Lab 3: Point-to-Point Communication in MPI with Python

Point-to-point communication in MPI allows direct message exchange between two processes. This includes sending and receiving data between specific ranks using ***send()*** and ***recv()*** (or their variants).

1. **Basic MPI Send and Receive**

MPI provides the following functions for point-to-point communication:

- **Blocking Communication**:

  ***comm.send(data, dest, tag)*:** Sends data to a specific process.

  ***comm.recv(source, tag)*:** Receives data from a specific process.

- **Non-blocking Communication**:

  comm.isend(data, dest, tag): Initiates sending data without waiting.

  comm.irecv(source, tag): Initiates receiving data without blocking.

Example: Basic Send and Receive

```python
from mpi4py import MPI

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

if rank == 0:
    data = "Hello from process 0"
    comm.send(data, dest=1, tag=11)  # Sending to process 1
    print(f"Process {rank} sent: {data}")

elif rank == 1:
    received_data = comm.recv(source=0, tag=11)  # Receiving from process 0
    print(f"Process {rank} received: {received_data}")
```

Process 0 sends a message **"Hello from process 0"** to Process 1.

Process 1 receives the message and prints it.

`tag` is an optional identifier to differentiate messages.

Run the script with multiple processes:

```
mpirun -n 2 python lab3a.py
```

2. **Non-Blocking Send and Receive**

Non-blocking communication allows processes to continue executing without waiting for the send/receive to complete.

Example: Non-Blocking Send and Receive

```python
from mpi4py import MPI

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

if rank == 0:
    data = "Hello from process 0"
    req = comm.isend(data, dest=1, tag=11)  # Non-blocking send
    req.wait()  # Ensure the send operation is completed
    print(f"Process {rank} sent: {data}")

elif rank == 1:
    req = comm.irecv(source=0, tag=11)  # Non-blocking receive
    received_data = req.wait()  # Wait for the message
    print(f"Process {rank} received: {received_data}")
```

Key Differences

- **isend()** and **irecv()** start sending/receiving but do not block execution.
- **wait()** ensures the operation completes before proceeding.

3. **Sending and Receiving Structured Data**

MPI can send dictionaries, lists, NumPy arrays, etc.

Example: Sending a Dictionary

```python
from mpi4py import MPI

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

if rank == 0:
    data = {"message": "Hello", "rank": rank}
    comm.send(data, dest=1)
    print(f"Process {rank} sent: {data}")

elif rank == 1:
    received_data = comm.recv(source=0)
    print(f"Process {rank} received: {received_data}")
```

4. **Using** Send() **and** Recv() **for Large Data Transfers**

For large data, **send()** and **recv()** can be inefficient due to serialization overhead. **Send()** and **Recv()** provide a lower-level, efficient way to transfer NumPy arrays.

Example: Sending NumPy Arrays

```python
from mpi4py import MPI
import numpy as np

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

if rank == 0:
    data = np.array([1, 2, 3, 4, 5], dtype=np.int32)
    comm.Send([data, MPI.INT], dest=1)
    print(f"Process {rank} sent: {data}")

elif rank == 1:
    received_data = np.empty(5, dtype=np.int32)
    comm.Recv([received_data, MPI.INT], source=0)
    print(f"Process {rank} received: {received_data}")
```

*Why Use **Send()** and **Recv()**?*

- Direct memory buffer transfer (avoids pickling overhead).
- More efficient for large numerical data.

5. **Synchronous, Buffered, and Ready Modes**

MPI provides different sending modes:

1. **Synchronous Send (Ssend)**: Blocks until the receiver is ready.
2. **Buffered Send (Bsend)**: Uses a user-specified buffer for sending.
3. **Ready Send (Rsend)**: Requires the receiver to be ready beforehand.

Example: Synchronous Send (Ssend)

```
from mpi4py import MPI

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

if rank == 0:
    data = "Hello"
    comm.ssend(data, dest=1)
    print(f"Process {rank} sent synchronously")

elif rank == 1:
    received_data = comm.recv(source=0)
    print(f"Process {rank} received: {received_data}")
```

Key Differences

- send() completes once the message is handed over to MPI.
- ssend() ensures the receiver is ready before completing.

Task-1: *Write an MPI program where* ***Process 0*** *sends an integer array of size 5 to* ***Process 1****, which then modifies the array (e.g., doubles each value) and sends it back to* ***Process 0****. Process 0 should then print the modified array.*

Task-2: *Write an MPI program where* ***Process 0*** *sends a string message to* ***Process 1*** *using non-blocking communication (isend and irecv). Process 1 should receive the message and print it while performing another computation in parallel.*