Assignment - 12

Q1) Write about derived data types used in MPI programming.

Derived data types in MPI (Message Passing Interface) programming are user-defined data structures that allow for more complex data communication than simple built-in types like integers or floats. These types are particularly useful when dealing with structured data such as arrays, structs, or nested data types.

Q2) Steps to create and use derived data types

Here are the steps to create and use derived data types in MPI programming: (1) Define the structure: First, define the structure of your derived data type. This could be a simple struct or a more complex nested structure. (2)

Create the MPI datatype: Use MPI functions to create a new MPI datatype that corresponds to your defined structure. You can specify the arrangement of data within your structure and how it should be communicate(2) d.

Use the MPI datatype: Once you have created the derived datatype, you can use it in MPI communication functions to send and receive data.

Q3) Write its uses.

Derived data types are useful in MPI programming for several reasons: (1) Efficiency: By defining the layout of your data explicitly, you can optimize communication by minimizing the amount of data that needs to be transferred.(2)

Abstraction: Derived data types allow you to abstract away the details of the underlying data structure, making your code cleaner and more modula(3) r.

Complex data structures: MPI's built-in communication functions are designed for simple data types like integers and floats. Derived data types allow you to communicate more complex data structures efficiently.

Q4) Implement communication of derived data using one suitable example.

In [1]:
```python
from mpi4py import MPI

# Create MPI communicator
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Define the structure of the data
class Particle:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Data to be sent from rank 0
send_data = None
if rank == 0:
    send_data = Particle(64, 108)

# Scatter data from rank 0 to other processes
recv_data = comm.scatter([send_data] * size, root=0)

# Print received data on each process
print(f"Process {rank} received data: x={recv_data.x}, y={recv_data.y}")

MPI.Finalize()
```

```
Process 0 received data: x=64, y=108
```

In [3]:
```
!mpiexec -n 4 python 12.py
```

```
Process 0 received data: x=64, y=108
Process 3 received data: x=64, y=108
Process 2 received data: x=64, y=108
Process 1 received data: x=64, y=108
```

In [ ]: