Week13-mpi-with-python

🛠️ Basic MPI Concepts in Python

1. **Ranks**
   Each process in MPI gets a unique ID called a "rank".
   `rank == 0` is usually the **master** process.
2. **Size**
   Total number of processes running.
3. **Communication**
   Use `send()`, `recv()`, or collective operations like `bcast()` and `gather()`.

🔁 Example: Hello World with `mpi4py`

```
# hello_mpi.py
from mpi4py import MPI

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

print(f"Hello from process {rank} of {size}")
```

Run it with 4 processes:

```
mpiexec -n 4 python hello_mpi.py
```

`mpi4py`

is a **Python wrapper** for the **MPI (Message Passing Interface)** standard, allowing Python programs to perform **parallel processing** across multiple CPUs, cores, or even multiple machines — just like traditional HPC (High Performance Computing) applications written in C or Fortran using MPI.

🐍 What is `mpi4py`?

- `mpi4py` is a Python package that **binds the MPI C API** to Python.
- It enables you to:
  - Launch multiple processes (`mpiexec -n N python your_script.py`)
  - Send and receive messages (`send()`, `recv()`)
  - Use collective communication (`bcast()`, `scatter()`, `gather()`, `reduce()`)
- It supports **NumPy arrays**, making it efficient for large data communication.

🧱 Core Concepts in `mpi4py`

Concept	MPI API	Description
World Comm	`COMM_WORLD`	The default communicator (all processes)
Rank	`Get_rank()`	Each process has a unique rank ID
Size	`Get_size()`	Total number of processes in the group
Send/Recv	`send()`, `recv()`	Point-to-point messaging
Broadcast	`bcast()`	Send data from one to all
Scatter	`scatter()`	Split data across processes
Gather	`gather()`	Collect data from all processes
Reduce	`reduce()`	Aggregate data (e.g., sum, max)
Barrier	`barrier()`	Synchronize all processes

🧪 Why Use `mpi4py`?

- Run the same code across multiple processes (not just threads).
- Avoid Python's Global Interpreter Lock (GIL) for CPU-bound tasks.
- Scale programs across multiple nodes in an HPC cluster.
- Works great with NumPy for numerical and matrix-heavy applications.

📨 1. Point-to-Point Communication: `send()` and `recv()`

```python
# point_to_point.py
from mpi4py import MPI

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

if rank == 0:
    data = "Hello from rank 0"
    comm.send(data, dest=1)
    print(f"[{rank}] Sent data to rank 1")
elif rank == 1:
    data = comm.recv(source=0)
    print(f"[{rank}] Received data: {data}")
```

📣 2. Broadcast: `bcast()`

```python
# broadcast.py
from mpi4py import MPI

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

data = None
if rank == 0:
    data = "Broadcast message"

data = comm.bcast(data, root=0)
print(f"[{rank}] Got: {data}")
```

📦 3. Scatter: `scatter()`

```python
# scatter.py
from mpi4py import MPI

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

if rank == 0:
    data = [i for i in range(size)]  # One value per process
else:
    data = None

recv_data = comm.scatter(data, root=0)
print(f"[{rank}] Received {recv_data}")
```

📥 4. Gather: `gather()`

```python
# gather.py
from mpi4py import MPI

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

send_data = rank * 2
gathered_data = comm.gather(send_data, root=0)

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

➕ 5. Reduce: `reduce()`

```python
# reduce.py
from mpi4py import MPI

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

send_value = rank + 1  # 1 for rank 0, 2 for rank 1, etc.
total = comm.reduce(send_value, op=MPI.SUM, root=0)

if rank == 0:
    print(f"[{rank}] Sum of ranks+1 = {total}")
```

📦 Example: Distributed Array Sum

```python
# sum_mpi.py
from mpi4py import MPI
import numpy as np

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

# Split array among processes
if rank == 0:
    data = np.arange(100, dtype='i')
    chunks = np.array_split(data, size)
else:
    chunks = None

# Scatter chunks to all processes
local_data = comm.scatter(chunks, root=0)
```

```
# Each process computes local sum
local_sum = np.sum(local_data)

# Gather all local sums at root
total_sum = comm.reduce(local_sum, op=MPI.SUM, root=0)

if rank == 0:
    print("Total sum:", total_sum)
```

⚡ When to Use MPI in Python?

- **Large-scale data processing**
- **Scientific simulations**
- **Parallel numerical computations**
- Where **shared memory models (like multithreading)** won’t scale across machines

---

🧠 Things to Keep in Mind

- MPI excels on clusters; not always efficient on a single machine.
- Python has overhead—use compiled extensions (NumPy, Numba) for heavy lifting.
- Debugging is harder—consider writing unit tests for components before running them distributed.

How to setup in windows?

Microsoft MPI v10.1.3

Stand-alone, redistributable and SDK installers for Microsoft MPI

https://www.microsoft.com/en-us/download/details.aspx?id=105289

✅ To fix this on Windows, follow these steps:

1. Install Microsoft MPI

Download the **Microsoft MPI (MS-MPI)** from the official site:
https://www.microsoft.com/en-us/download/details.aspx?id=105289

- Install both:
  - **MS-MPI Redistributable Package**
  - **MS-MPI SDK**

This will install `mpiexec.exe` and necessary DLLs.

---

2. Add MPI to System PATH (if needed)

If after install it still fails, manually add this to your PATH:

```
C:\Program Files\Microsoft MPI\Bin
```

---

3. Verify Installation

Open a new **Command Prompt**, and run:

```
mpiexec
```

You should see usage info instead of an error.

---

4. Install `mpi4py` (if not done yet)

```
pip install mpi4py
```

🧪 Test It