НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

_____Кафедра обчислювальної техніки_____
(повна назва кафедри, циклової комісії)

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з дисципліни «Паралельне програмування»
(назва дисципліни)

на тему: «Розробка програмного забезпечення для паралельних комп'ютерних систем з локальною памяттю»

Студента (ки) 3 курсу ІО-21 групи
спеціальності
123 «Комп'ютерна інженерія»

Безщасний Р. Р.
(прізвище та ініціали)

Керівник доцент Корочкін О.В.

Кількість балів: _____

Оцінка: ECTS _____

Київ - 2024 рік

Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"

Факультет (інститут) інформатики та обчислювальної техніки
(повна назва)

Кафедра обчислювальної техніки
(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 123 «Комп'ютерна інженерія»
(шифр і назва)

З А В Д А Н Н Я
НА РГР СТУДЕНТУ

Безщасному Роману Руслановичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для паралельних комп'ютерних систем з локальною памяттю»
керівник роботи Корочкін Олександр Володимирович к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 24 12 2024 р.
3. Вхідні дані до роботи
   - математична задача
   - структури ПКС ЛП
   - мови (бібліотеки) паралельного програмування
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - огляд засобів програмування для ПКС ЛП
   - розробка програми
   - тестування програми
5. Перелік графічного матеріалу
   - структурна схема ПКС ЛП
   - схема взаємодії потоків

6. Дата видачі завдання 23 12 2024.

ЗМІСТ

ВСТУП

Розрахунково-графічна робота (РГР) має на меті закріпити теоретичні знання та сформувати практичні навички, пов’язані з розробкою, аналізом і тестуванням паралельних алгоритмів і програм для комп’ютерних систем із локальною пам’яттю (ПКС ЛП). У рамках завдання передбачається опанування мов паралельного програмування та спеціалізованих бібліотек, таких як MPI, які реалізують обмін даними між потоками за допомогою передачі повідомлень.

Робота зосереджена на створенні програмного продукту, що забезпечує виконання паралельних обчислень у середовищі систем із локальною пам’яттю. Формули для обчислень, а також структура взаємодії процесорів і пристроїв вводу-виводу встановлюються згідно з варіантом завдання.

Основними етапами є розробка алгоритму, який відповідає заданій математичній моделі, його реалізація у вигляді паралельної програми, тестування з різними наборами вхідних даних та аналіз ефективності отриманого рішення. Для управління взаємодією потоків використовуються синхронізаційні механізми, зокрема семафори, бар’єри або черги повідомлень.

Результатом роботи є програмний застосунок, спроектований згідно з визначеною схемою взаємодії процесорів і пристроїв вводу-виводу, який виконує задані обчислення. Додатково проводиться аналіз продуктивності системи, включаючи оцінку розподілу завдань і його впливу на швидкодію, що сприяє вдосконаленню методів оптимізації паралельних алгоритмів.

Таким чином, виконання РГР сприяє поглибленню теоретичних знань і розвитку професійних компетенцій у галузі програмування для паралельних обчислювальних систем. Окрему увагу приділяють вибору оптимальних підходів до реалізації паралельних задач із врахуванням апаратних обмежень та практичних умов роботи системи.

РОЗДІЛ 1 ОГЛЯД БІБЛІОТЕКИ MPI

1.1 Засоби програмування потоків

Взаємодія між процесорами у комп'ютерних системах із локальною пам'яттю є критично важливою для ефективного виконання паралельних обчислень. Для цього широко використовується стандарт **MPI (Message Passing Interface)** — набір функцій, які забезпечують обмін повідомленнями між потоками.

Розроблений у 1993–1994 роках групою MPI Forum, MPI став універсальним рішенням для побудови паралельних програм. Основна мета стандарту — забезпечити ефективний механізм комунікації між процесами, які можуть працювати на різних обчислювальних вузлах. MPI підтримує синхронізацію процесів, передачу даних і управління ресурсами, що дозволяє адаптувати алгоритми для різноманітних задач і платформ.

Переваги MPI

1. Масштабованість
MPI ідеально підходить для масштабних обчислювальних систем. Завдяки механізмам оптимального розподілу задач між процесами стандарт забезпечує ефективну роботу навіть на суперкомп'ютерах.

2. Універсальність
Реалізації MPI доступні для багатьох мов програмування. Це дозволяє використовувати один і той самий підхід у різних середовищах та інтегрувати його в різні проєкти.

3. Гнучкість обміну даними
MPI підтримує як точковий, так і колективний обмін повідомленнями, що робить його адаптивним до будь-яких потреб програми.

4. Продуктивність

Мінімальні накладні витрати на обмін даними роблять MPI одним із найефективніших рішень для паралельних обчислень.

5. Підтримка складних алгоритмів

MPI дозволяє легко реалізовувати складні моделі комунікації, необхідні для виконання сучасних алгоритмів.

MPI, у поєднанні з Python через mpi4py, є потужним інструментом для створення паралельних програм, здатним працювати з великими обсягами даних та складними моделями взаємодії потоків.

1.2 Засоби організації взаємодії потоків

У роботі використовується мова Python та бібліотека **mpi4py**, яка є повною реалізацією стандарту MPI. Python обрано завдяки її сучасності, простоті та наявності великої кількості бібліотек, що спрощують реалізацію математичних обчислень. mpi4py дозволяє інтегрувати паралельну обробку даних безпосередньо у Python-код із використанням стандартних функцій MPI.

Можливості бібліотеки mpi4py:

1. **Ідентифікація процесів**:
   - Get_rank() повертає унікальний номер поточного процесу (rank).
   - Get_size() визначає загальну кількість процесів у комунікаторі.
2. **Обмін повідомленнями**:
   - **Точковий обмін**:
     - Send(data, dest) — надсилає дані у вказаний процес.

- Recv(source) — приймає дані від зазначеного процесу.

- **Колективні операції**:

- bcast(data, root) — транслює дані від одного процесу до всіх інших.

- scatter(data, root) — розподіляє частини даних між усіма процесами.

- gather(data, root) — збирає дані від усіх процесів у кореневий процес.

- reduce(data, op, root) — виконує колективну операцію (наприклад, суму) над даними всіх процесів.

3. **Синхронізація**:

- Метод Barrier() зупиняє виконання кожного процесу, доки всі не досягнуть цієї точки.

4. **Робота з великими масивами**:

- mpi4py інтегрується з бібліотекою **NumPy**, що дозволяє ефективно обробляти великі обсяги даних.

Висновок

У цьому розділі розглянуто основи роботи з потоками в системах із локальною пам’яттю, зокрема стандарт **MPI** та бібліотеку **mpi4py** для Python. Було висвітлено ключові можливості MPI, зокрема ідентифікацію процесів, обмін повідомленнями, синхронізацію та інтеграцію з великими масивами даних. Основна увага приділена перевагам стандарту: масштабованості, гнучкості та ефективності, які роблять його універсальним інструментом для реалізації паралельних обчислень. mpi4py поєднує простоту Python із потужністю MPI, забезпечуючи зручне середовище для розробки паралельних програм.

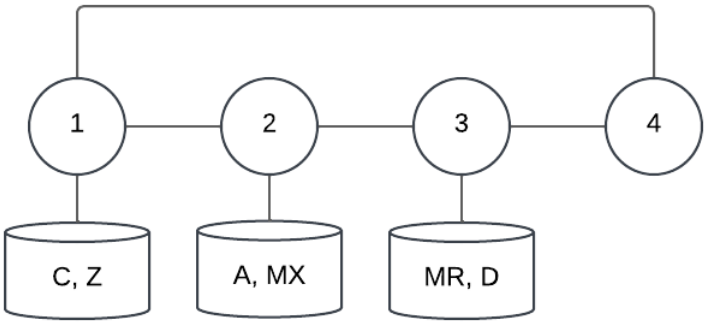
РОЗДІЛ 2 РОЗРОБКА ПРОГРАМИ ДЛЯ ПКС ЛП

2.1 Аналіз структури ПКС ЛП

Визначаємо варіант за номером студента у списку групи, по таблиці наданій викладачем. Отримуємо варіант 2:

Варіант	Задача	Введення - виведення	Структура
2	A= min(C)*Z + D*(MX*MR)	1-C, Z 2- A, MX P-MR,D	Структура 2

По варіанту маємо кільцеву структуру на 4 процесори ПКС ЛП та 4 ПВВ.



2.2 Розробка паралельного математичного алгоритму

1. ai = min(Ch), i = 1…4
2. a = min(a, ai)
3. Ah = a * Z + D * (MX * MRh)

N - розмір масивів

P = 4 - кількість процесорів ПКС ЛП

h = N/P

2.3 Розробка алгоритмів потоків

Задача T1

1. Ввід C, Z
2. Передача C, Z до T4
3. Передача C, Z до T2
4. Отримання MX, MR, D від T2
5. Обчислення 1: a1 = min(Ch)
6. Передача a1 до T2
7. Отримання a від T2
8. Обчислення 2: A1 = a * Zh + D * (MX * MRh)
9. Передача A1 до T2

Задача T2

1. Ввід MX
2. Отримання C, Z від T1
3. Отримання MR, D від T3
4. Передача C, Z, MX до T3
5. Передача MX, MR, D до T1
6. Обчислення 1: a2 = min(Ch)
7. Отримання a1 від T1
8. Передача a1, a2 до T3
9. Отримання a від T3
10. Передача a до T1
11. Обчислення 2: A2 = a * Zh + D * (MX * MRh)
12. Отримання A1 від T1
13. Отримання A3, A4 від T3
14. Вивести A

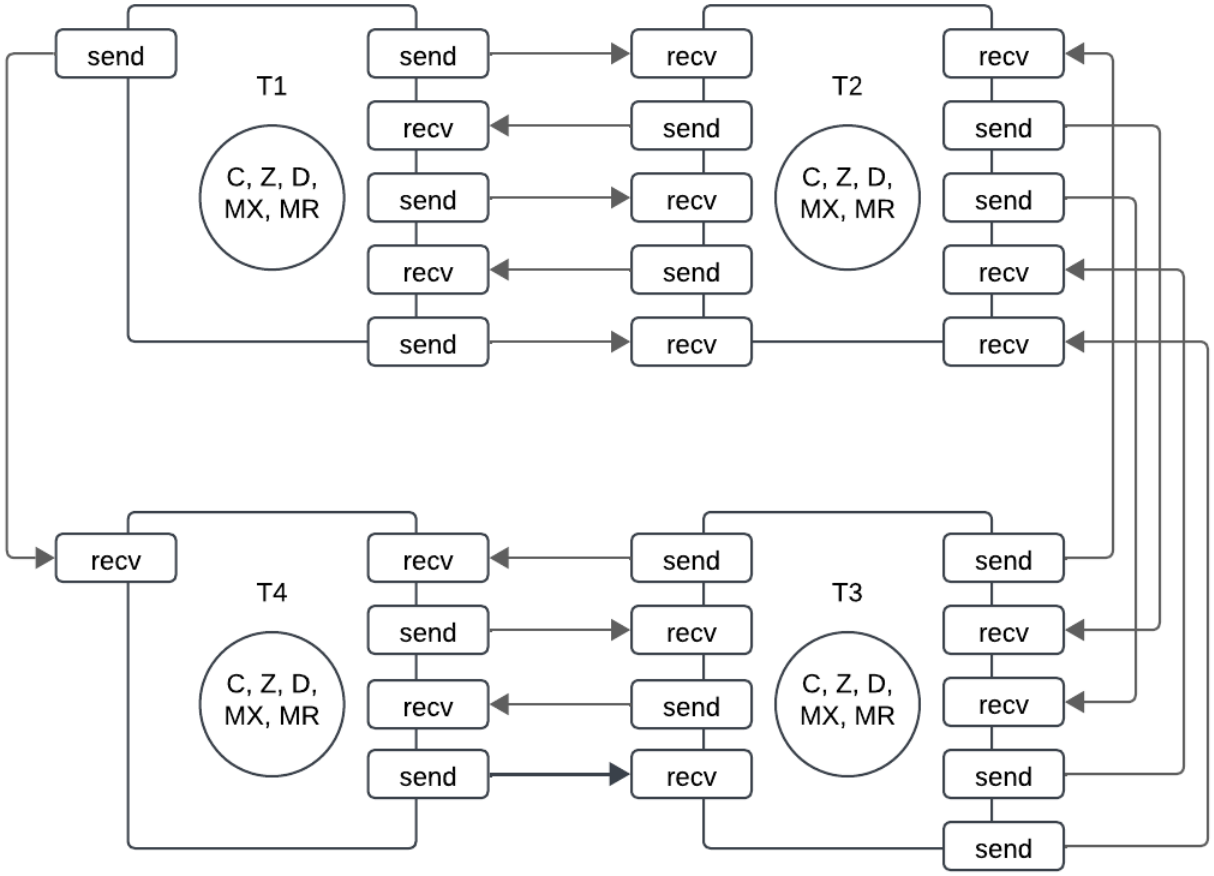
Задача T3

1. Ввід MR, D
2. Передача MR, D до T2
3. Отримання C, Z, MX від T2
4. Передача MX, MR, D до T4
5. Обчислення 1: a3 = min(Ch)
6. Отримання a1, a2 від T2
7. Отримання a4 від T4
8. Обчислення 1.1: min(a, ai)
9. Передача a до T2
10. Передача a до T4
11. Обчислення 2: A3 = a * Zh + D * (MX * MRh)
12. Отримання A4 від T4
13. Передача A3, A4 до T2

Задача T4

1. Отримання C, Z від T1
2. Отримання MX, MR, D від T3
3. Обчислення 1: a4 = min(Ch)
4. Передача a4 до T3
5. Отримання a від T3
6. Обчислення 2: A4 = a * Zh + D * (MX * MRh)
7. Передача A4 до T3

2.4 Розробка схеми взаємодії потоків



Оскільки маємо просто кільцеву структуру без особливих корисних можливостей та потреб до використання інших інструментів mpi4py, то обмежуємося методами Send() і Recv().

Висновки

У межах роботи було виконано розробку паралельної програми з кульцевою структурою, що передбачало створення загального алгоритму та алгоритмів для кожного процесора. На основі отриманих результатів вдалося побудувати ефективну схему взаємодії потоків, яка забезпечує узгоджену роботу програми.

РОЗДІЛ 3 РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМИ

3.1 Розробка і опис програми

Під час розробки було використано такі бібліотеки:

- **mpi4py** — для реалізації функціоналу MPI у Python,
- **numpy** — для математичних обчислень із масивами,
- **time** — для вимірювання часу виконання програми.

На початку роботи час старту записується у змінну start_time:
start_time = time.time()

Далі відбувається ініціалізація змінних для роботи з MPI, а також визначення кількості процесорів та розмірів масивів:

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()

N = 1000
P = 4
```

Потоки за допомогою конструкції match-case генерують локальні змінні зі своїх початкових даних та передають їх сусідам.
Перше обчислення виконується так:

```
# min(Ch)
Ch = np.array_split(C, P)
Ch = Ch[rank]
ai = min(Ch)
```

Локальні результати кожен потік передає у вузол **T3**, який обчислює значення a = min(a, ai) та розсилає фінальний результат усім потокам.
Далі кожен потік ділить несумісні масиви на частини для локальної обробки:

```
# Ah = a * Zh + D * (MX * MRh)
# Getting Zh
Zh = np.array_split(Z, P)
Zh = Zh[rank]
# Getting MRh
MRh = np.hsplit(MR, P)
MRh = MRh[rank]
```

```
# Getting the result - Ah
Ah = np.add(np.dot(a, Zh),
            np.dot(D, np.dot(MX, MRh)))
```

Після завершення обчислень потоки надсилають свої результати у вузол **T2**, який виводить підсумкове значення та час виконання обчислень.

3.2 Тестування програми

Перше тестування для N=4:

```
PS C:\Users\bezsh\Desktop\For_studying\5_semester\Паралельне програмування\РГР> mpiexec -np 4 python main.py

Length of output: 4
Output: [17 17 17 17]
Execution time:  0.005507230758666992

PS C:\Users\bezsh\Desktop\For_studying\5_semester\Паралельне програмування\РГР>
```

Перевірка результату:

A= min(C)*Z + D*(MX*MR)

$$A = \min([1,1,1,1]) * Z + [1,1,1,1] * \left(\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix} * \begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}\right) =$$

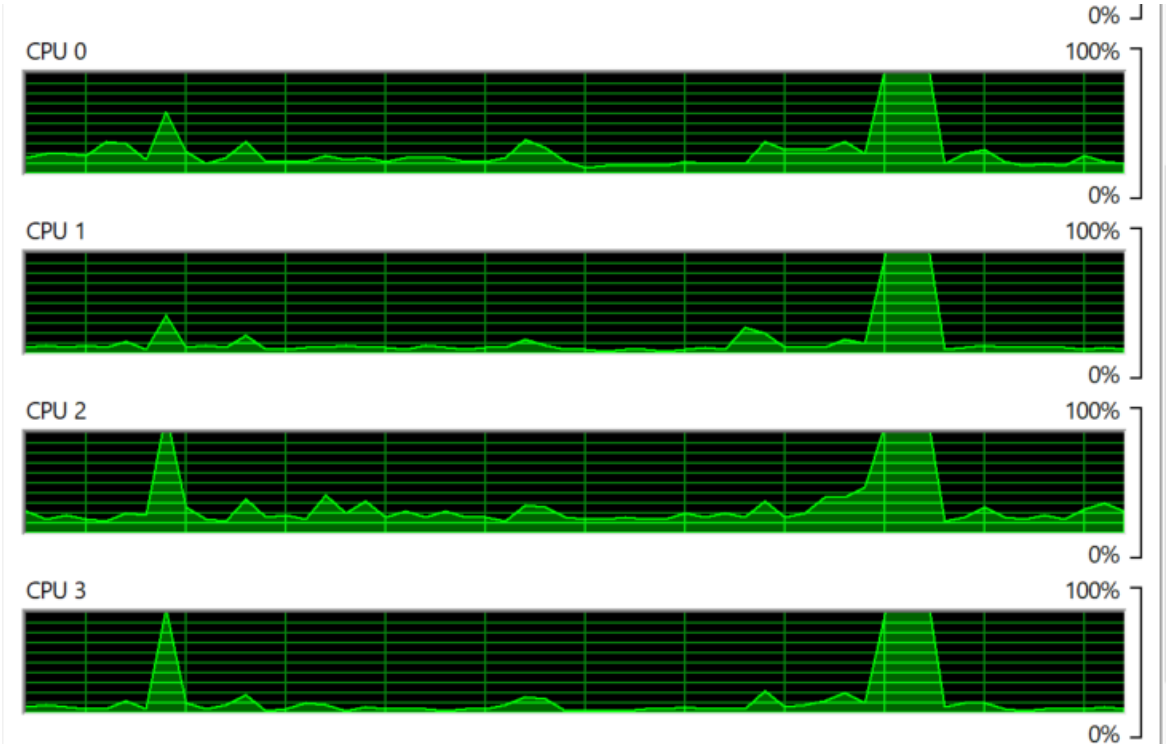
$$[1,1,1,1] + [1,1,1,1] * \left(\begin{bmatrix}4&4&4&4\\4&4&4&4\\4&4&4&4\\4&4&4&4\end{bmatrix}\right) = [1,1,1,1] + [16,16,16,16] =$$

[17,17,17,17]

Відповіді співпадають, отже програма працює коректно.

Тестуємо швидкодію з параметром N=1000:

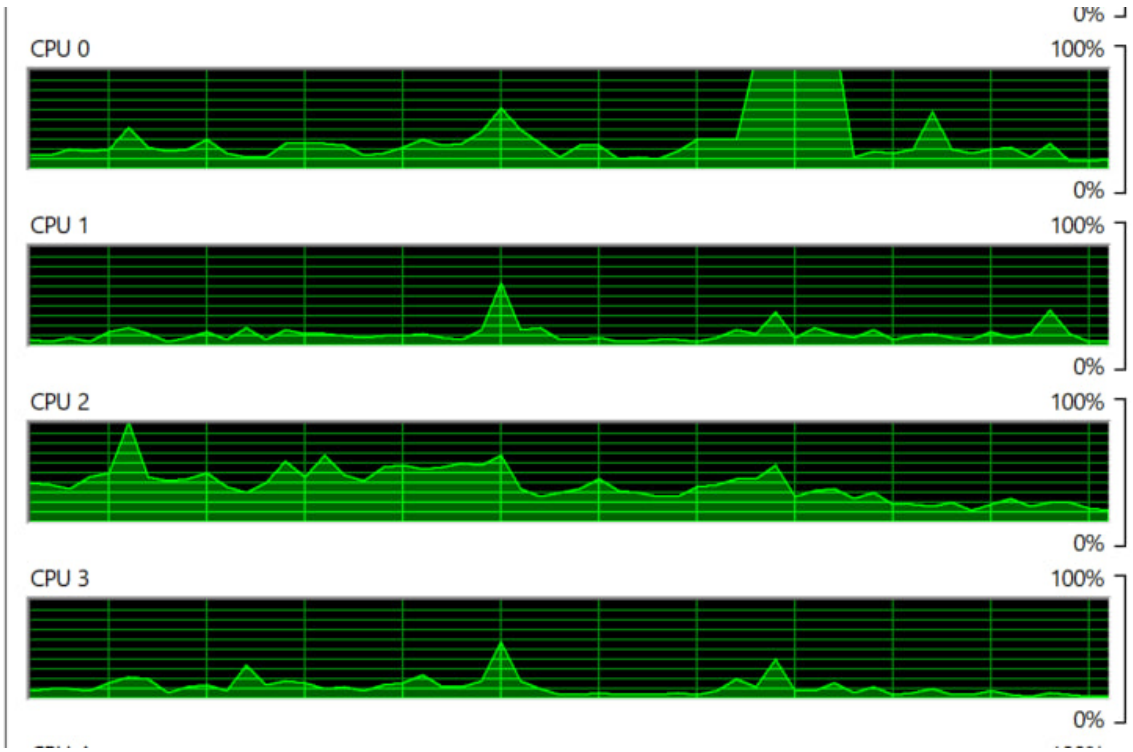
При 4 ядрах:




```
PS C:\Users\bezsh\Desktop\For_studying\5_semester\Паралельне програмування\РГР> mpiexec -np 4 python main.py

Length of output: 1000
Output: [1000001, 1000001, 1000001 ... 1000001]
Execution time:  1.126145601272583
```

При одному ядрі:



```
PS C:\Users\bezsh\Desktop\For_studying\5_semester\Паралельне програмування\РГР> mpiexec -np 4 python main.py

Length of output: 1000
Output: [1000001, 1000001, 1000001 ... 1000001]
Execution time:  1.7320051193237305
```

Розраховуємо коефіцієнт прискорення:
1.732 / 1.1261 = 1.538

Висновок

Була розроблена програма на мові програмування Python, з використанням бібліотек для взаємодії з numpy масивами, MPI та визначення часу виконання роботи програми: numpy, mpi4py, time. Ця програма обчислювала математичну функцію, подану за варіантом по кільцевій структурі та протестована для різних розмірів масиву та декількох варіацій кількості робочих процесорів, тобто для N=4, кількість ядр – 4, для N=1000, кількість ядр – 4, для N=1000, кількість ядр – 1. При подальших розрахунках отримали коефіцієнт прискорення = 1.538

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ

У процесі виконання роботи було досліджено принципи організації паралельних обчислень із використанням бібліотеки MPI та її реалізації mpi4py у середовищі Python. Розглянуто основні можливості mpi4py, зокрема обмін даними між процесами, виконання колективних операцій, синхронізацію та обробку масивів.

На основі аналізу поставленої задачі та архітектури кільцевої системи ПКС ЛП було створено паралельний алгоритм для розв'язання математичної задачі. Для кожного процесора було розроблено індивідуальний алгоритм, а також побудовано схему їх взаємодії.

Реалізована програма використовує mpi4py для управління потоками та бібліотеку NumPy для виконання математичних операцій. Тестування на масивах розміру N=4 підтвердило правильність обчислень. Під час аналізу швидкодії з великими масивами (N=1000) на 1 і 4 ядрах було визначено коефіцієнт прискорення, що склав 1.538.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жуков І., Корочкін О. Паралельні та розподілені обчислення. Навч. посібник – К.:Корнійчук, 2015. - с. 240.

2. Програмне забезпечення високопродуктивних комп'ютерних систем. Конспект лекцій: Навч. посібник для здобувачів ступеня бакалавр / Корочкін О.В., Русанова О.В. , Кулаков Ю.О.– Київ : КПІ ім. Ігоря Сікорського, 2024. – 257 с. Електронний ресурс. Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол No 8 від 19.02.2024 р.)
https://ela.kpi.ua/handle/123456789/66519

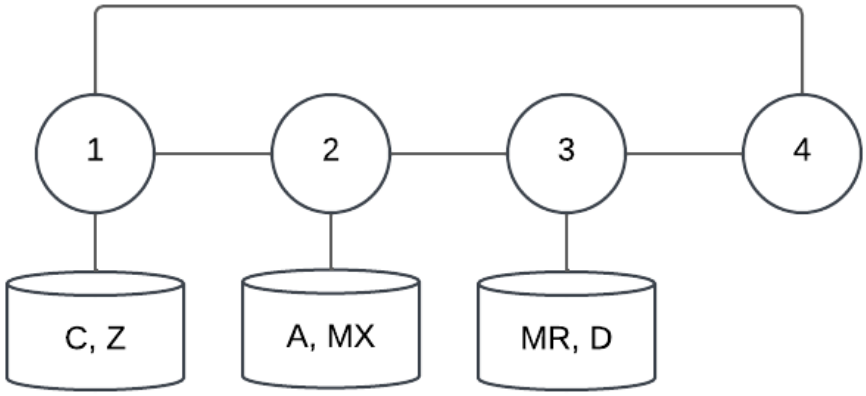
3. Офіційна документація mpi4py
https://mpi4py.readthedocs.io/en/stable/#

4. Офіційна документація NumPy
https://numpy.org/doc/stable/

5. Сторінка Вікіпедії з деякими визначеннями про Архітектуру паралельних обчислювальних систем
https://uk.wikipedia.org/wiki/Архітектура_паралельних_обчислювальних_систем

6. Навчальний посібник "Паралельні та розподілені обчислення"
https://dspace.kntu.kr.ua/server/api/core/bitstreams/396e02d2-725b-47b5-a1c0-ae07a9bec326/content

ДОДАТОК А



ДОДАТОК Б

```
# ІО-21 Безщасний Роман
# РГР, Варіант №2
# A = min(C) * Z + D * (MX * MR)

from mpi4py import MPI
import numpy as np
import time

# Start timer
start_time = time.time()

# MPI initialization
comm = MPI.COMM_WORLD
rank = comm.Get_rank()  # Thread number
N = 1000  # Array size
P = 4  # Number of threads

# Input of all variables
match rank:
    case 0:
        C = np.ones(N, dtype=int)
        Z = np.ones(N, dtype=int)

        comm.send(C, dest=1, tag=1)
        comm.send(Z, dest=1, tag=2)
        comm.send(C, dest=3, tag=1)
        comm.send(Z, dest=3, tag=2)

        MX = comm.recv(source=1, tag=3)
        MR = comm.recv(source=1, tag=4)
        D = comm.recv(source=1, tag=5)

    case 1:
        MX = np.ones((N, N), dtype=int)
```

```
        C = comm.recv(source=0, tag=1)
        Z = comm.recv(source=0, tag=2)

        MR = comm.recv(source=2, tag=4)
        D = comm.recv(source=2, tag=5)

        comm.send(C, dest=2, tag=1)
        comm.send(Z, dest=2, tag=2)
        comm.send(MX, dest=2, tag=3)

        comm.send(MX, dest=0, tag=3)
        comm.send(MR, dest=0, tag=4)
        comm.send(D, dest=0, tag=5)

    case 2:
        MR = np.ones((N, N), dtype=int)
        D = np.ones(N, dtype=int)

        comm.send(MR, dest=1, tag=4)
        comm.send(D, dest=1, tag=5)

        C = comm.recv(source=1, tag=1)
        Z = comm.recv(source=1, tag=2)
        MX = comm.recv(source=1, tag=3)

        comm.send(MX, dest=3, tag=3)
        comm.send(MR, dest=3, tag=4)
        comm.send(D, dest=3, tag=5)

    case 3:
        C = comm.recv(source=0, tag=1)
        Z = comm.recv(source=0, tag=2)
        MX = comm.recv(source=2, tag=3)
        MR = comm.recv(source=2, tag=4)
        D = comm.recv(source=2, tag=5)

# min(Ch)
Ch = np.array_split(C, P)
Ch = Ch[rank]
ai = min(Ch)

# min(a, ai)
match rank:
    case 0:
        comm.send(ai, dest=1, tag=7)

        a = comm.recv(source=1, tag=6)

    case 1:
        a1 = comm.recv(source=0, tag=7)

        comm.send(a1, dest=2, tag=7)
        comm.send(ai, dest=2, tag=8)

        a = comm.recv(source=2, tag=6)
```

```
        comm.send(a, dest=0, tag=6)

    case 2:
        a1 = comm.recv(source=1, tag=7)
        a2 = comm.recv(source=1, tag=8)
        a4 = comm.recv(source=3, tag=9)

        a = min(ai, a1, a2, a4)

        comm.send(a, dest=1, tag=6)
        comm.send(a, dest=3, tag=6)

    case 3:
        comm.send(ai, dest=2, tag=9)

        a = comm.recv(source=2, tag=6)

# Ah = a * Zh + D * (MX * MRh)
# Getting Zh
Zh = np.array_split(Z, P)
Zh = Zh[rank]
# Getting MRh
MRh = np.hsplit(MR, P)
MRh = MRh[rank]
# Getting the result - Ah
Ah = np.add(np.dot(a, Zh),
            np.dot(D, np.dot(MX, MRh)))

# Sending results to T2 for outputting results
match rank:
    case 0:
        comm.send(Ah, dest=1, tag=10)

    case 1:
        A1 = comm.recv(source=0, tag=10)
        A3 = comm.recv(source=2, tag=11)
        A4 = comm.recv(source=2, tag=12)
        temp = np.array([A1, Ah, A3, A4])
        res = temp.flatten()
        print('\nLength of output: ' + str(len(res)))
        if len(res) == 4:
            print(f'Output: {res}')
        else:
            print(f'Output: [{res[0]}, {res[1]}, {res[2]} ... {res[-1]}]')
        print(f'Execution time: ', (time.time() - start_time), '\n')

    case 2:
        A4 = comm.recv(source=3, tag=12)

        comm.send(Ah, dest=1, tag=11)
        comm.send(A4, dest=1, tag=12)

    case 3:
        comm.send(Ah, dest=2, tag=12)
```