Lab 4: Advanced Communication Between MPI Processes in Python

Beyond **point-to-point communication** (send()/recv()), MPI provides **advanced communication** methods for **better efficiency, scalability, and flexibility**.

These include:

1. Non-Blocking Communication (isend() **and** irecv()**)**

- Overlaps computation with communication.
- Doesn't block the process; allows it to perform other tasks while the message is being transferred.

Benefits:

Overlaps computation and communication.
Improves performance in large-scale systems.

```python
#Non Bocking Communication
from mpi4py import MPI
import numpy as np

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

if rank == 0:
    data = np.array([100], dtype=np.int32)
    comm.send(data, dest=1, tag=11)
    print(f"Process {rank} sent {data}")

elif rank == 1:
    data = comm.recv(source=0, tag=11)
    print(f"Process {rank} received {data}")
```

2. Collective Communication

Collective communication involves multiple processes and is essential for distributing and collecting data efficiently. MPI provides various collective communication methods:

a) Broadcast (Bcast**)**

A **single process (root)** sends data to **all other processes**.

```
from mpi4py import MPI
import numpy as np

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

data = np.array([100], dtype=np.int32) if rank == 0 else np.empty(1, dtype=np.int32)

comm.Bcast(data, root=0)  # Broadcasting data from root
print(f"Process {rank} received {data}")
```

b) Scatter (Scatter**)**

The **root process** distributes **chunks of data** to different processes.

```
#Scatter processes
from mpi4py import MPI
import numpy as np

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

send_data = np.arange(size, dtype=np.int32) if rank == 0 else None  # Data at root
recv_data = np.empty(1, dtype=np.int32)

comm.Scatter(send_data, recv_data, root=0)
print(f"Process {rank} received {recv_data}")
```

c) Gather (Gather**)**

Each process **sends data to the root**, which collects them in order.

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

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

send_data = np.array([rank], dtype=np.int32)
gathered_data = np.empty(comm.Get_size(), dtype=np.int32) if rank == 0 else None

comm.Gather(send_data, gathered_data, root=0)

if rank == 0:
    print(f"Gathered data at root: {gathered_data}")
```

d) Reduce (Reduce**)**

Performs **an operation (sum, max, min, etc.)** on data from all processes and stores the result in the root process**.**

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

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

send_data = np.array([rank], dtype=np.int32)
result = np.array([0], dtype=np.int32)

comm.Reduce(send_data, result, op=MPI.SUM, root=0)

if rank == 0:
    print(f"Sum of all ranks: {result[0]}")
```

Lab Task: MPI Communication Techniques

Task 1: Non-blocking Communication (Isend/Irecv)

- Implement an MPI program where **Process 0** sends an integer to **Process 1** using **non-blocking communication**(Isend and Irecv).
- Ensure that the receiving process performs a computation before printing the received value.
- How does Isend differ from send, and why is Wait() necessary in non-blocking communication?

Task 2: Reduce Operation

- Write an MPI program where each process initializes a variable with its **rank number** and uses the **Reduce operation** to compute the sum of all ranks at **Process 0**.
- Modify the program to use MPI.PROD instead of MPI.SUM. What difference do you observe in the result?

Task 3: Gather Operation (MPI.Gather)

- Implement an MPI program where each process initializes a number **equal to its rank** and sends it to **Process 0**using MPI.Gather.
- What happens if recv_data is not initialized correctly at the root process?
- Modify the program to use MPI.Gatherv for gathering different amounts of data from each process.

Task 4: Scatter Operation (MPI.Scatter)

- Write an MPI program where **Process 0** initializes an array [10, 20, 30, 40] (for 4 processes) and distributes one value to each process using MPI.Scatter.
- What happens if the number of processes is different from the number of elements in the array?
- Modify the program so that after receiving data, each process **doubles** its value and sends it back to **Process 0** using MPI.Gather.