum for numBlocks of subarrays*

async function runTestForBlocks(*array*, *numBlocks*) {

const subarraySize = Math.ceil(*array*.length / *numBlocks*);

const subarrays = [];

for (let i = 0; i < *numBlocks*; i++) {

const start = i \* subarraySize;

const end = start + subarraySize;

subarrays.push(*array*.slice(start, end));

}

const overallStart = process.hrtime.bigint();

const promises = subarrays.map((*sub*, *index*) =>

runWorker(*sub*, `Part-${*index* + 1}`)

);

const results = await Promise.all(promises);

const overallEnd = process.hrtime.bigint();

const overallTime = Number(overallEnd - overallStart) / 1e6;

const totalWorkerTime = results.reduce(

(*acc*, *curr*) => *acc* + *curr*.workerTime,

0

);

const totalSum = results.reduce((*acc*, *curr*) => *acc* + *curr*.result, 0);

return {

overallTime,

totalWorkerTime,

totalSum,

};

}

async function main() {

const maxElements = 1000000;

const testArray = Array.from({ length: maxElements }, () =>

Math.floor(Math.random() \* 100)

);

const maxBlocks = 50;

const runsPerTest = 3;

const tableData = [];

console.log(

`Виконується тестування з масивом ${maxElements} елементів, розподіленим на від 1 до ${maxBlocks} блоків (по ${runsPerTest} запусків для кожного варіанту).`

);

*//testing for different values N*

for (let numBlocks = 1; numBlocks <= maxBlocks; numBlocks++) {

let sumOverallTime = 0;

let sumWorkerTime = 0;

console.log(`\nТестування для ${numBlocks} блок(ів):`);

for (let run = 1; run <= runsPerTest; run++) {

console.log(` Запуск ${run}...`);

const { overallTime, totalWorkerTime, totalSum } = await runTestForBlocks(

testArray,

numBlocks

);

console.log(

` Результат: Загальний час: ${overallTime.toFixed(

3

)} мс, Сумарний час роботи воркерів: ${totalWorkerTime.toFixed(

3

)} мс, Сума: ${totalSum}`

);

sumOverallTime += overallTime;

sumWorkerTime += totalWorkerTime;

}

const avgOverallTime = sumOverallTime / runsPerTest;

const avgWorkerTime = sumWorkerTime / runsPerTest;

tableData.push({

numBlocks,

avgOverallTime,

avgWorkerTime,

});

}

console.log("\nПорівняльна таблиця результатів:");

console.log(

"Блоків\tСередній загальний час (мс)\tСередній сумарний час роботи воркерів (мс)"

);

tableData.forEach((*row*) => {

console.log(

`${*row*.numBlocks}\t${*row*.avgOverallTime.toFixed(

3

)}\t\t\t${*row*.avgWorkerTime.toFixed(3)}`

);

});

*//save files in csv*

let csvContent = "Blocks,AvgOverallTime(ms),AvgWorkerTime(ms)\n";

tableData.forEach((*row*) => {

csvContent += `${*row*.numBlocks},${*row*.avgOverallTime.toFixed(

3

)},${*row*.avgWorkerTime.toFixed(3)}\n`;

});

fs.writeFileSync("results.csv", csvContent);

console.log("\nРезультати збережено у файлі results.csv");

}

main().catch((*err*) => console.error(*err*));

const { workerData, parentPort } = require("worker\_threads");

*// calculate sum by method "divide-and-conquer"*

function recursiveSum(*arr*) {

if (*arr*.length === 0) return 0;

if (*arr*.length === 1) return *arr*[0];

const mid = Math.floor(*arr*.length / 2);

return recursiveSum(*arr*.slice(0, mid)) + recursiveSum(*arr*.slice(mid));

}

function main() {

const start = process.hrtime.bigint();

const result = recursiveSum(workerData.array);

const end = process.hrtime.bigint();

const timeElapsed = Number(end - start) / 1e6;

parentPort.postMessage({ result, time: timeElapsed });

}

main();

Генерується масив із 1 000 000 елементів.

const maxElements = 1000000;

const testArray = Array.from({ length: maxElements }, () =>

Math.floor(Math.random() \* 100)

);

Для кожного значення кількості блоків N від 1 до 50, масив розбивається на N підмасивів.

for (let numBlocks = 1; numBlocks <= maxBlocks; numBlocks++) {

Для кожного варіанту розподілу виконуються 3 запуску, під час яких за допомогою модуля worker\_threads обчислюється сума елементів кожного підмасиву.

let sumOverallTime = 0;

let sumWorkerTime = 0;

for (let run = 1; run <= runsPerTest; run++) {

Для кожного запуску фіксується:

**Загальний час виконання** (час роботи програми, що включає створення воркерів, їх виконання та збір результатів).

sumOverallTime += overallTime;

**Сумарний час роботи воркерів** (час, який кожен воркер витратив на виконання операції).

sumWorkerTime += totalWorkerTime;

Після тестування для кожного варіанту підраховується середнє значення часу (як загального, так і сумарного часу роботи воркерів).

const avgOverallTime = sumOverallTime / runsPerTest;

const avgWorkerTime = sumWorkerTime / runsPerTest;

Результати виводяться у консоль у вигляді таблиці

tableData.forEach((*row*) => {

console.log(

`${*row*.numBlocks}\t${*row*.avgOverallTime.toFixed(

3

)}\t\t\t${*row*.avgWorkerTime.toFixed(3)}`

);

, а також записуються у CSV-файл для подальшого графічного аналізу.

let csvContent = "Blocks,AvgOverallTime(ms),AvgWorkerTime(ms)\n";

tableData.forEach((*row*) => {

csvContent += `${*row*.numBlocks},${*row*.avgOverallTime.toFixed(

3

)},${*row*.avgWorkerTime.toFixed(3)}\n`;

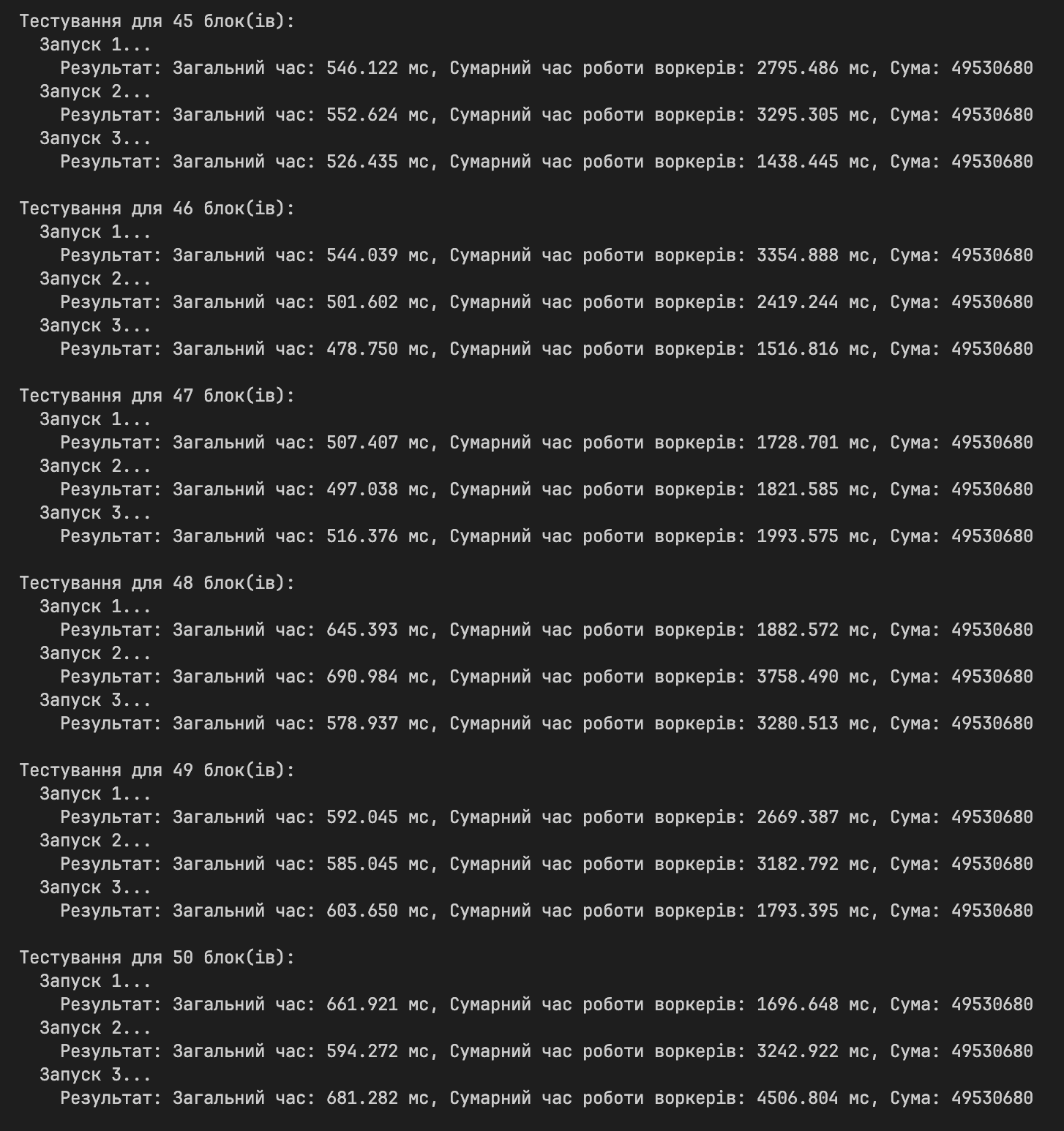
});

fs.writeFileSync("results.csv", csvContent);

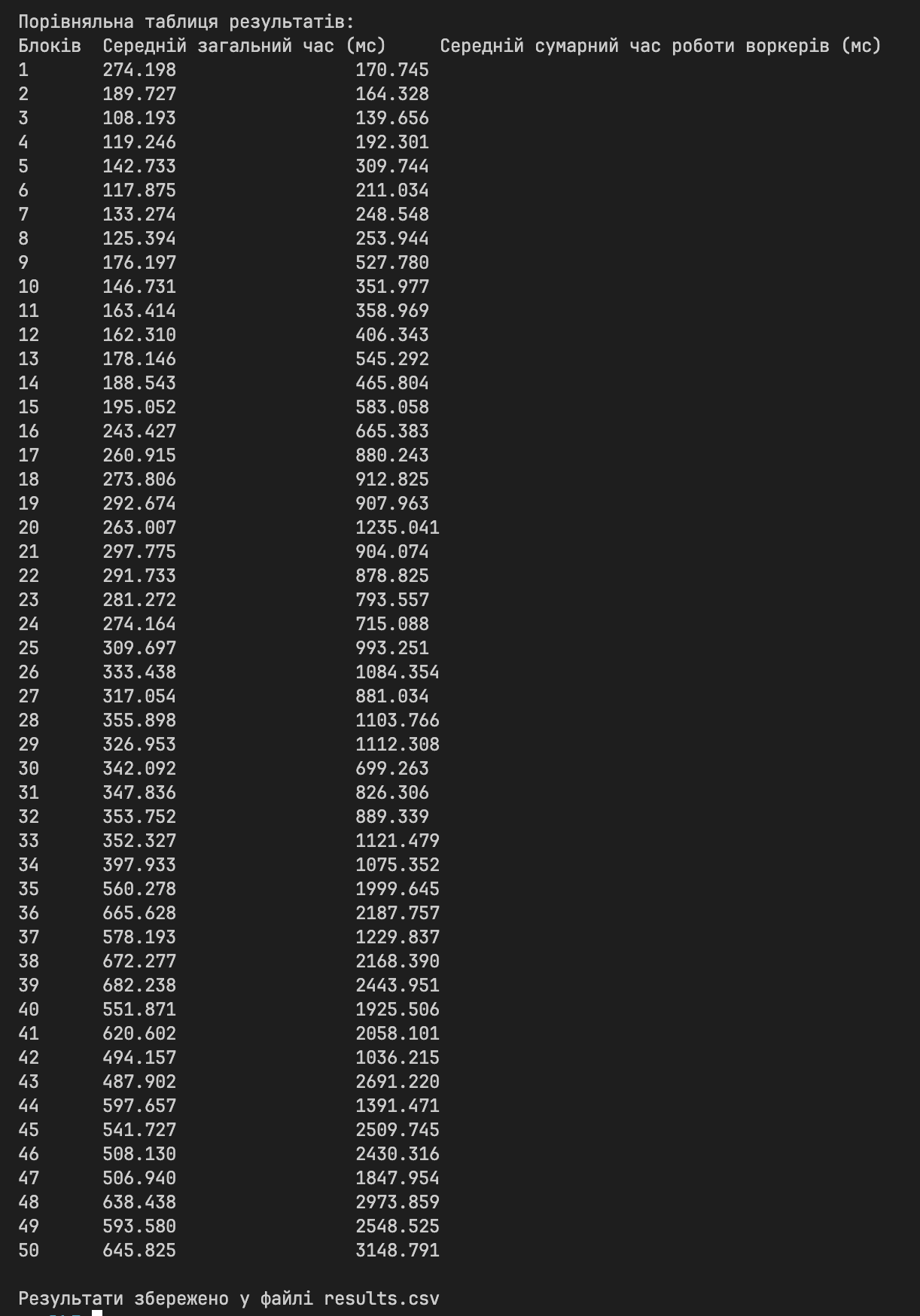
console.log("\nРезультати збережено у файлі results.csv");

**Опис та скріншоти виконання програми**

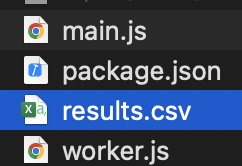
Приклад тестування для блоків



Таблиця результатів



Створений файл CSV



Збереженні дані в файлі CSV

