Distributed Computing and Introduction to High Performance Computing

Imad Kissami[1]
[1]Mohammed VI Polytechnic University, Benguerir, Morocco

Outline of this lecture

Point-to-point Communications

- Blocking Send and Receive
- Simultaneous Send and Receive

Point-to-point Communications

Blocking Send MPI_SEND

```
MPI_SEND(buf,count,datatype,dest,tag,comm,code)

<type>:: buf
integer :: count, datatype
integer :: dest, tag, comm, code
```

- Sending, from the address buf, a message of count elements of type datatype, tagged tag, to the process of rank dest in the communicator comm.
- the execution remains **blocked** until the message can be re-written without risk of overwriting the value to be sent. In other words, the execution is blocked as long as the message has not been received.

```
COMM.Send(self, data, int dest, int tag=0)
```

Point-to-point Communications

Blocking Receive MPI_RECV

```
MPI_RECV(buf,count,datatype,source,tag,comm,status_msg,code)

<type>:: buf
integer :: count, datatype
integer :: source, tag, comm, code
integer, dimension(MPI_STATUS_SIZE) :: status_msg
```

- Receiving, at the address buf, a message of count elements of type datatype, tagged tag, from the process of rank source in the communicator comm.
- status_msg stores the state of a receive operation : source, tag, code, ...
- An MPI_RECV can only be associated to an MPI_SEND if these two calls have the same envelope (source, dest, tag, comm).
- the execution remains **blocked** until the message content corresponds to the received message.

```
data = COMM.recv(self, source, int tag=0)
# or
Comm.Recv(self, buf, int source, int tag=0, Status status=None)
```

Point-to-point Communications

Blocking Send / Receive Full example

```python
from mpi4py import MPI

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

tag = 99


if RANK == 2:
    sendbuf = 1000
    COMM.send(sendbuf, dest=5, tag=tag)

if RANK == 5:
    recvbuf = COMM.recv(source=2, tag=tag)
    print("I, process 5, I received ",recvbuf," from the process 2.")
```

```
mpirun -n 6 python sendrecv.py

I, process 5, I received  1000  from the process 2.
```

Point-to-point Communications

Simultaneous send and receive MPI_SENDRECV

```fortran
MPI_SENDRECV(sendbuf,sendcount,sendtype,
             dest,sendtag,
             recvbuf,recvcount,recvtype,
             source,recvtag, comm, status_msg, code)

<type >:: sendbuf, recvbuf
integer :: sendcount, recvcount
integer :: sendtype, recvtype
integer :: source, dest, sendtag, recvtag, comm, code

integer, dimension(MPI_STATUS_SIZE) :: status_msg
```

- Sending, from the address sendbuf, a message of sendcount elements of type sendtype, tagged sendtag, to the process dest in the communicator comm ;
- Receiving, at the address recvbuf, a message of recvcount elements of type recvtype, tagged recvtag, from the process source in the communicator comm.
- Here, the receiving zone recvbuf must be different from the sending zone sendbuf.

```python
Comm.Sendrecv(self, sendbuf, int dest, int sendtag=0, recvbuf=None, int source=ANY_SOURCE, int recvtag=ANY_TAG, Status status=None)
```

Point-to-point Communications

Simultaneous send and receive MPI_SENDRECV: Full example

```python
from mpi4py import MPI

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()
SIZE = COMM.Get_size()
tag = 99

#We define the process we will communicate with (we suppose that we have exactly 2 processes)

num_proc = (RANK+1)%2

sendbuf = RANK + 1000
recvdata = COMM.sendrecv(sendbuf, num_proc, sendtag=tag, recvtag = tag, status=None)
# or
recvdata = COMM.sendrecv(sendbuf, num_proc)

print("I, process {proc_send}, I received {data} from the process "
      "{proc_recv}.".format(proc_send = RANK, data = recvdata, proc_recv = num_proc))
```

```
mpirun -n 2 python simultaneoussendrecv.py
```

```
I, process 0, I received 1001 from the process 1.
I, process 1, I received 1000 from the process 0.
```

Point-to-point Communications

Simultaneous send and receive MPI_SENDRECV: Remarks

In the case of a synchronous implementation of the MPI_SEND() subroutine, if we replace the MPI_SENDRECV() subroutine in the example above by MPI_SEND() followed by MPI_RECV() , the code will deadlock. Indeed, each of the two processes will wait for a receipt confirmation, which will never come because the two sending operations would stay suspended.

```
COMM.send(sendbuf, dest=5, tag=tag)
recvbuf = COMM.recv(source=2, tag=tag)
```