

Міністерство освіти і науки України
Національний авіаційний університет
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

Лабораторна робота № 1.5
з дисципліни «Паралельні і розподілені обчислення»
на тему «Обмін повідомленнями. Бібліотека MPI»

Виконав:
студент ФККПІ
групи СП-425
Клокун В. Д.
Перевірив:
Корочкін О. В.

Київ 2019

1. Завдання роботи

Розробити програму для заданої паралельної комп'ютерної системи із локальною пам'яттю (рис. 1).

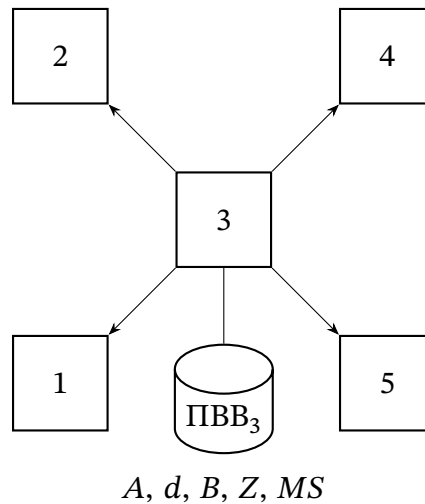


Рис. 1: Задана паралельна комп'ютерна система зі спільною пам'яттю

Програма, розроблена для даної системи, повинна обчислювати значення виразу: $A = d \cdot B + Z \cdot MS$.

2. Хід роботи

2.1. Побудова паралельного алгоритму

Щоб паралельно обчислити значення виразу $A = d \cdot B + Z \cdot MS$, складаємо паралельний алгоритм:

1. $A_H = d \cdot B_H + Z \cdot MS_H$.

Паралельний алгоритм реалізуємо на мові програмування Python з використанням програмного інтерфейсу MPI, реалізованого бібліотекою Microsoft MPI. Щоб отримати доступ до бібліотеки Microsoft MPI з програми, написаною мовою Python, використаємо обгортку mpi4py.

2.2. Розробка алгоритмів потоків

Розробивши паралельний алгоритм, переходимо до розробки алгоритмів задач. Представимо алгоритми у вигляді таблиці (табл. 1).

Табл. 1: Паралельні алгоритми задач

а) $T1$	б) $T2$
Дія	Дія
Отримати d, B, Z, MS від $T3$ Обчислити $A_H = d \cdot B_H + Z \cdot MS_H$ Відправити A_H в $T3$	Отримати d, B, Z, MS від $T3$ Обчислити $A_H = d \cdot B_H + Z \cdot MS_H$ Відправити A_H
в) $T4$	г) $T5$
Дія	Дія
Отримати d, B, Z, MS від $T3$ Обчислити $A_H = d \cdot B_H + Z \cdot MS_H$ Відправити A_H	Отримати d, B, Z, MS від $T3$ Обчислити $A_H = d \cdot B_H + Z \cdot MS_H$ Відправити A_H
д) $T3$	
Дія	
Ввести d, B, Z, MS Відправити d, B, Z, MS в $T1, T2, T4, T5$ Отримати A_H від $T1, T2, T4, T5$ Обчислити $A_H = d \cdot B_H + Z \cdot MS_H$ Вивести A	

2.3. Розробка структурної схеми взаємодії задач

Після розробки алгоритмів потоків, розроблюємо структурну схему взаємодії задач (рис. 2).

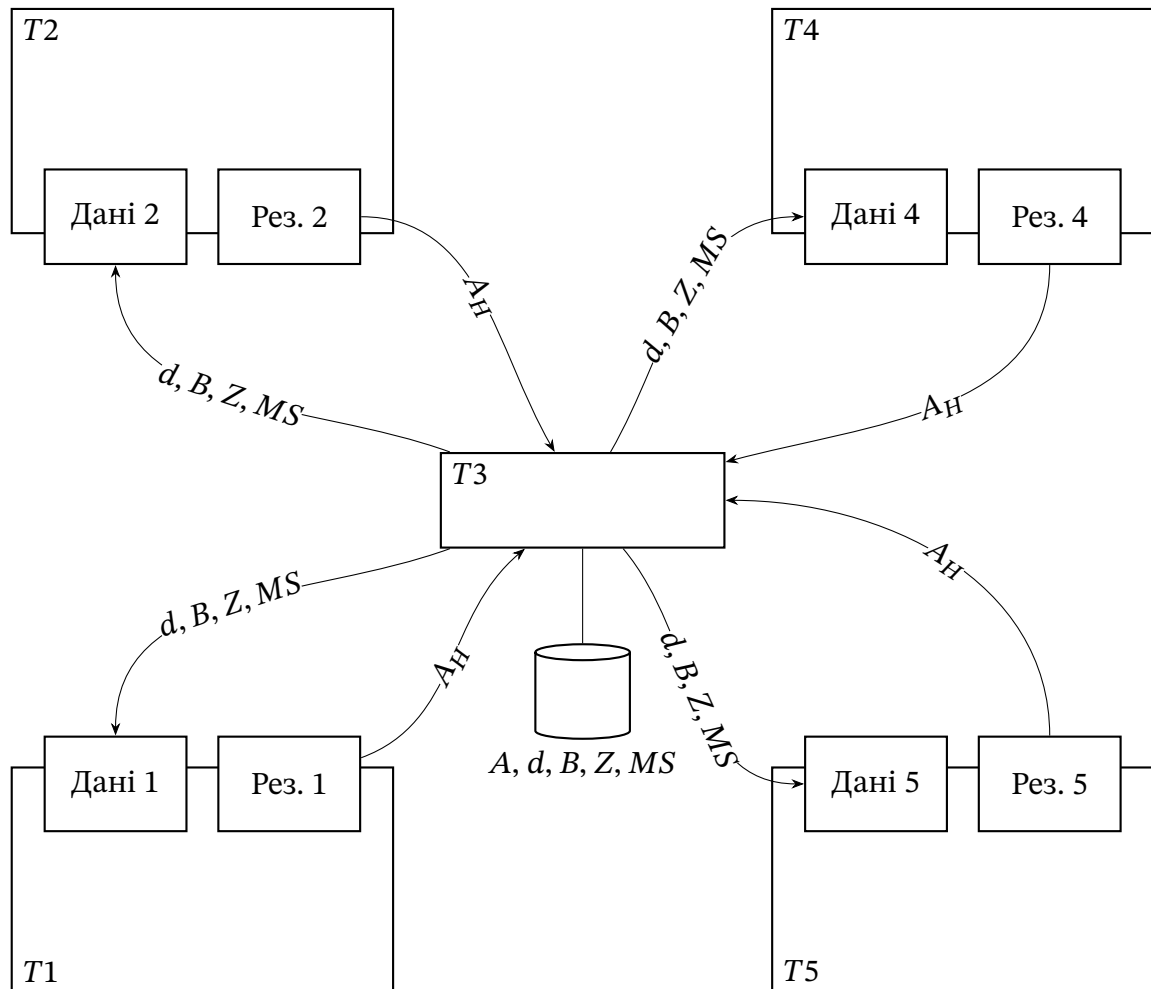
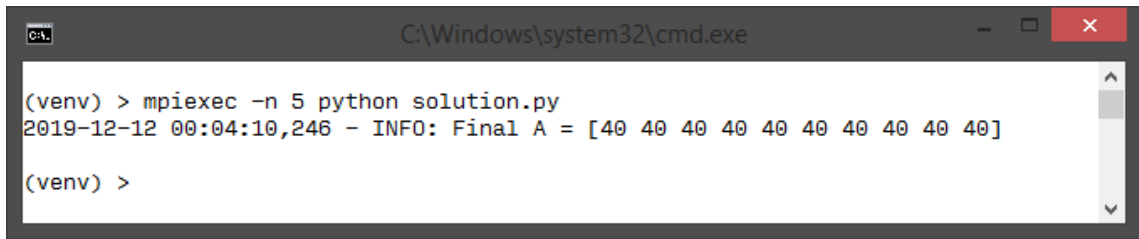


Рис. 2: Структурна схема взаємодії задач

2.4. Розробка програми

Коли структурна схема розроблена, створюємо програму на мові програмування Python (лістинг А.1). Для обміну повідомленнями використовуємо функції, які надає пакет `mpi4py`. Після розробки програми запускаємо її на виконання і спостерігаємо результат (рис. 3).

Як видно, програма коректно обчислює значення заданого виразу з вхідними значеннями, заданими у програмі.

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\system32\cmd.exe". The command prompt shows the following text:

```
(venv) > mpiexec -n 5 python solution.py
2019-12-12 00:04:10,246 - INFO: Final A = [40 40 40 40 40 40 40 40 40 40]

(venv) >
```

Рис. 3: Результат виконання розробленої програми

3. Висновок

Виконуючи дану лабораторну роботу, ми розробили програму для заданої паралельної комп'ютерної системи із локальною пам'яттю, ознайомились із процесом розробки паралельних алгоритмів, а також із обміном повідомленнями за допомогою програмного інтерфейсу MPI.

A. Програма для розв'язку поставленої задачі

Лістинг A.1: Початковий код програмного модуля для розв'язання задачі

```
1  """
2  Solution to PDC lab about message passing.
3
4  Before running, make sure and MPI implementation is installed, e.g. Open
   ↳ MPI, Microsoft MPI etc.
5
6  After an MPI environment has been installed, run using:
7      mpiexec -n 5 python solution.py
8  """
9  from mpi4py import MPI
10 import numpy as np
11
12 import logging
13
14 logging.basicConfig(
15     level=logging.INFO,
16     format="%(asctime)s - %(levelname)s: %(message)s",
17 )
18
19 DIMENSION = 10
20 THREAD_COUNT = 5
21 WORKER_IDS = [0, 1, 3, 4]
22
23 H = DIMENSION // THREAD_COUNT
24
25 D_VAL = 5
26 B_VAL = np.full((DIMENSION), 2)
27 Z_VAL = np.full((DIMENSION), 3)
28 MS_VAL = np.full((DIMENSION, DIMENSION), 1)
29
30 comm = MPI.COMM_WORLD
31 rank = comm.Get_rank()
32
33 def compute_A_H(d, B, Z, MS, chunk_id):
34     start = chunk_id * H
35     stop = (chunk_id + 1) * H
36     B_H = B[start:stop]
37     MS_H = MS[:, start:stop]
38     A_H = d * B_H + Z.dot(MS_H)
39     logging.debug("A_H = {}".format(A_H))
40
41     res = {
42         "part_data": A_H,
```

```

43         "start": start,
44         "stop": stop,
45     }
46     return res
47
48     # Follower scenario. Store no data.
49     if rank != 2:
50         data = comm.recv(source=2, tag=1)
51         logging.debug(data)
52         A_H = compute_A_H(**data, chunk_id=rank)
53         comm.send(A_H, dest=2, tag=2)
54     # Leader scenario. Stores all the data.
55     elif rank == 2:
56         # Initialize data
57         A = np.empty(DIMENSION, dtype=np.int)
58         d = D_VAL
59         B = B_VAL
60         Z = Z_VAL
61         MS = MS_VAL
62
63         # Gather all data into one dictionary for simple message passing
64         data = {
65             "d": d,
66             "B": B,
67             "Z": Z,
68             "MS": MS,
69         }
70
71         # Send messages to all followers
72         for i in WORKER_IDS:
73             logging.debug(i)
74             comm.send(data, dest=i, tag=1)
75
76         # Receive and compile results from all followers
77         for i in WORKER_IDS:
78             res = comm.recv(source=i, tag=2)
79             logging.debug("Received {} from worker {}".format(res, i))
80             A[res["start"]:res["stop"]] = res["part_data"]
81
82         # Compute the leader's part
83         res = compute_A_H(**data, chunk_id=2)
84         A[res["start"]:res["stop"]] = res["part_data"]
85
86         logging.info("Final A = {}".format(A))

```
