Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра системного проектування

Розподілені обчислення та інтернет-технології

Лабораторна робота №4

Виконала:

студентка групи ДА-11мп

Молчанова В.С.

Київ – 2021 р.

## Дослідження парного обміну

### Парний блокуючий обмін

Блокуючі функції мають на увазі вихід із них лише після повного закінчення операції, тобто викликаючий процес блокується, доки операція не буде завершена. Для функції відправки повідомлення це означає, що всі дані, що пересилаються, поміщені в буфер (для різних реалізацій MPI це може бути або якийсь проміжний системний буфер, або безпосередньо буфер одержувача). Для функції прийому повідомлення блокується виконання інших операцій, поки всі дані буфера не будуть поміщені в адресний простір приймаючого процесу.

У стандартному режимі виконання операції обміну включає три етапи:

1. Передавальна сторона формує пакет повідомлення, в який крім інформації, що передається, упаковуються адреса відправника (source), адреса одержувача (dest), ідентифікатор повідомлення (tag) і комунікатор (comm). Цей пакет передається відправником у системний буфер, і на цьому функція надсилання повідомлення закінчується.
2. Повідомлення системними засобами передається адресатові.
3. Приймаючий процесор отримує повідомлення із системного буфера, коли в нього з'явиться потреба в цих даних. Змістовна частина повідомлення міститься в адресний простір приймаючого процесу (параметр buf), а службова - параметр status.

Блокуюча функція відправки в mpi4py:

send(obj, dest, tag=0)

Параметри:

* obj (Any) – дані для пересилки
* dest ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса-отримувача
* tag ([int](https://docs.python.org/3/library/functions.html#int)) – ідентифікатор повідомлення

Блокуюча функція отримання в mpi4py:

recv(obj , source=ANY\_SOURCE, tag=ANY\_TAG, status=None)

Параметри:

* obj (Any) – змінна для отриманих даних
* dest ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса-отримувача
* tag ([int](https://docs.python.org/3/library/functions.html#int)) – ідентифікатор повідомлення
* status (Optional[Status]) – атрибути отриманого повідомлення

Програма-шаблон використання блокуючого обміну:

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

if rank == 0:

    data = {'a': 1, 'b': 2}

    print(f'Sent data: {data}')

    # data is changed right after the send command, so we need blocking send to be sure that

    # other process receives original version

    comm.send(data, dest=1, tag=1)

    data['a'] = 'hello'

elif rank == 1:

    # data is changed right after the receive command, so we need blocking receive to be sure that

    # original object is not changed in the sending process

    data = comm.recv(source=0, tag=1)

    print(f'Received data: {data}')

    data['a'] = 'world'

print(f'Changed data in {rank}: {data}')

Результат виконання:

A picture containing diagram

Description automatically generated

### Парний неблокуючий обмін

Неблокуючі функції мають на увазі поєднання операцій обміну з іншими операціями, тому неблокуючі функції передачі та прийому по суті є функціями ініціалізації відповідних операцій, які повертають об’єкт типу Request – запит обміну. Для завершення операції вводяться додаткові функції wait та test. Повернення з підпрограми відбувається негайно (immediate) без очікування закінчення передачі даних. Цим пояснюється префікс I в іменах функцій. Тому змінну buf повторно використовувати не можна доти, доки не буде погашено "запит обміну".

Неблокуюча функція відправки в mpi4py:

isend(obj , dest, tag=0): Request

Параметри:

* obj  (Any) – дані для пересилки
* dest ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса-отримувача
* tag ([int](https://docs.python.org/3/library/functions.html#int)) – ідентифікатор повідомлення

Повертає об'єкт типу Request

Неблокуюча функція отримання в mpi4py:

irecv(obj , source=ANY\_SOURCE, tag=ANY\_TAG, status=None): Request

Параметри:

* obj (Any) – змінна для отриманих даних
* dest ([int](https://docs.python.org/3/library/functions.html#int)) – номер процеса-отримувача
* tag ([int](https://docs.python.org/3/library/functions.html#int)) – ідентифікатор повідомлення
* status (Optional[Status]) – атрибути отриманого повідомлення

Повертає об'єкт типу Request

Клас Request має наступні методи для обробки операцій пересилок:

* Нелокальна блокуюча операція Wait. Повернення відбувається після завершення операції, пов'язаної із запитом. У параметрі status повертається інформація про завершену операцію.

Wait(status=None)

Параметри:

status (Optional[Status]) – атрибути отриманого повідомлення

* Локальна неблокуюча функція Test. Якщо пов'язана із запитом операція завершена, повертається true, а status містить інформацію про завершену операцію. Якщо операція, що перевіряється, не завершена, повертається false, а значення status у цьому випадку не визначено.

Test(status=None): bool

Параметри:

status (Optional[Status]) – атрибути отриманого повідомлення

Програма-шаблон використання неблокуючого обміну:

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

if rank == 0:

    data = {'a': 7, 'b': 3.14}

    print(f'Sent data: {data}')

    req = comm.isend(data, dest=1, tag=11)

elif rank == 1:

    req = comm.irecv(source=0, tag=11)

for i in range(2):

        print(f'Calculation {i} in process {rank} that is done while data is sending')

if rank == 0:

    req.wait()

elif rank == 1:

    data = req.wait()

    print(f'Received data: {data}')

Результат виконання:

Text

Description automatically generated

### Додаткові режими комунікації

Як для блокуючих, так і неблокуючих операцій MPI підтримує чотири режими виконання. Ці режими стосуються лише функцій передачі даних, тому для блокуючих та неблокуючих операцій є чотири функції посилки повідомлення. У таблиці нижче перераховані імена базових комунікаційних функцій типу точка-точка, що є в бібліотеці MPI.

|  |  |  |
| --- | --- | --- |
| **Режим виконання** | **З блокуванням** | **Без блокування** |
| Стандартна відправка | MPI\_Send | MPI\_Isend |
| Синхронна відправка | MPI\_Ssend | MPI\_Issend |
| Буферизована відправка | MPI\_Bsend | MPI\_Ibsend |
| Узгоджена відправка | MPI\_Rsend | MPI\_Irsend |
| Прийом | MPI\_Recv | MPI\_Irecv |

* Префикс S (synchronous) – означає синхронний режим передачі. Операція передачі даних закінчується лише тоді, коли закінчується прийом даних. Функція нелокальна.
* Префикс B (buffered) – означає буферизований режим передачі. В адресному просторі процесу передачі за допомогою спеціальної функції створюється буфер обміну, який використовується в операціях обміну. Операція посилки закінчується, коли дані вміщені в цей буфер. Функція має локальний характер.
* Префикс R (ready) – узгоджений чи підготовлений режим передачі. Операція передачі починається лише тоді, коли приймаючий процесор виставив ознаку готовності прийому даних, ініціювавши операцію прийому. Функція нелокальна.
* Префикс I (immediate) – відноситься до неблокуючих операцій.

Всі функції передачі та прийому повідомлень можуть використовуватись у будь-якій комбінації одна з одною. Функції передачі, що знаходяться в одному стовпці, мають абсолютно однаковий синтаксис і відрізняються лише внутрішньою реалізацією.

## Дослідження конструювання та обміну  структур даних

### Аналіз функцій конструювання структур даних

При розробці паралельних програм іноді виникає потреба передавати дані різних типів (наприклад, структури) або дані, розташовані в безмежних областях пам'яті (частини масивів, що не утворюють безперервну послідовність елементів). MPI надає два механізми ефективного пересилання даних у згаданих вище випадках:

* шляхом створення похідних типів для використання у комунікаційних операціях замість визначених типів MPI;
* пересилання упакованих даних (процес-відправник упаковує дані перед їх відправкою, а процес-одержувач розпаковує їх після отримання).

Однак так як в mpi4py функції приймають не буфери, а тип Any, конструювання структур даних є надлишковим, адже за потреби можна переслати просто список різних об’єктів

### Програма-шаблон для парного обміну списками даних

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

if rank == 0:

    data = [1, True, 'and any other objects']

    print(f'Sent data: {data}')

    comm.send(data, dest=1, tag=1)

elif rank == 1:

    data = comm.recv(source=0, tag=1)

    print(f'Received data: {data}')

Результат виконання:

Text

Description automatically generated with medium confidence

### Програма-шаблон для кільцевого парного обміну списками даних

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

n\_procs = comm.Get\_size()

if n\_procs > 1:

    if rank == 0:

        data = [0 for i in range(n\_procs)]

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

        comm.send(data, dest=1)

        data = comm.recv()

        print(f'Final data: {data}')

    else:

        data = comm.recv()

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

        data[rank] = rank

        print(f'Process {rank} made changes')

        comm.send(data, (rank + 1) % n\_procs)

else:

    print("Number of processes should be greater than 1")

A picture containing text

Description automatically generated

## Використання шаблонів для розподілених обчислень

Обчислення середнього арифметичного випадкового масиву чисел:

from mpi4py import MPI

import random

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

n\_procs = comm.Get\_size()

def chunks(lst, n):

    for i in range(0, len(lst), n):

        yield lst[i:i + n]

if n\_procs > 1:

    if rank == 0:

        start = MPI.Wtime()

        n = 1000000

        data = [random.randrange(1, n, 1) for i in range(n)]

        requests = []

        chunks = list(chunks(data, n\_procs - 1))

        for i in range(1, n\_procs):

            requests.append(comm.isend(chunks[i-1], dest=i))

        MPI.Request.waitall(requests)

        requests = []

        for i in range(1, n\_procs):

            requests.append(comm.irecv(source=i))

        results = MPI.Request.waitall(requests)

        print(f'Arifmetic mean: {sum(results)/n}')

        print(f'Elapsed time: {MPI.Wtime() - start}')

    else:

        data = comm.recv()

        sum = sum(data)

        comm.send(sum, dest=0)

else:

    print("Number of processes should be greater than 1")

Час виконання t в залежності від кількості процесів-робітників p (при розмірі масиву чисел n = 1000000):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| p | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 0.8525 | 0.6818 | 0.591 | 0.5735 | 0.5647 | 0.5351 | 0.5432 |

Закон Амдала:

A picture containing text, clock

Description automatically generated

В цій програмі треба виконати n обчислень: n-1 операцію суми та 1 операцію ділення. З них в головному процесі виконується послідовно p операцій: p-1 операцій суми отриманих від процесів проміжних сум та 1 операція ділення. Тоді

Отже, в нашому випадку формула для розрахунку прискорення за законом Амдала набуває вигляд:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| p | 1 | 2 | 3 | 4 | 5 | 6 |
| Sp | 1 | 1.999 | 2.99 | 3.99 | 4.99 | 5.99 |

Можна побачити, що ця задача надто просто для розподілених обчислень, адже з часом на пересилку даних між зростаючою кількістю процесів витрачається більше часу ніж на безпосередньо обчислення незважаючи на теоретичне лінійне прискорення за законом Амдала.

## Контрольні запитання

### Можно ли использовать функцию MPI\_Recv, если не известен отправитель сообщения или тэг сообщения?

Так, адже за замовчуванням параметри source та tag дорівнюють константам «будь-який відправник» та «будь-який тег».

### В каком случае использование функций MPI\_Send и MPI\_Recv приведет к дедлоку?

Якщо усі процеси намагатимуться спочатку отримати дані, а потім відправити їх

### В чем различие между блокирующим и неблокирующим обменом?

При блокуючому обміні викликаючий його процес блокується, доки операція обміну не завершилася та не переходить до наступних команд. При неблокуючому обміні операції відправки та отримання спочатку відбувається лише ініціалізація відповідних операцій, після якої процес може переходити до наступних команд, а завершення цих операцій гарантується додатковими функціями.