Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра системного проектування

Розподілені обчислення та інтернет-технології

Лабораторна робота №5

Виконала:

студентка групи ДА-11мп

Молчанова В.С.

Київ – 2021 р.

Головна відмінність колективних операцій від операцій типу точка-точка полягає в тому, що в них беруть участь всі процеси, пов'язані з деяким комунікатором. Недотримання цього правила призводить або до аварійного завершення завдання, або ще більш неприємного зависання завдання.

Відмінні риси колективних операцій:

* Колективні комунікації не взаємодіють із комунікаціями типу точка-точка.
* Колективні комунікації виконуються в режимі блокування. Повернення з підпрограми у кожному процесі відбувається тоді, коли його участь у колективній операції завершилося, однак це не означає, що інші процеси завершили операцію.
* Кількість отриманих даних має дорівнювати кількості посланих даних.
* Повідомлення не мають ідентифікаторів.

## Дослідження широкомовного обміну

### Аналіз функції широкомовного обміну bcast ()

Широкомовне розсилання даних виконується за допомогою функції MPI\_Bcast. Процес з номером root розсилає повідомлення з буфера передачі всім процесам області зв'язку комунікатора comm. Після завершення підпрограми кожен процес в області зв'язку комунікатора comm, включаючи самого відправника, отримає копію повідомлення від процесу-відправника root.

Функція широкомовного обміну в mpi4py:

bcast(obj, root=0): Any

Параметри:

* obj  (Any) – дані для пересилки
* root ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса-відправника

Повертає об'єкт, що надсилався, усім процесам.

Програма-шаблон використання широкомовного обміну:

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

n\_procs = comm.Get\_size()

root = 0

if rank == root:

    data = "Some data"

else:

    data = None

data = comm.bcast(data, root=root)

print(f'Process {rank} received data: {data}')

Результат виконання:

Table

Description automatically generated

### Аналіз функцій операцій  редукцій reduce()

У паралельному програмуванні математичні операції над блоками даних, розподілених на процесорах, називають глобальними операціями редукції. У випадку операцією редукції називається операція, аргументом якої є вектор, а результатом - скалярна величина, отримана застосуванням деякої математичної операції всім компонентам вектора. Зокрема, якщо компоненти вектора розташовані в адресних просторах процесів, що виконуються на різних процесорах, то в цьому випадку говорять про глобальну (паралельну) редукцію. Наприклад, нехай в адресному просторі всіх процесів деякої групи процесів є копії змінної var (необов'язково мають одне й те саме значення), тоді застосування до неї операції обчислення глобальної суми або, іншими словами, операції редукції SUM поверне одне значення, яке міститиме суму всіх локальних значень цієї змінної. Використання цих операцій одна із основних засобів організації розподілених обчислень.

Функція редукції в mpi4py:

reduce(sendobj, op=SUM, root=0): Optional[Any]

Параметри:

* obj  (Any) – дані для виконання редукції
* op (Union[Op, Callable[[Any, Any], Any]]) – операція MPI або користувацька функція
* root ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса-отримувача результату

Повертає процесу root результат редукції

Програма-шаблон використання редукції:

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

n\_procs = comm.Get\_size()

root = 0

def my\_reduce\_operation(data1: str, data2: str):

    return f'{data1} and {data2}'

data = f'data{rank}'

print(f'Data of process {rank}: {data}')

reduced\_data = comm.reduce(data, my\_reduce\_operation, root=root)

if rank == root:

    print(f'Reduced data: {reduced\_data}')

Результат виконання:

Text

Description automatically generated

## Дослідження функцій розсилки та збору

### Аналіз функції розсилки scatter()

Функція MPI\_Scatter розбиває повідомлення з буфера посилки процесу root на рівні частини і посилає i-ю частину буфер прийому процесу з номером i (в тому числі і самому собі). Процес root використовує обидва буфери (посилки і прийому), тому в підпрограмі, що викликається, всі параметри є суттєвими. Інші процеси групи з комунікатором comm є лише одержувачами, тому для них параметри, що специфікують буфер посилки, не суттєві.

Функція розсилки в mpi4py:

reduce(sendobj, root=0): Any

Параметри:

* sendobj  (Sequence[Any]) – список даних для розсилки (використовється тільки в root процесі)
* root ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса, що виконує розсилку

Повертає кожному процесу відповідний об'єкт із надіслоного списку.

Програма-шаблон використання розсилки:

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

n\_procs = comm.Get\_size()

root = 0

if rank == root:

    data\_list = [f'data for process {i}' for i in range(n\_procs)]

    print(f'Data to scatter:{data\_list}')

else:

    data\_list = None

proc\_data = comm.scatter(data\_list, root=root)

print(f'Process {rank} received: {proc\_data}')

Результат виконання

Text

Description automatically generated

### Аналіз функції збору gather()

Функція MPI\_Gather здійснює складання блоків даних, що посилаються всіма процесами групи, в один масив процесу з номером root. Довжина блоків передбачається однаковою. Об'єднання відбувається у порядку збільшення номерів процесів-відправників. Тобто дані, надіслані процесом i зі свого буфера sendbuf, поміщаються в i-у порцію буфера recvbuf процесу root. Довжина масиву, в який збираються дані, повинна бути достатньою для їхнього розміщення.

Функція збору в mpi4py:

gather(sendobj, root=0): Optional[List[Any]]

Параметри:

* sendobj  (Sequence[Any]) – дані від процеса для збору
* root ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса, що отримає список-результат

Повертає процесу root список з даними від усіх процесів.

Програма-шаблон використання збору:

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

n\_procs = comm.Get\_size()

root = 0

proc\_data = f'data{rank}'

print(f'Data from process {rank}: {proc\_data}')

data\_list = comm.gather(proc\_data, root=root)

if rank == root:

    print(f'Gathered data: {data\_list}')

Результат виконання

Chart

Description automatically generated with low confidence

## Використання шаблонів для розподілених обчислень

Обчислення добутку матриць:

from mpi4py import MPI

import random

from functools import reduce

import numpy as np

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

n\_procs = comm.Get\_size()

root = 0

m\_size = 500

def generate\_matrix():

    return [[ random.randint(0, 100) for \_ in range(m\_size)] for \_ in range(m\_size)]

def scalar\_mult(v1, v2):

    return sum(map(lambda x: x[0]\*x[1], zip(v1, v2)))

if rank == root:

    matrix1 = generate\_matrix()

    matrix2 = np.array(generate\_matrix())

    matrix1\_chunks = list(map(lambda l: l.tolist(), np.array\_split(matrix1, n\_procs)))

    start = MPI.Wtime()

else:

    matrix1\_chunks = None

    matrix2 = None

matrix1\_chunk = comm.scatter(matrix1\_chunks, root=0)

matrix2 = comm.bcast(matrix2, root=root)

proc\_rows = [[scalar\_mult(row, matrix2[:, i]) for i in range(m\_size)] for row in matrix1\_chunk]

result\_chunks = comm.gather(proc\_rows, root=0)

if rank == root:

    result = np.array(reduce(list.\_\_add\_\_, result\_chunks))

    time = MPI.Wtime() - start

    print(f'Elapsed time: {time}')

Час виконання t в залежності від кількості процесів р

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| p | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 27.61 | 13.74 | 9.09 | 7.61 | 6.05 | 5.57 | 4.84 |

## Контрольні запитання

1. Означает ли возврат из функции MPI\_Bcast, что содержимое буфера обмена скопировано во все процессы?

Ні, адже повернення з підпрограми у кожному процесі відбувається тоді, коли його участь у колективній операції завершилося, однак це не означає, що інші процеси завершили операцію

1. В каком порядке располагает сообщения корневой процесс при выполнении операции сборки данных MPI\_Gather?

Об'єднання відбувається у порядку збільшення номерів процесів-відправників.

1. Сколько процессов получают данные при использовании MPI\_Reduce?

Один процес, який був вказаний як кореневий