МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРОННИХ ТА ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ

Засоби інтеграції розподілених систем

Методичні вказівки
до виконання розрахунково-графічної та самостійної роботи
для здобувачів вищої освіти
спеціальності 121 – "**Інженерія програмного забезпечення**"
рівень вищої освіти – *перший (бакалаврський)*

Обговорено і рекомендовано
на засіданні кафедри інформаційних
технологій та програмної інженерії
протокол № 1 від 31.08.2021

Чернігів, НУ «Чернігівська політехніка», 2021

Засоби інтеграції розподілених систем. Методичні вказівки до виконання розрахунково-графічної та самостійної роботи для здобувачів вищої освіти спеціальності 121 – "Інженерія програмного забезпечення", рівень вищої освіти – перший (бакалаврський) /Укл.: Білоус І.В. – РВВ НУ «Чернігівська політехніка», 2021. – 15 с.

Укладач: Білоус Ірина Володимирівна, кандидат технічних наук, доцент, завідувач кафедри інформаційних технологій та програмної інженерії

Відповідальний за випуск: Войцеховська М.М., доктор філософії, викладач кафедри інформаційних технологій та програмної інженерії

Рецензент: Бичко В.А., кандидат фізико-математичних наук, доцент, доцент кафедри інформаційних та комп’ютерних систем Національного університету "Чернігівська політехніка"

ЗМІСТ

ВСТУП

Лабораторні роботи є сполучною ланкою між лекційними заняттями і самостійною роботою здобувачів ВО. В процесі виконання лабораторних робіт експериментально перевіряються ключові питання курсу «Засоби інтеграції розподілених систем», набуваються практичні навички проектування і реалізації розподіленої взаємодії з використанням засобів інтеграції розподілених систем, перевіряється рівень засвоєння основних положень дисципліни.

Виконання розрахунково-графічної роботи є частиною самостійної роботи здобувачів над дисципліною. В процесі виконання роботи закріплюються ключові питання курсу, набуваються практичні навички проектування і реалізації розподіленої взаємодії з використанням засобів інтеграції розподілених систем, перевіряється рівень засвоєння основних положень дисципліни. Здобувачі експериментально перевіряють ключові питання курсу «Засоби інтеграції розподілених систем».

Методичні рекомендації до виконання розрахунково-графічної та самостійної роботи орієнтовані на використання засобів інтеграції розподілених систем Java RMI, MPI, Java WSO2 Microservices Framework. Передбачається, що здобувачі знайомі з основами роботи з цими технологіями, мовами програмування Java/C++ та володіють англійською мовою технічного рівня. Розрахунко-графічна робота може бути виконана за допомогою інших технологій розподіленого програмування за погодженням з викладачем.

Під час семестру дисципліни здобувач показує викладачеві результати виконання роботи, проводить консультації з виниклих питань та завершує роботу. Здобувач зобов'язаний до початку залікового тижня виконати та захистити розрахунково-графічну роботу. Обсяг виконаної роботи може бути різним, залежно від того, на яку оцінку претендує здобувач. Коли робота закінчена, здобувач повинен її захистити. Захист полягає в відповідях на питання по темі роботи і внесення деяких змін в розроблювану систему в присутності викладача.

По розрахунково-графічній роботі здобувач повинен оформити звіт. Звіти оформляються за допомогою текстового редактора Word на папері формату А4 відповідно до вимог стандартів на оформлення технічної документації. Звіт по роботі є розділом підсумкового документа.

За розрахунково-графічну роботу здобувач може отримати до 20 балів, з урахуванням своєчасності і якості виконання всіх складових роботи. Складовими є: звіт, проект, відповідність оформлення вимогам і відповіді на контрольні питання. Оцінка, отримана за розрахунково-графічну роботу враховуються при виставленні семестрової оцінки поточного контролю. Для отримання допуску на семестровий контроль має бути здана і захищена розрахунково-графічна робота на оцінку не менше ніж 12 балів.

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА. СТВОРЕННЯ І РЕАЛІЗАЦІЯ ЕФЕКТИВНОГО ПАРАЛЕЛЬНОГО ТА РОЗПОДІЛЕНОГО АЛГОРИТМУ

1 Мета роботи

- створити паралельний алгоритм завдання згідно з варіантом;
- вибрати засоби для його розподіленої реалізації;
- реалізувати додаток згідно з алгоритмом;
- обґрунтувати ефективність створеного алгоритму.

2 Завдання до розрахунково-графічної роботи

Розрахунково-графічна робота має за мету створення паралельного та розподіленого алгоритму завдання згідно з таблицею 2.1. По кожному завданню є можливість реалізації створеного алгоритму різними засобами (з наведених в таблиці 2.1 або іншими за погодженням з викладачем). Тому в описовій частині до роботи має бути обґрунтоване обрання відповідного засобу. Номер варіанту вибирається в довільному порядку, забезпечуючи при цьому неповторюваність завдання і засобу його реалізації.

Таблиця 2.1 – Завдання для роботи

Варіант	Завдання
1	Напишіть розподілений додаток множення квадратної матриці довільного розміру N на вектор довжини N засобами:
	1.1 Java RMI 1.2 MPI 1.3 Java WSO2 Microservices Framework
2	Напишіть розподілений додаток, що виводить на екран таблицю множення довільного розміру N у вигляді матриці засобами :
	2.1 Java RMI 2.2 MPI 2.3 Java WSO2 Microservices Framework
3	Напишіть розподілений додаток, що розбиває число N на прості множники. Для цього визначите масив простих чисел, і перевіряйте подільність числа N на відповідні прості числа з масиву і зменшуйте (ділячи) N у разі подільності без залишку. Завдання реалізуйте засобами:
	3.1 Java RMI 3.2 MPI 3.3 Java WSO2 Microservices Framework
4	Напишіть розподілений додаток, що обчислює мінімаксимум в квадратній матриці розміру N засобами :
	4.1 Java RMI 4.2 MPI 4.3 Java WSO2 Microservices Framework

5	Напишіть розподілений додаток сортування масиву розмірності N засобами:
	5.1 Java RMI 5.2 MPI 5.3 Java WSO2 Microservices Framework
6	Напишіть розподілений додаток для множення квадратної матриці розмірності N на квадратну матрицю розміру N засобами :
	6.1 Java RMI 6.2 MPI 6.3 Java WSO2 Microservices Framework
7	Напишіть розподілений додаток пошуку елементу у відсортованому масиві розмірності N засобами:
	7.1 Java RMI 7.2 MPI 7.3 Java WSO2 Microservices Framework
8	Напишіть розподілений додаток, що обчислює число Фібоначчі N засобами :
	8.1 Java RMI 8.2 MPI 8.3 Java WSO2 Microservices Framework
9	Напишіть розподілений додаток, що обчислює найбільшого загального дільника (до числа M) в масиві розмірності N засобами :
	9.1 Java RMI 9.2 MPI 9.3 Java WSO2 Microservices Framework
10	Напишіть розподілений додаток обчислення інтеграла однією змінною методом прямокутників засобами:
	10.1 Java RMI 10.2 MPI 10.3 Java WSO2 Microservices Framework
11	Напишіть розподілений додаток сортування квадратної матриці довільного розміру N по горизонталі справа-наліво:
	11.1 Java RMI 11.2 MPI 11.3 Java WSO2 Microservices Framework
12	Напишіть розподілений додаток пошуку квадратної підматриці розміру M з максимальною сумою елементів квадратної матриці розміру N засобами:
	12.1 Java RMI 12.2 MPI 12.3 Java WSO2 Microservices Framework

3 Приклад створення і реалізації алгоритму

Нижче розглядається приклад створення алгоритму і його реалізації, який подібний до того, що слід розробити.

3.1 Завдання до роботи

Написати розподілений додаток, що обчислює суму чисел в масиві великої розмірності.

3.2 Визначення моделі, виду, типу алгоритму згідно узагальненої класифікації

Узагальнена класифікація паралельних алгоритмів

Здатність алгоритму до розпаралелювання потенційно пов'язана з однією з двох (чи одночасно з обома) внутрішніх властивостей, які характеризуються як паралелізм завдань (message passing) і паралелізм даних (data parallel).

Основні особливості підходу data parallel:

- обробкою даних управляє одна програма;
- простір імен є глобальним;
- слабка синхронізація обчислень на розподілених вузлах;
- паралельні операції над елементами масиву виконуються одночасно на усіх доступних вузлах.

Message passing – режим обчислень, при якому обчислювальне завдання розбивається на декілька відносно самостійних підзадач і кожен вузол обчислює окрему підзадачу. Для кожної підзадачі пишеться своя власна програма на звичайній мові програмування.

- підвищена трудомісткість розробки і відладки програми;
- програміст відповідає за рівномірне і збалансоване завантаження вузлів;
- мінімізація обміну даними між завданнями;
- виникнення конфліктів.

Основні моделі паралельного програмування

1) Модель процес/канал

Процес – виконувана на процесорі програма, використовує для своєї роботи частину локальної пам'яті процесора і містить ряд операцій прийому/передачі даних для організації інформаційної взаємодії між виконуваними процесами паралельної програми. Канал передачі даних – черга повідомлень, в яку один або декілька процесів можуть відправляти дані, що пересилаються, і з якої процес-адресат може витягати повідомлення.

2) Модель обмін повідомленнями

Програми цієї моделі, створюють безліч процесів, з кожним з яких асоційовані локальні дані. Кожен процес ідентифікується унікальним ім'ям. Процеси взаємодіють, посилаючи і отримуючи повідомлення. У таких системах кожен процес виконує одну і ту ж програму, але працює з різними даними.

3) Модель паралелізму даних

Програма з паралелізмом даних складається з послідовностей операцій, одна і та ж операція застосовується до безлічі елементів структур даних. Оскільки

операції над кожним елементом даних можна розглядати як незалежні процеси, то міра деталізації таких обчислень дуже велика, а поняття «локальності» даних відсутнє.

4) Модель загальна пам'ять

Усі процеси спільно використовують загальний адресний простір, до якого вони асихронно звертаються із запитами на читання і запис. Для управління доступом до загальної пам'яті використовуються всілякі механізми синхронізації типу семафорів і блокувань процесів.

Класифікація алгоритмів за типом паралелізму

1) Алгоритми, що використовують паралелізм даних (Data Parallelism). Цей тип паралелізму характерний для чисельних алгоритмів обробки, що мають справу з великими масивами, що представляються, наприклад, у вигляді векторів і матриць. Простим прикладом такого завдання є, наприклад, процедура перемножування двох матриць.

2) Алгоритми з розподілом даних (Data Partitioning). Це різновид паралелізму даних, при якому простір даних може бути розділений на області, що не перетинаються, з кожною з яких пов'язані незалежні процеси, що оперують кожен зі своїми даними. Потрібно лише рідкісний обмін між цими процесами.

3) Релаксаційні алгоритми (Relaxed Algorithm). Алгоритм може бути представлений у вигляді незалежних процесів без синхронізації зв'язку між ними, але вузли повинні мати доступ до загальних даних.

4) Алгоритми з синхронізацією ітерацій (Synchronous Iteration). Багато хто із стандартних чисельних ітераційних паралельних алгоритмів вимагає синхронізації у кінці кожної ітерації, що полягає в тому, що дозвіл на початок наступної ітерації дається після того, як усі процесори завершили попередню ітерацію.

5) Самовідтворювані завдання (Replicated Workers). Для завдань цього класу створюється і підтримується центральний пул (сховище) схожих обчислювальних завдань. Процеси, що паралельно реалізовуються, здійснюють вибір завдань з пулу, виконання необхідних обчислень і додавання нових завдань до пулу. Обчислення закінчуються, коли пул порожній. Ця технологія характерна для досліджень графа або дерева.

6) Конвеєрні обчислення (Pipelined Computation). Цей тип обчислень характерний для процесів, які можуть бути представлені у вигляді деякої регулярної структури, наприклад, у вигляді кільця або двовимірної мережі. Кожен процес, що знаходиться у вузлі цієї структури, реалізує певну фазу обчислень.

Неважко помітити, що деякі алгоритми з приведеного списку мають явно виражені властивості паралелізму завдань або паралелізму за даними. В той же час ряд алгоритмів в тій чи іншій мірі має обидві вказані властивості. Це слід враховувати при виборі способу і схеми декомпозиції задачі на підзадачі.

Опис рішення поставленої задачі

Для вирішення цього завдання пропонується використати наступний алгоритм здвоювання, коли одночасно сумуються сусідні пари доданків, а потім їх суми (рис. 3.1).

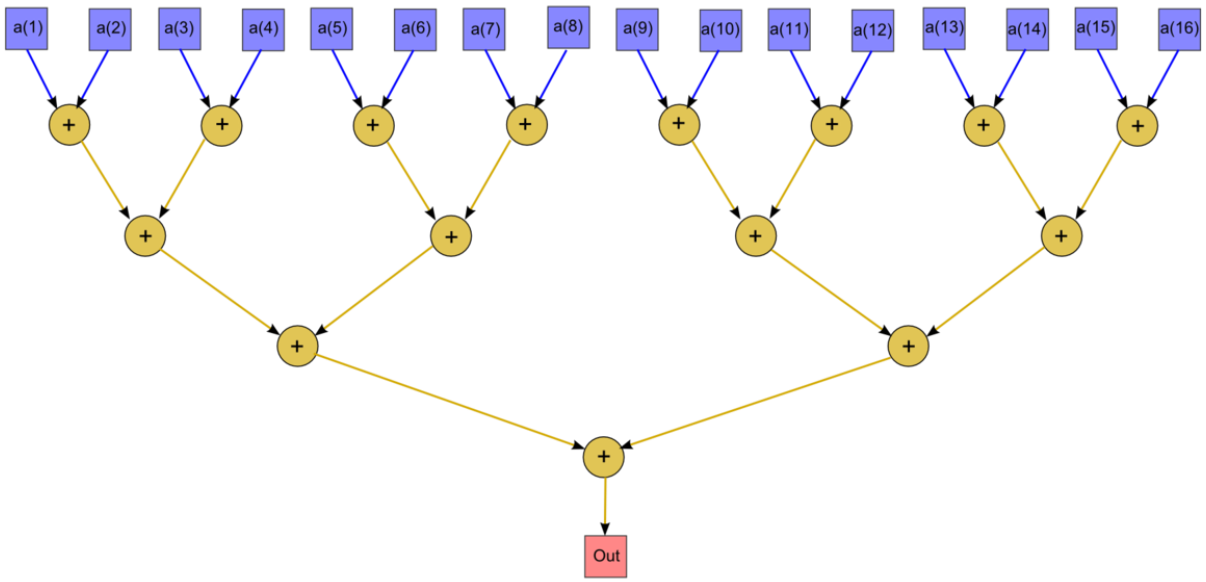

Рисунок 3.1 – Ілюстрація алгоритму здвоювання

Завдання підсумовування розділяється на незалежні підзадачі обчислення суми в масиві, розбитому на стільки підмасивів, скільки обчислювальних вузлів є в наявності. Розмір отриманих підмасивів не може бути менше порогу, при якому обчислення суми на одному вузлі ефективніше, ніж розподілення завдання. Це розбиття може бути реалізоване і рекурсивно.

Таким чином, цей алгоритм:

– має внутрішню властивість паралелізму за даними;
– відноситься до моделі паралелізму даних;
– є алгоритмом з розподілом даних;
– може бути дрібнозернистим при досить великому числі обчислювальних вузлів.

Оскільки алгоритм має внутрішню властивість паралелізму за даними, не має залежностей між підзадачами і потрібний лише рідкісний обмін між цими підзадачами можна обрати засіб MPI.

3.3 Опис етапів розробки розподіленого алгоритму

Декомпозиція. На цьому етапі виконуються аналіз задачі і оцінка можливості розпаралелювання. Задача і пов'язані з нею дані розділяються на дрібніші частини – підзадачі і фрагменти структур даних. Особливості архітектури конкретної обчислювальної системи на цьому етапі можуть не враховуватися.

Алгоритм розпаралелювання задачі підрахунку суми елементів масиву допускає розділення завдання на підзадачі з обчисленням елементів підмасиву такої розмірності, що не перевищує певного порогу.

Проектування комунікацій (обмінів даними) між завданнями. Визначаються комунікації, необхідні для пересилки початкових даних, проміжних результатів виконання підзадач, а також комунікації, необхідні для управління роботою підзадач. Вибираються методи комунікації.

Алгоритм розпаралелювання завдання підрахунку суми елементів масиву допускає пересилку початкового масиву (чи його частини) і очікування обчислення результатів підзадач. Комунікації, необхідні для управління роботою підзадач не передбачаються.

Укрупнення. Підзадачі об'єднуються у більші блоки, якщо це дозволяє підвищити ефективність алгоритму і понизити трудомісткість розробки.

Проектований алгоритм допускає розбиття на підзадачі і обчислення результатів в підмасивах досить маленького розміру. Проте, занадто дрібні підзадачі не можуть дати максимального прискорення у зв'язку з надмірними витратами на організацію розподілених обчислень.

Тому необхідно визначити «поріг», при якому завдання має сенс розпаралелювання, а також необхідно гарантувати рівномірний розподіл підмасивів по вузлах.

Планування обчислень. Розподіл підзадач між вузлами. Основний критерій вибору способу розміщення підзадач – ефективне використання вузлів з мінімальними витратами часу на обміни даними.

При використанні MPI можливе обрання ефективної схеми розміщення даних між вузлами з використанням відповідних регулярних структур (типу решітки). Однак для даної задачі не має потреби в такій організації даних на вузлах.

3.4 Розрахунки теоретичних показників ефективності і доказ ефективності запропонованого алгоритму

Загальна оцінка часу виконання послідовного алгоритму :

$$T_1 = (n/2+n/4+\ldots+1)\, t_1 \rightarrow nt_1 \qquad (3.1)$$

де n – розмірність початкової задачі;

t_1 – час виконання базової операції складання.

Загальна оцінка часу виконання паралельного алгоритму :

$$T_p = nt_1 / p + \alpha_2 * t_2 + \alpha_3 * t_3 \qquad (3.2)$$

де p – кількість обчислювальних вузлів;

α_2 – кількість послідовних блоків;

t_2 – час на виконання послідовних блоків;

α_3 – кількість пересилок даних;

t_3 – час на пересилку даних.

Якщо реалізовувати алгоритм на одній машині, накладні витрати на пересилку даних нікчемно малі ($\alpha_3, t_3 = 0$), а послідовні блоки виникають із-за необхідності обслуговувати 2P-1 потоків, при цьому час на обслуговування одного потоку для різних ОС може бути різним.

Загальна оцінка показників прискорення і ефективності при нескінченно великому n:

$$Sp = T_1 / T_p = n * t_1 / (n * t_1 / p) \approx p \qquad (3.3)$$

$$Ep = S_p / p \approx 1 \qquad (3.4)$$

Розроблений спосіб розподілених обчислень дозволяє досягти ідеальних показників прискорення і ефективності тільки при використанні загальної пам'яті.

У разі реалізації розподілених обчислень необхідно враховувати накладні витрати на пересилку даних. Часто при аналізі масштабованості отриманого алгоритму необхідно вивести функцію ізоефективності.

Накладні витрати можна описати так:

$$T_0 = pT_p - T_1 = p((n^* t_1 / p + \log_2 p * t_3) - n * t_1 = p * \log_2 p * t_3 \quad (3.5)$$

Тоді функція ізоефективності набирає вигляду:

$$n = K * p \log_2 p * t_3 \quad (3.6)$$

Для забезпечення рівня ефективності E=0,5 (тобто K=1), при збільшенні числа процесорів з p до q (q>p) для забезпечення пропорційного зростання прискорення (Sq/Sp)=(q/p) необхідно збільшити число підсумовуваних значень n в $(q\log_2 q)/(p\log_2 p)$ раз.

3.5 Тексти створених програм з поясненнями у вигляді коментарів для усіх основних елементів

```
#include <stdio.h>
#include "mpi.h"
#define n 1000
int main(int argc, char **argv)
{
  int rank, i, size, nproc;
  double time_start, time_finish;
  double a[n], b[n], c[n];
  MPI_Status status;
  MPI_Init(&argc, &argv);
// отримання кількості доступних вузлів
  MPI_Comm_size(MPI_COMM_WORLD, &size);
// отримання поточного номеру вузла
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  nproc = size;
  for(i = 0; i<n; i++) a[i] = 1.0/size;
// синхронізація всіх процесів за допомогою бар'єрів
  MPI_Barrier(MPI_COMM_WORLD);
  time_start = MPI_Wtime();
  for(i = 0; i<n; i++) c[i] = a[i];
  while(nproc>1){
    if(rank<nproc/2){
// отримання сумованих значень
      MPI_Recv(b, n, MPI_DOUBLE, nproc-rank-1, 1, MPI_COMM_WORLD,
&status);
      for(i = 0; i<n; i++) c[i] = c[i] + b[i];
    }
    else if(rank<nproc)
```

```
// розсилка значень масиву
        MPI_Send(c, n, MPI_DOUBLE, nproc-rank-1, 1, MPI_COMM_WORLD);
      nproc = nproc/2;
    }
    for(i = 0; i<n; i++) b[i] = c[i];
    time_finish = MPI_Wtime()-time_start;
    if(rank==0) printf("model b[1]=%lf\n", b[1]);
    printf ("rank=%d model time=%lf\n", rank, time_finish);
    for(i = 0; i<n; i++) a[i] = 1.0/size;
    MPI_Barrier(MPI_COMM_WORLD);
    time_start = MPI_Wtime();
// збір сумованих значень
    MPI_Reduce(a, b, n, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    time_finish = MPI_Wtime()-time_start;
    if(rank==0) printf("reduce b[1]=%lf\n", b[1]);
    printf("rank=%d reduce time =%lf\n", rank, time_finish);
    MPI_Finalize();
}
```

3.6 Результати тестування додатку

Тестування додатку необхідно проводити для декількох значень розмірності початкового завдання (N) і надати:

– дані про час виконання для послідовного варіанту рішення;

– дані про час виконання при паралельному виконанні на різних системах (як мінімум 2, 4 і 8 обчислювальних вузлів у разі розподіленого запуску);

– значення прискорення і ефективності.

Отримані дані надати в табличному виді і у вигляді графіків залежності отриманих показників від кількості обчислювальних вузлів.

Результати запуску розподіленого алгоритму підрахунку суми елементів масиву великої розмірності представлені в таблиці 3.1 і на рисунках 3.2-3.3.

Таблиця 3.1. – Результати запуску на різних системах

		2 вузли	4 вузли	8 вузлів
N=800 млн.	**T1**	0,737	0,737	0,737
	Tp	0,383	0,232	0,136
	Sp	1,9	3,2	5,4
	Ep	0,95	0,8	0,675
N=400 млн.	**T1**	0,379	0,379	0,379
	Tp	0,205	0,122	0,076
	Sp	1,85	3,1	5
	Ep	0,925	0,775	0,625
N=200 млн.	**T1**	0,195	0,195	0,195
	Tp	0,108	0,065	0,042
	Sp	1,8	3,0	4,6
	Ep	0,9	0,75	0,575

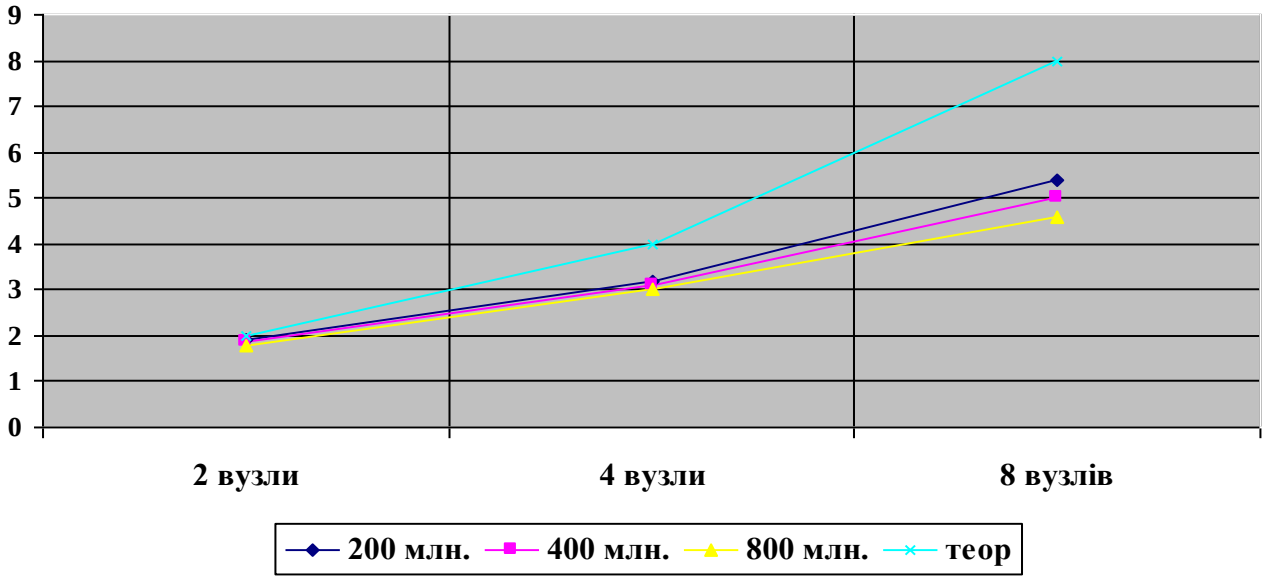

Рисунок 3.2 – Результати запуску на різних вузлах. Показники прискорення

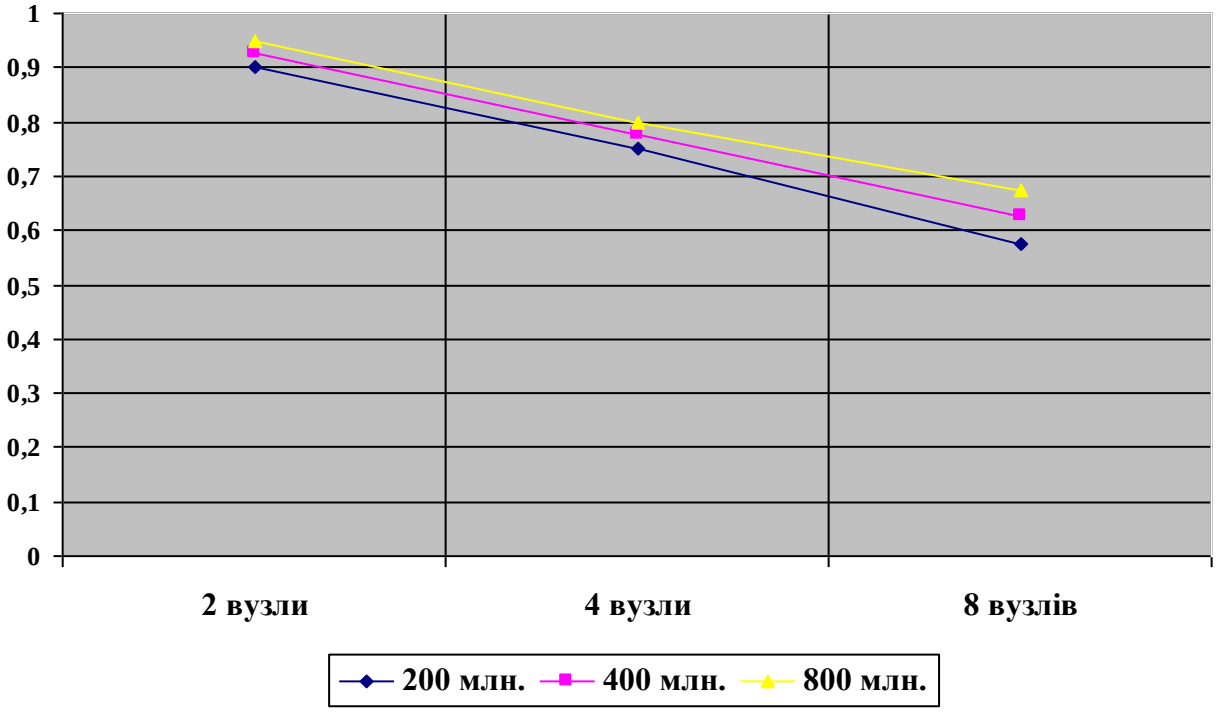

Рисунок 3.3 – Результати запуску на різних системах. Показники ефективності

3.7 Порівняння теоретичних і отриманих показників ефективності

У п. 3.4 у формулах 3.4-3.5 виведено наближене значення прискорення при паралельній реалізації – кількість доступних обчислювальних вузлів, значення ефективності – 1.

Як видно з рисунків 3.2-3.3 ефективність алгоритму дещо знижується при збільшенні числа обчислювальних вузлів, доступних системі, – від 0,95 до 0,575. Це пояснюється необхідністю обслуговувати пересилку даних між 2P-1 вузлами,

при цьому час на обслуговування пересилки досить великий в порівнянні з обробкою даних вибраної розмірності.

Для поліпшення ефективності алгоритму можна:

- збільшити розмірність початкового масиву;
- провести укрупнення підзадач;
- спланувати обчислення на вузлах подібної обчислювальної потужності.

4 Зміст звіту

- Найменування роботи.
- Мета роботи.
- Завдання до роботи з обґрунтуванням обрання засобу реалізації.
- Визначення моделі, виду, типу алгоритму згідно узагальненої класифікації.
- Опис етапів розробки паралельного та розподіленого алгоритму.
- Розрахунки передбачуваних показників ефективності і доказ ефективності запропонованого алгоритму.
- Тексти створених програм з поясненнями у вигляді коментарів для усіх основних елементів.
- Результати тестування проекту, у вигляді графіку запуску додатку для різного числа обчислювальних вузлів (від 1 до 8) і різної розмірності завдання (N).
- Порівняння теоертичних і отриманих показників ефективності.
- Висновки про особливості розподіленого програмування поставленого завдання.

5 Рекомендована література

1) Аксак Н.Г. Паралельні та розподілені обчислення: підручник / Н.Г. Аксак, О.Г., Руденко, А.М. Гуржій. – Х.: Компанія СМІТ, 2009. – 480с.

2) Хорстман К., Корнелл Г. Java. Библиотека профессионала, том 2. Расширенные средства, 9е изд.: пер. с англ. – М.: ООО «И.Д. Вильямс», 2014. – 1008с.

3) Таненбаум Э. С., Стин М. Распределенные системы. – М.: ДМК Пресс, 2020. – 584 с.

4) Дейтел, Х.М. Технологии программирования на Java 2: пер. с англ. Кн. 2 Распределённые приложения / Х.М. Дейтел, П.Дж. Дейтел, С.И. Сантри. – М.: Бином-Пресс, 2003. – 464 с.

5) Бэкон, Д. Операционные системы: параллельные и распределенные системы / Д. Бэкон, Т. Харрис. – СПб; К. : Питер; Изд. гр. BHV, 2004. – 799 С.

6) Бернс Б. Распределенные системы. Паттерны проектирования: пер. с англ. – СПб.: Питер, 2019. – 224 с.

7) Малявко А.А. Параллельное программирование на основе технологий OpenMP, MPI, CUDA 2-е изд., испр. и доп. Учебное пособие для вузов. – М.: Юрайт, 2019. – 116 с.

8) Ньюмен С. Создание микросервисов : пер. с англ. – СПб: Питер, 2018. – 304 с.

9) Kasun Indrasiri, Prabath Siriwardena, Microservices for the Enterprise – San Jose, CA, USA, 2018. – 434 с.

10) Christian Posta, Microservices for Java Developers – O'Reilly Media, 2015. – 129 с.

11) Boni García, Mastering Software Testing with JUnit 5 – Packt Publishing, 2017. – 399 с.