Lab 2: Introduction to Message Passing Interface (MPI) in Python

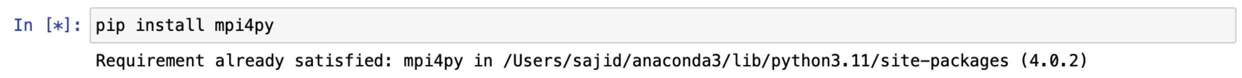
Message Passing Interface (MPI) is a standardized communication protocol used for parallel computing, allowing multiple processes to communicate and work together on distributed systems. It is widely used in high-performance computing (HPC) to improve the efficiency of large-scale applications.

Why Use MPI?

- Enables parallel processing across multiple processors or nodes.
- Supports distributed memory computing.
- Scales efficiently for large computations.

Setting Up MPI in Python

To use MPI in Python, we use the **mpi4py** library, which provides bindings for MPI in Python.



```
In [*]: pip install mpi4py
        Requirement already satisfied: mpi4py in /Users/sajid/anaconda3/lib/python3.11/site-packages (4.0.2)
```

Ensure that an MPI implementation (like MPICH or OpenMPI) is installed on your system.

Basic MPI Concepts

MPI follows a **communicator-based** approach, where processes work together by exchanging messages.

1. MPI Initialization and Finalization in Python (mpi4py)

Call MPI_Init() and MPI_Finalize(), mpi4py automatically initializes MPI when the script starts and finalizes it when the script ends. However, we can still manually control it using:

- **MPI.Init()** – Starts the MPI environment (Optional in mpi4py).
- **MPI.Finalize()** – Cleans up the MPI environment (Optional in mpi4py but useful for manual control).

2. *MPI Communicator*

- **MPI.COMM_WORLD**: A global communicator that includes all available processes.
- Used to send and receive messages between processes.

3. *Rank and Size*

Each process in an MPI program has:

Rank: A unique identifier assigned to each process.

Size: The total number of processes running.

```
In [ ]: from mpi4py import MPI

        comm = MPI.COMM_WORLD  # Global communicator
        rank = comm.Get_rank()  # Get process rank
        size = comm.Get_size()  # Get total number of processes

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

If you are using jupyter notebook file with .ipynb extension then covert the .ipynb to .py extension.

Method 1: Using Jupyter nbconvert (Recommended):

To convert a Jupyter Notebook (.ipynb) file to a Python script (.py), you can use the following methods:

Run the following command in your terminal or command prompt:

```
jupyter nbconvert --to script your_notebook.ipynb
```

This will generate a your_notebook.py file in the same directory.

Method 2: Using nbconvert in a Python Script

You can also do this inside a Python script:

```
In [1]: #COnvert jupyter file to .py
        import nbformat
        from nbconvert import PythonExporter

        # Load the notebook file
        with open('lab1.ipynb') as f:
            notebook = nbformat.read(f, as_version=4)

        # Convert to Python script
        python_exporter = PythonExporter()
        script, _ = python_exporter.from_notebook_node(notebook)

        # Save the output to a .py file
        with open('lab1.py', 'w') as f:
            f.write(script)
```

Method 3: Using Jupyter Notebook Interface

1. Open the .ipynb file in Jupyter Notebook.
2. Click on **File → Download as → Python (.py)**.
3. The .py file will be downloaded.

Run the script using (mpirun):
To run the script, you must use the **mpirun** command, which is the tool that executes MPI jobs:

```
mpirun -n 4 python test_mpi.py
```

Output:

```
Hello from rank 0 out of 4 processes.
Hello from rank 1 out of 4 processes.
Hello from rank 2 out of 4 processes.
Hello from rank 3 out of 4 processes.
```

Running MPI in Jupyter Notebooks: Jupyter notebooks run interactively and typically only in a single process, use a small trick to launch multiple MPI processes within the notebook. Run MPI in Jupyter by using the ! operator to run shell commands, like **mpirun**, directly within the notebook:

> *!mpirun -n 4 python -c "from mpi4py import MPI; comm = MPI.COMM_WORLD; rank = comm.Get_rank(); size = comm.Get_size(); print(f'Hello from rank {rank} out of {size} processes.')"*

Tasks:

1. Explain the role of mpirun/ mpiexec and why we use it to run MPI programs.
2. Write a Python script that retrieves and prints the rank (process ID) and total number of processes.
3. Explain what MPI.COMM_WORLD does in an MPI program.
4. Write a program Count the number of CPU cores installed on your system using Python.