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»

Тишнюк І. О.

(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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*Тишнюк Івану Олеговичу*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(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

7. Дата видачі завдання \_\_\_\_\_\_\_\_\_21 11 2024\_\_\_\_\_\_\_\_\_\_\_.

**ЗМІСТ**

ВСТУП……………………………………………………………………………… 4

РОЗДІЛ 1. ОГЛЯД БІБЛІОТЕКИ MPI (МОВИ ПРОГРАМУВАННЯ ADA)........5

1.1 Засоби програмування потоків ………………………………………. 5

1.2 Засоби організації взаємодії потоків……………………………….….6

Висновки до розділу 1………………………………………………….7

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ДЛЯ ПКСЛП..………………………….... 8

2.1 Аналіз структури ПКС ЛП………………………………………….….8

2.2 Розробка паралельного математичного алгоритму………………..….8

2.3 Розробка алгоритмів потоків………………………………………..…9

2.4 Розробка схеми взаємодії потоків……………………………….…...12

Висновки до розділу 2…………………………………………………12

РОЗДІЛ 3. РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМИ ……………………….... 13

3.1 Розробка і опис програми ……………………………………………..13

3.2 Тестування програми ……………………………………………….…14

Висновки до розділу 3…………………………………………………16

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ …………….…………………………..17

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…………………………………….……18

ДОДАТКИ……………………………………………………..................................19

Додаток А. Структура ПКС ЛП ………………………………………………..19

Додаток Б. Лістинг програми ………………………………….………………19

ВСТУП

Паралельні обчислення відіграють ключову роль у сучасних комп'ютерних системах, де зростаючі вимоги до швидкості обробки даних і продуктивності змушують розробників шукати нові підходи до реалізації алгоритмів. У наш час, коли ресурсоємні задачі, такі як моделювання фізичних процесів, аналіз великих обсягів даних і тренування моделей штучного інтелекту, стають все більш поширеними, ефективне використання обчислювальних ресурсів є критично важливим. Одним із ключових способів досягнення цього є застосування методів паралельного програмування.

Об'єктом даної розрахунково-графічної роботи є створення програмного застосунку, який реалізує паралельні обчислення за допомогою бібліотеки MPI (Message Passing Interface). Бібліотека MPI надає потужні інструменти для організації взаємодії між процесами в системах із локальною пам’яттю. У роботі передбачається визначення математичної задачі, схеми з'єднання процесорів та їхньої взаємодії відповідно до заданого варіанту.

Метою цієї роботи є закріплення теоретичних знань, вмінь та навичок розробки і аналізу паралельних алгоритмів, розробки та налагодження програм для паралельних комп’ютерних систем з локальною пам’яттю (ПКС ЛП), отримання практичних навичок по роботі з паралельними мовами та бібліотеками паралельного програмування (MPI) при використанні моделі взаємодії потоків через повідомлення

**РОЗДІЛ 1 ОГЛЯД БІБЛІОТЕКИ MPI**

1.1 Засоби програмування потоків

У сучасних комп'ютерних системах із локальною пам'яттю організація взаємодії між процесорами є ключовим завданням для досягнення високої продуктивності. Потоки в таких системах працюють незалежно один від одного, що вимагає спеціальних підходів для ефективного розподілу задач і забезпечення їх синхронізації. Одним із найбільш популярних інструментів для вирішення цих завдань є бібліотека MPI (Message Passing Interface) — стандарт взаємодії процесів у паралельних програмах [ ] . Посилання ?

MPI — це стандарт програмного забезпечення, розроблений у 1993–1994 роках групою MPI Forum. Він був створений для забезпечення ефективної комунікації між процесорами у паралельних системах. Головною метою MPI є організація передачі повідомлень між потоками, що виконуються на різних обчислювальних вузлах. Цей стандарт не містить засобів для реалізації паралельного введення-виведення чи налагодження програм, однак це компенсується його високою гнучкістю та універсальністю.

MPI підтримує широкий спектр функцій для роботи з потоками, серед яких розподіл задач, обмін повідомленнями, синхронізація процесів і управління ресурсами. Завдяки цьому MPI став одним із найважливіших інструментів для реалізації паралельних обчислень як на обчислювальних кластерах, так і на суперкомп'ютерах.

#### Основні переваги MPI:

1. **Масштабованість**: MPI дозволяє створювати програми, що ефективно працюють на системах із десятками тисяч процесорів.
2. **Ефективність**: Використання MPI забезпечує оптимальний розподіл задач між процесорами, що знижує загальний час виконання обчислень.
3. **Гнучкість**: Стандарт підтримує різні схеми обміну даними, що дозволяє адаптувати алгоритми під специфічні вимоги задачі.
4. **Універсальність**: Існують реалізації MPI для багатьох мов програмування, зокрема C, C++, Python (mpi4py), Java (MPJ Express), що спрощує інтеграцію в різні проєкти.

1.2 Засоби організації взаємодії потоків

Для реалізації паралельних обчислень у рамках даної роботи було обрано мову програмування Python і бібліотеку mpi4py. Такий вибір обумовлений кількома важливими факторами. Python це сучасна та зручна для роботи мова програмування. Велика кількість бібліотек дозволяє простіше реалізовувати математичні операції для нашого алгоритму. Наявність бібліотеки mpi4py яка цілком відповідає стандартам MPI дозволяє легко реалізувати алгоритм наданий варіантом

#### Огляд можливостей mpi4py

Бібліотека **mpi4py** забезпечує повну функціональність стандарту MPI і дозволяє організовувати взаємодію потоків у паралельних програмах. Основними можливостями бібліотеки є:

1. **Призначення ідентифікаторів**
   * mpi4py дозволяє отримати унікальний ідентифікатор (rank) кожного процесу за допомогою методу Get\_rank().
   * Загальна кількість процесів у комунікаторі визначається за допомогою методу Get\_size().
2. **Обмін повідомленнями**
   * **Точковий обмін**:
     + Send(data, dest): передає дані у вказаний процес.
     + Recv(source): отримує дані від конкретного процесу.
   * **Колективні операції**:
     + bcast(data, root): передає дані від одного процесу до всіх інших.
     + scatter(data, root): розподіляє частини масиву даних між усіма процесами.
     + gather(data, root): збирає дані з усіх процесів у заданий кореневий процес.
     + reduce(data, op, root): виконує операцію (наприклад, суму) над даними всіх процесів і передає результат кореневому процесу.
3. **Синхронізація**
   * Для синхронізації процесів використовується метод Barrier(), який змушує всі процеси дочекатися один одного перед продовженням виконання.
4. **Підтримка роботи з масивами**
   * mpi4py інтегрується з бібліотекою NumPy, що дозволяє ефективно працювати з великими масивами даних.

Висновок

В цьому розділі було розглянуто засоби організації паралельних обчислень та засоби їх програмування за допомогою бібліотеки MPI та її реалізацію mpi4py на мові програмування Python. Вказано причини вибори саме мови програмування Python. Також було розглянуто основні можливості бібліотеки mpi4py

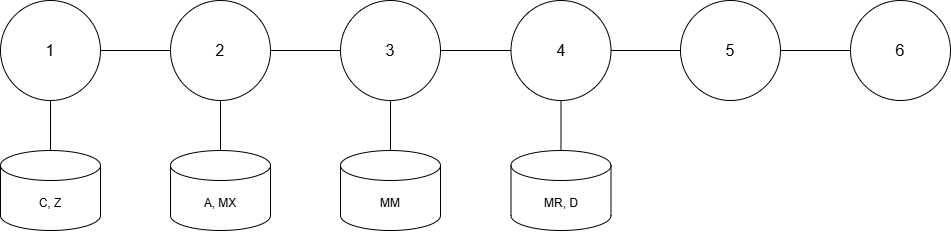
**РОЗДІЛ 2 РОЗРОБКА ПРОГРАМИ ДЛЯ ПКС ЛП**

2.1 Аналіз структури ПКС ЛП

Визначаємо варіант за номером студента у списку групи, по таблиці наданій викладачем. Отримуємо варіант 26:

| Варіант | Задача | Введення - виведення | Структура |
| --- | --- | --- | --- |
| 26 | A=D \* (MX \* MR) - min(C) \* (Z \* MM) | 1 -C, Z; 2 - A, MX; 3 - MM; 4 -MR,D; | Структура 1 |

По варіанту маємо лінійну структуру на 6 процесорів ПКС ЛП та 4 ПВВ.



2.2 Розробка паралельного математичного алгоритму

1. ai = min(C), i = 1…6

2. a = max(a, ai)

3. Ah = D \* (MX \* MRh) - a \* (Z \* MMh)

N - розмір масивів

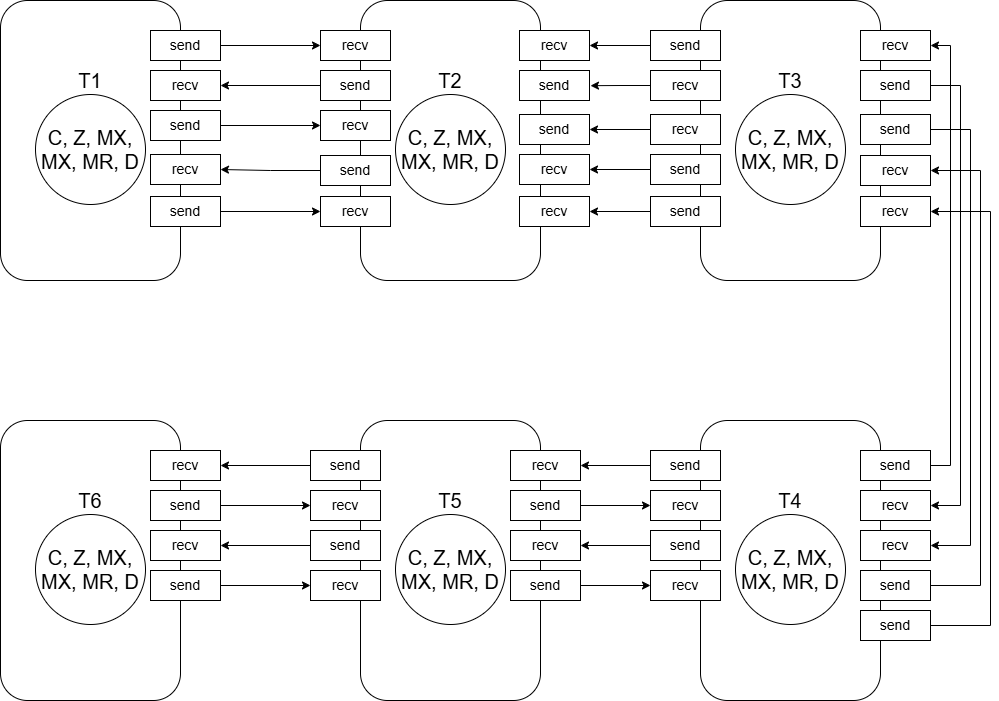
P = 6 - кількість процесорів ПКС ЛП

h = N/P

2.3 Розробка алгоритмів потоків ВСЕ ПЕРЕДАЄТЬСЯ ЦІЛКОМ?

| Задача Т1   1. Ввід C, Z С передається частинами ? 2. Передача C, Z до T2 3. Отримання MX, MM, MR, D від Т2 4. Обчислення 1: a1 = min(C) 5. Передача а1 до Т2 6. Отримання а від Т2 7. Обчислення 2: A1 = D \* (MX \* MR1) - a \* (Z \* MM1) 8. Передача A1 до Т2 |
| --- |
| Задача Т2   1. Ввід MX 2. Отримання C, Z від T1 3. Отримання MM, MR, D від Т3 4. Передача C, Z, MX до Т3 5. Передача MX, MM, MR, D до Т1 6. Обчислення 1: a2 = min(C) 7. Отримання а від Т1 8. Передача а1, а2 до Т3 9. Отримання а від Т3 10. Передача а до Т1 11. Обчислення 2: A2 = D \* (MX \* MR2) - a \* (Z \* MM2) 12. Отримання Ah від Т1, Т3 13. Вивести А |
| Задача Т3   1. Ввід MM 2. Отримання MR, D від Т4 3. Передача MM, MR, D до Т2 4. Отримання C, Z, MX від Т2 5. Передача C, Z, MX, MM до Т4 6. Обчислення 1: a3 = min(C) 7. Отримання а1, а2 від Т2 8. Передача а1, а2, а3 до Т4 9. Отримання а від Т4 10. Передача а до Т2 11. Обчислення 2: A3 = D \* (MX \* MR3) - a \* (Z \* MM3) 12. Отримання А4, А5, А6 від Т4 13. Передача А3, А4, А5, А6 до Т2 |
| Задача Т4   1. Ввід MR, D 2. Передача MR, D до Т3 3. Отримання C, Z, MX, MM від Т3 4. Передача C, Z, MX, MX, MR, D до Т5 5. Обчислення 1: a4 = min(C) 6. Отримання а1, а2, а3 від Т3 7. Отримання а5, а6 від Т5 8. Обчислення 1.1: min(a, ai) 9. Передача а до Т3, Т5 10. Обчислення 2: A4 = D \* (MX \* MR4) - a \* (Z \* MM4) 11. Отримання А5, А6 від Т5 12. Передача А4, А5, А6 до Т3 |
| Задача Т5   1. Отримання C, Z, MX, MX, MR, D від Т4 2. Передача C, Z, MX, MX, MR, D до Т6 3. Обчислення 1: a5 = min(C) 4. Отримання а6 від Т6 5. Передача а5, а6 до Т4 6. Отримання а від Т4 7. Передача а до Т6 8. Обчислення 2: A5 = D \* (MX \* MR5) - a \* (Z \* MM5) 9. Отримання А6 від Т6 10. Передача А5, А6 до Т4 |
| Задача Т6   1. Отримання C, Z, MX, MX, MR, D від Т5 2. Обчислення 1: a6 = min(C) 3. Передача а6 до Т5 4. Отримання а від Т5 5. Обчислення 2: A6 = D \* (MX \* MR6) - a \* (Z \* MM6) 6. Передача А6 до Т5 |

2.4 Розробка схеми взаємодії потоків Опис? Обсяг повідомлення?



Через те що варіанту маємо лінійну структуру нажаль не вийде користуватися колективними операціями тому в роботі обмежуємося лише лише Receive та Send

Висновки

Було отримано варіант роботи який передбачає створення паралельної програми по лінійній структурі. Для математичної задачі розроблено загальний алгоритм та алгоритм кожного процесора. На основі розроблених алгоритмів було побудовано структуру взаємодії потоків під час роботи

РОЗДІЛ 3 РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМИ

3.1 Розробка і опис програми

Під час розробки використовувалися бібліотеки:

* mpi4py - для реалізації можливостей MPI на мові програмування Python
* numpy - для проведення математичних обчислень з масивами
* time - для замірів часу який програма витрачає на виконання

На початку роботи записуємо час початку виконання у змінну start:

start = time.time()

Після цього ініціалізуємо змінні для роботи з MPI та задаємо змінні які визначають кількість процесорів та розміри масивів

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

N = 1002

P = 6

Далі потоки за допомогою структури match case генерують змінні зі своїх ПВВ та передають ці змінні усім сусідам.

Після цього проводиться перше обчислення

Ch = np.array\_split(C, P)

Ch = Ch[rank]

ai = min(Ch)

Свої локальні результати потоки передають по ланцюгу в T4 який рахує a = min(a, ai) і після цього передає фінальний результат а до інших потоків

Потім потоки ділять масиви які не є спільними змінними на окремі частини і рахують свою частину фінального результату

MRh = np.hsplit(MR, P)

MRh = MRh[rank]

MMh = np.hsplit(MM, P)

MMh = MMh[rank]

Ah = np.subtract(

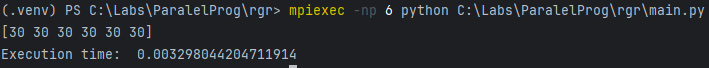
np.dot(D, np.dot(MX, MRh)),

np.dot(a, np.dot(Z, MMh)))

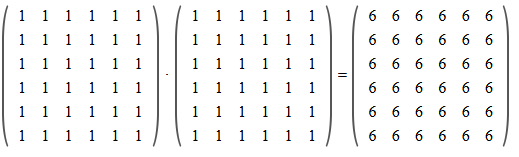
Після завершення обчислень потоки передають результати до T2 який виводить фінальний результат і час який знадобився на обчислення

3.2 Тестування програми

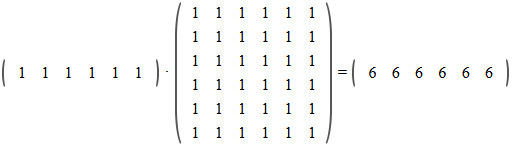
Перше тестування для N=6:



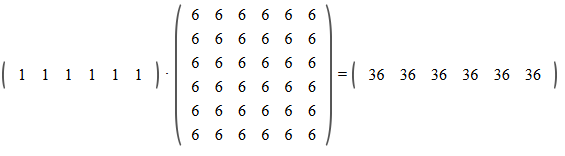
Перевірка результату:

MX \* MR:  


Z \* MM:



D \* (MX \* MR):



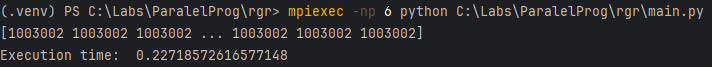
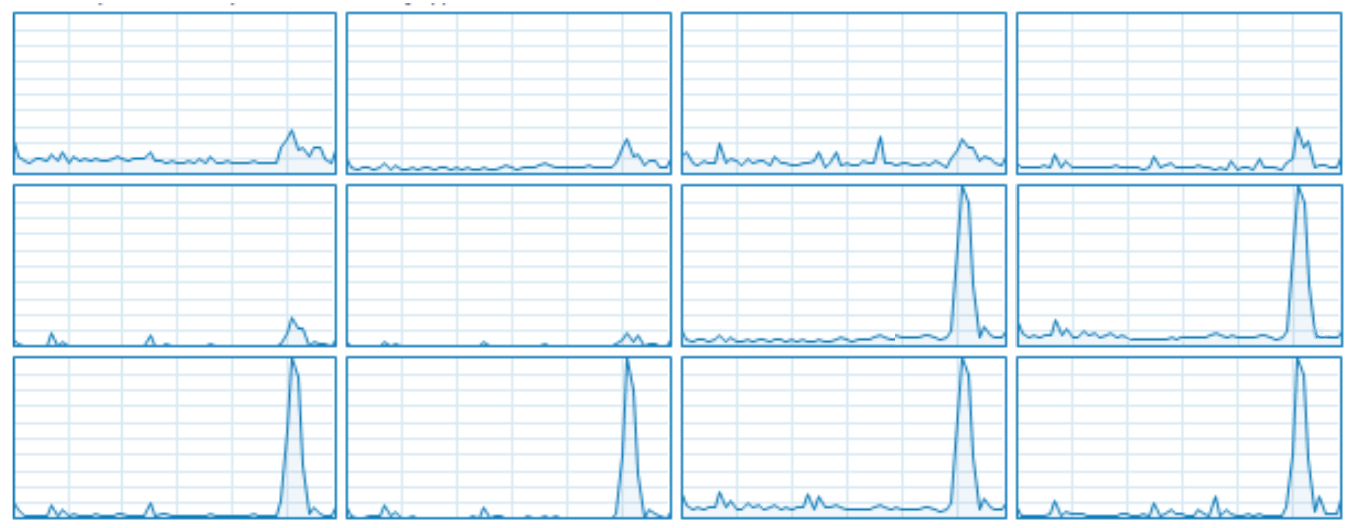
D \* (MX \* MR) - min(C) \* (Z \* MM):



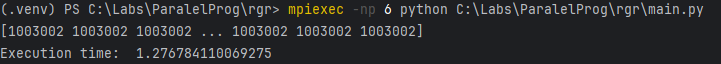
Відповідь зійшлася, отже програма працює коректно

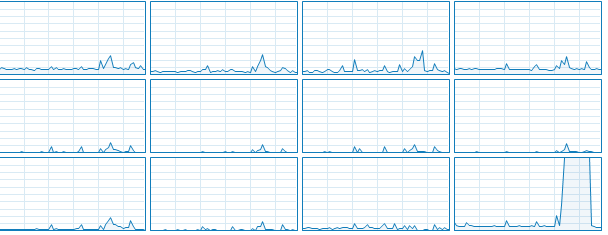
Тестуємо швидкодію з параметром N=1002:

При P=6:



При P=1





Розраховуємо коефіцієнт прискорення:

1.2768/0.2272 = 5.6

Висновок

Було розроблено програму на мові програмування Python яка паралельно обчислює математичну задачу по варіанту по лінійній структурі на 6 процесорах. Результати виконання при N=6 показали правильний результат. Після проведення тестування програми на 1 та 6 ядрах було вирахувано коефіцієнт прискорення 5.6

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ

У ході виконання роботи було розглянуто основи організації паралельних обчислень із використанням бібліотеки MPI та її реалізації mpi4py у мові програмування Python. Було розглянуто основні можливості бібліотеки mpi4py, такі як обмін даних між процесорами, колективні операції, синхронізація процесів і робота з масивами.

На основі аналізу задачі та структури лінійної системи ПКС ЛП розроблено паралельний алгоритм розв'язання математичної задачі. Для кожного процесора було створено окремий алгоритм. На основі аналізу та розробленого алгоритму було побудовано схему взаємодії процесорів

Було реалізовано програму, яка використовує mpi4py для організації взаємодії між потоками та бібліотеку NumPy для математичних обчислень з масивами. Протестувавши програму на масивах розміру N=6 було отримано результат обчислень правильність якого підтвердилася при окремій перевірці. Після тестування швидкодії з великими масивами N=1002 на 1 та 6 ядрах, було обчислено коефіцієнт прискорення 5.6

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ де посилання в тексті

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>

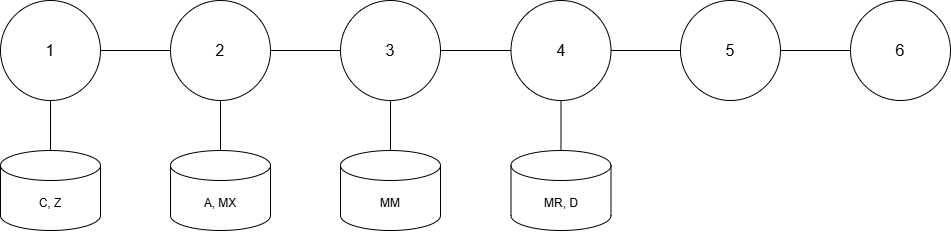
3. Офіційна документація mpi4py

<https://mpi4py.readthedocs.io/en/stable/#>

4. Офіційна документація NumPy

<https://numpy.org/doc/stable/>

ДОДАТОК А



ДОДАТОК Б

from mpi4py import MPI

import numpy as np

import time

# Start timer

start = time.time()

# MPI initialization

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

N = 1002 # Array size

P = 6 # 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)

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

MM = comm.recv(source=1, tag=4)

MR = comm.recv(source=1, tag=5)

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

case 1:

MX = np.ones((N, N), dtype=int)

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

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

MM = comm.recv(source=2, tag=4)

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

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

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(MM, dest=0, tag=4)

comm.send(MR, dest=0, tag=5)

comm.send(D, dest=0, tag=6)

case 2:

MM = np.ones((N, N), dtype=int)

MR = comm.recv(source=3, tag=5)

D = comm.recv(source=3, tag=6)

comm.send(MM, dest=1, tag=4)

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

comm.send(D, dest=1, tag=6)

C = comm.recv(source=1, tag=1)

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

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

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

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

comm.send(MX, dest=3, tag=3)

comm.send(MM, dest=3, tag=4)

case 3:

MR = np.ones((N, N), dtype=int)

D = np.ones(N, dtype=int)

comm.send(MR, dest=2, tag=5)

comm.send(D, dest=2, tag=6)

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

Z = comm.recv(source=2, tag=2)

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

MM = comm.recv(source=2, tag=4)

comm.send(C, dest=4, tag=1)

comm.send(Z, dest=4, tag=2)

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

comm.send(MM, dest=4, tag=4)

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

comm.send(D, dest=4, tag=6)

case 4:

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

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

MX = comm.recv(source=3, tag=3)

MM = comm.recv(source=3, tag=4)

MR = comm.recv(source=3, tag=5)

D = comm.recv(source=3, tag=6)

comm.send(C, dest=5, tag=1)

comm.send(Z, dest=5, tag=2)

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

comm.send(MM, dest=5, tag=4)

comm.send(MR, dest=5, tag=5)

comm.send(D, dest=5, tag=6)

case 5:

C = comm.recv(source=4, tag=1)

Z = comm.recv(source=4, tag=2)

MX = comm.recv(source=4, tag=3)

MM = comm.recv(source=4, tag=4)

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

D = comm.recv(source=4, tag=6)

# 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=11)

a = comm.recv(source=1, tag=10)

case 1:

a1 = comm.recv(source=0, tag=11)

comm.send(a1, dest=2, tag=11)

comm.send(ai, dest=2, tag=12)

a = comm.recv(source=2, tag=10)

comm.send(a, dest=0, tag=10)

case 2:

a1 = comm.recv(source=1, tag=11)

a2 = comm.recv(source=1, tag=12)

comm.send(a1, dest=3, tag=11)

comm.send(a2, dest=3, tag=12)

comm.send(ai, dest=3, tag=13)

a = comm.recv(source=3, tag=10)

comm.send(a, dest=1, tag=10)

case 3:

a1 = comm.recv(source=2, tag=11)

a2 = comm.recv(source=2, tag=12)

a3 = comm.recv(source=2, tag=13)

a5 = comm.recv(source=4, tag=15)

a6 = comm.recv(source=4, tag=16)

a = min(a1, a2, a3, ai, a5, a6)

comm.send(a, dest=2, tag=10)

comm.send(a, dest=4, tag=10)

case 4:

a6 = comm.recv(source=5, tag=16)

comm.send(ai, dest=3, tag=15)

comm.send(a6, dest=3, tag=16)

a = comm.recv(source=3, tag=10)

comm.send(a, dest=5, tag=10)

case 5:

comm.send(ai, dest=4, tag=16)

a = comm.recv(source=4, tag=10)

# D \* (MX \* MRh) - a \* (Z \* MMh)

MRh = np.hsplit(MR, P)

MRh = MRh[rank]

MMh = np.hsplit(MM, P)

MMh = MMh[rank]

Ah = np.subtract(

np.dot(D, np.dot(MX, MRh)),

np.dot(a, np.dot(Z, MMh)))

# Sending results to T2 for outputting results

match rank:

case 0:

comm.send(Ah, dest=1, tag=21)

case 1:

A1 = comm.recv(source=0, tag=21)

A3 = comm.recv(source=2, tag=23)

A4 = comm.recv(source=2, tag=24)

A5 = comm.recv(source=2, tag=25)

A6 = comm.recv(source=2, tag=26)

temp\_arr = np.array([A1, Ah, A3, A4, A5, A6])

res = temp\_arr.flatten()

print(res)

print("Execution time: ", (time.time()-start))

case 2:

A4 = comm.recv(source=3, tag=24)

A5 = comm.recv(source=3, tag=25)

A6 = comm.recv(source=3, tag=26)

comm.send(Ah, dest=1, tag=23)

comm.send(A4, dest=1, tag=24)

comm.send(A5, dest=1, tag=25)

comm.send(A6, dest=1, tag=26)

case 3:

A5 = comm.recv(source=4, tag=25)

A6 = comm.recv(source=4, tag=26)

comm.send(Ah, dest=2, tag=24)

comm.send(A5, dest=2, tag=25)

comm.send(A6, dest=2, tag=26)

case 4:

A6 = comm.recv(source=5, tag=26)

comm.send(Ah, dest=3, tag=25)

comm.send(A6, dest=3, tag=26)

case 5:

comm.send(Ah, dest=4, tag=26)