HPC_Assignment_12

February 22, 2024

1 Assignment 12 (Derived_data_Type)

1. Write about derived data types used in MPI programming.

- Derived data types in MPI (Message Passing Interface) programming are custom data structures formed by combining basic MPI data types such as integers, floating-point numbers, and characters. These custom data types enable the efficient transmission of complex data structures between MPI processes. Structures, arrays of structures, and arrays of arrays are examples of common derived data types.

2. Steps to create and use derived data types.

- The Procedure to Create and Use Derived Data Types in MPI:
  - Define the data type's structure using the MPI_Type_struct function.
  - Use the MPI_Type_create_struct function to specify the displacement and size of each element in the structure.
  - Use MPI_Type_commit to commit the newly created data type.
  - Use the derived data type for MPI communication functions like MPI_Send and MPI_Recv.

3. Write its uses.

- Applications of derived data types in MPI:
  - Efficient transmission of complex data structures: Derived data types enable the transmission of structured data, such as arrays or nested structures, via a single MPI communication call.
  - Improved performance: By using custom data types tailored to the application's data layout, MPI communication can be made more efficient.
  - Simplified communication code: Derived data types abstract the complexities of data transmission, making code cleaner and easier to maintain.

4. Implement communication of derived data using one suitable example.

[1]:
```python
from mpi4py import MPI
import numpy as np
```

[2]:
```python
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

[23]:
```
class Particle:
    def __init__(self, x, y):
        self.x = x
        self.y = y

send_data = None
if rank == 0:
    send_data = Particle(6, 46)

recv_data = comm.scatter([send_data] * size, root=0)

print(f"Process {rank} received data: x={recv_data.x}, y={recv_data.y}")

MPI.Finalize()
```

```
Process 0 received data: x=6, y=46
```

[24]:
```
!mpiexec -n 8 python as-12.py
```

```
Process 5 received data: x=6, y=46
Process 4 received data: x=6, y=46
Process 1 received data: x=6, y=46
Process 0 received data: x=6, y=46
Process 2 received data: x=6, y=46
Process 3 received data: x=6, y=46
Process 6 received data: x=6, y=46
Process 7 received data: x=6, y=46
```

[25]:
```
!mpiexec -n 16 python as-12.py
```

```
Process 2 received data: x=6, y=46
Process 9 received data: x=6, y=46
Process 1 received data: x=6, y=46
Process 10 received data: x=6, y=46
Process 7 received data: x=6, y=46
Process 8 received data: x=6, y=46
Process 3 received data: x=6, y=46
Process 6 received data: x=6, y=46
Process 13 received data: x=6, y=46
Process 11 received data: x=6, y=46
Process 5 received data: x=6, y=46
Process 12 received data: x=6, y=46
Process 14 received data: x=6, y=46
Process 4 received data: x=6, y=46
Process 15 received data: x=6, y=46
Process 0 received data: x=6, y=46
```