Parallel & Distributed Computing Lecture week 11: Distributed Memory Programming

Farhad M. Riaz

Farhad.Muhammad@numl.edu.pk

**Department of Computer Science
NUML, Islamabad**

Communicators and Ranks

```
from mpi4py import MPI
comm =MPI.COMM_WORLD
rank = comm.Get_rank()
print('My rank is ',rank)
```

Point-to-Point Communication

```
from mpi4py import MPI
import numpy
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
        data = {'a': 7, 'b': 3.14}
       comm.send(data, dest=1)
elif
       rank == 1: data = comm.recv(source=0)
       print('On process 1, data is ',data)
```

Collective Communication Broadcasting:

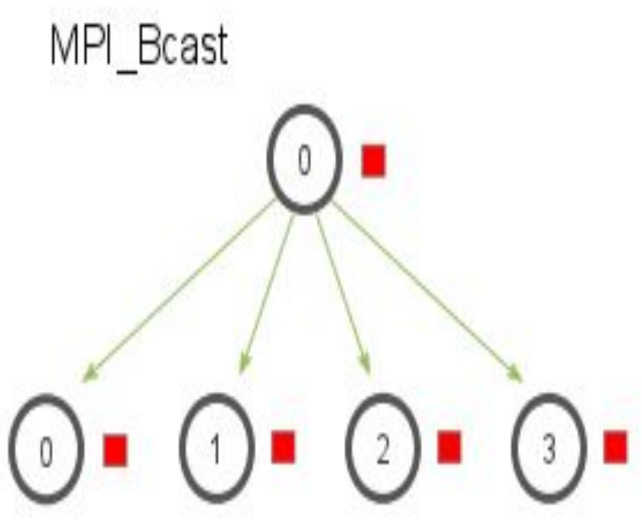
- Broadcasting takes a variable and sends an exact copy of it to all processes on a communicator.

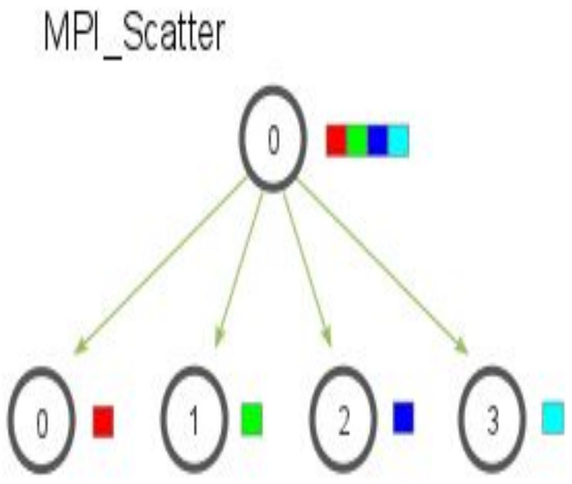
```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
    data = {'key1' : [1,2, 3], 'key2' : ( 'abc', 'xyz’)}
else:
    data = None
    data = comm.bcast(data, root=0)
print('Rank: ',rank,', data: ' ,data)
```

```
from mpi4py import MPI
import numpy as np
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
            # create a data array on process 0
           # in real code, this section might
           # read in data parameters from a file
           numData = 10
           data = np.linspace(0.0,3.14,numData)
 else:
           numData = None
           # broadcast numData and allocate array on other ranks:
           numData = comm.bcast(numData, root=0)
 if rank != 0:
           data = np.empty(numData, dtype='d’)
           comm.Bcast(data, root=0)
           # broadcast the array from rank 0 to all others
print('Rank: ',rank, ', data received: ',data)
```

Scattering:

- Scatter takes an array and distributes contiguous sections of it across the ranks of a communicator



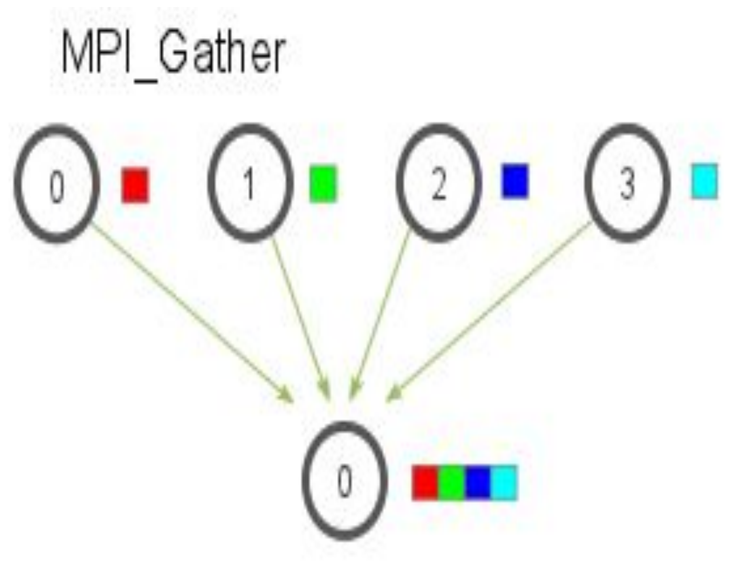


Example

```
from mpi4py import MPI
import numpy as np
comm = MPI.COMM_WORLD
 size = comm.Get_size()
        # new: gives number of ranks in comm
rank = comm.Get_rank()
numDataPerRank = 10
data = None
if rank == 0:
        data=np.linspace(1,size*numDataPerRank,numDataPerRank*size)
        # when size=4 (using -n 4), data = [1.0:40.0]
recvbuf = np.empty(numDataPerRank, dtype='d’)
         # allocate space for recvbuf
comm.Scatter(data, recvbuf, root=0)
print('Rank: ',rank, ', recvbuf received: ',recvbuf)
```

Gathering:

The reverse of a scatter is a gather, which takes subsets of an array that are distributed across the ranks, and *gathers* them back into the full array

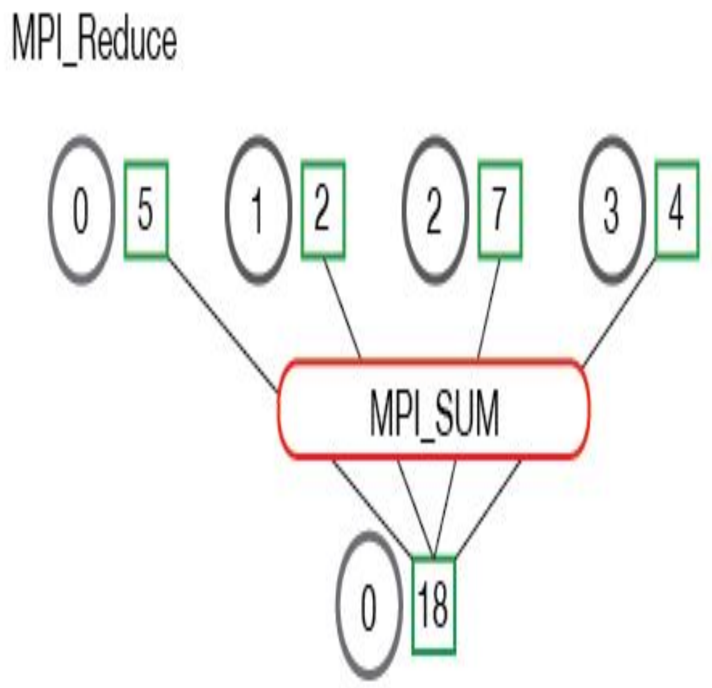


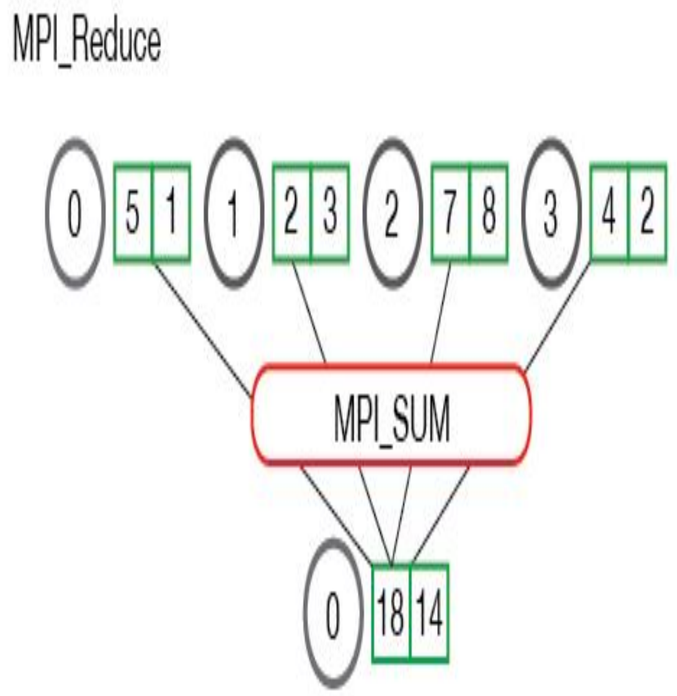
Example

```
from mpi4py import MPI
 import numpy as np
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
numDataPerRank = 10
Sendbuf=np.linspace(rank*numDataPerRank+1,(rank+1)*numDataPerRank,numDataPerRa
nk)
print('Rank: ',rank, ', sendbuf: ',sendbuf)
recvbuf = None
if rank == 0:
        recvbuf = np.empty(numDataPerRank*size, dtype='d') comm.Gather(sendbuf, recvbuf,
root=0)
if rank == 0:
print('Rank: ',rank, ', recvbuf received: ',recvbuf)
```

Reduction:

- The MP reduce operation takes values in from an array on each process and reduces them to a single result on the root process.
- This is essentially like having a somewhat complicated send command from each process to the root process, and then having the root process perform the reduction operation.


MPI_Reduce
0
5
1
2
2
7
3
4
MPI_SUM
0
18


MPI_Reduce
0
5
1
1
2
3
2
7
8
3
4
2
MPI_SUM
0
18
14