Introduction to Message Passing Interface

What is MPI?

- Message Passing Interface
  - Most useful on distributed memory machines
  - The *de facto* standard parallel programming interface
- Many implementations, interfaces in C/C++, Fortran, Python via MPI4Py

Message Passing Paradigm

- A parallel MPI program is launched as separate processes (tasks), each with thier own address space.
  - Requires partitioning data across tasks.
- Data is explicitly moved from task to task
  - A task accesses the data of another task through a transaction called "message passing" in which a copy of the data (message) is transferred (passed) from one task to another.
- There are two classes of message passing
  - Point-to-point involve only two tasks
  - Collective messages involve a set of tasks

MPI4Py

- MPI4Py provides an interface very similar to the MPI standard C++ interface
- You can communicate Python objects.
- What you may lose in performance, you gain in shorter development time

Communicators

- MPI uses communicator objects to identify a set of processes which communicate only within their set.
- `MPI_COMM_WORLD` is defined as all processes (ranks) of your job.
  - Usually required for most MPI calls
- Rank
  - Unique process ID within a communicator
  - Assigned by the system when the process initalizes
  - Used to specify sources and destinations of messages

helloMPI.py

```
#!/usr/bin/env python
from mpi4py import MPI
comm = MPI.COMM_WORLD
print "Hello, World! My rank is: " + str(comm.rank)
```

Point-to-Point Communication

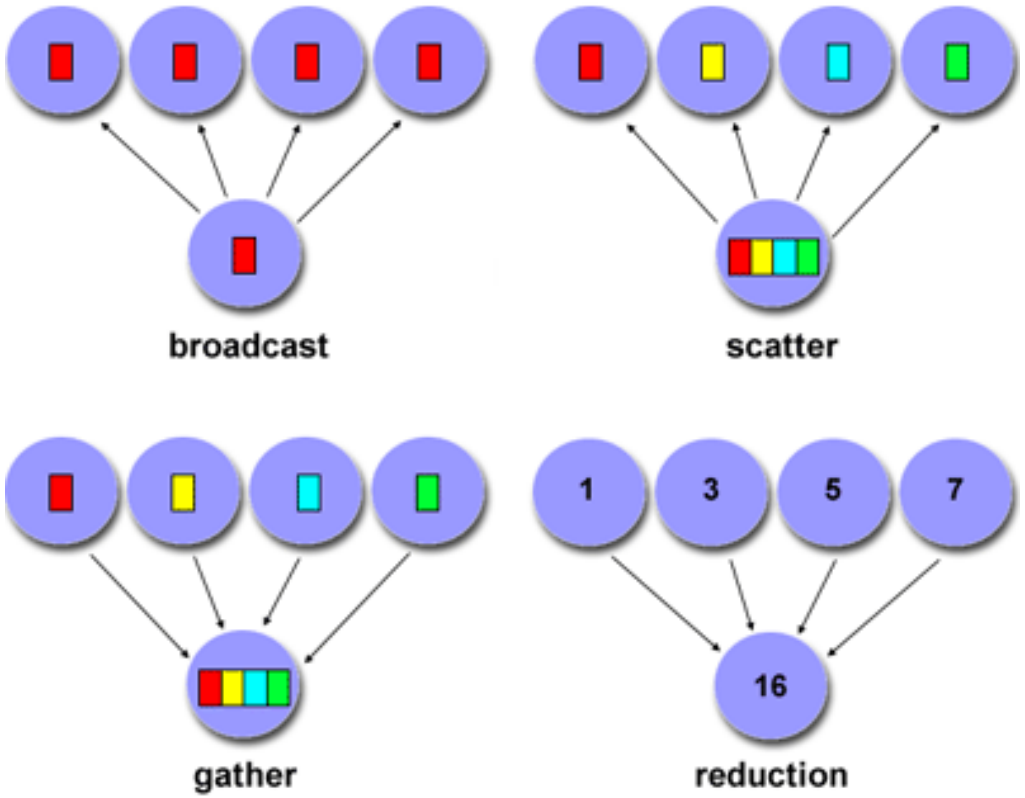
- Sending data from one point (process/task) to another point (process/task)

send_recv.py

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy
comm = MPI.COMM_WORLD
rank=comm.rank
size=comm.size
v=numpy.array([rank]*500,dtype=float)
if comm.rank==0:
    comm.send(v,dest=(rank+1)%size)
if comm.rank > 0:
    data=comm.recv(source=(rank-1)%size)
    comm.send(v,dest=(rank+1)%size)
if comm.rank==0:
    data=comm.recv(source=size-1)

print "My rank is " + str(rank)
print "I received this:"
print data
```

Collective communication



Scatter

- Rank 0 acts as a leader, creating a list and **scatter**ing it out to all ranks evenly

scatter.py

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy
sendbuf=[]
root=0
comm = MPI.COMM_WORLD
if comm.rank==0:
    m=numpy.random.randn(comm.size,comm.size)
    print(m)
    sendbuf=m
v=comm.scatter(sendbuf,root)
print("I got this array:")
print(v)
```

Gather

- `gather` is a command that collects results from all processes into a list.

gather.py

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy
comm = MPI.COMM_WORLD
sendbuf=[]
root=0
if comm.rank==0:
    m=numpy.array(range(comm.size*comm.size),dtype=float)
    m.shape=(comm.size,comm.size)
    print(m)
    sendbuf=m

v=comm.scatter(sendbuf,root)
print("I got this array:")
print(v)
v=v*v
recvbuf=comm.gather(v,root)
if comm.rank==0:
    print numpy.array(recvbuf)
```

Broadcast

- `bcast` sends a single object to every process.

bcast.py

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy
comm = MPI.COMM_WORLD
sendbuf=[]
root=0
if comm.rank==0:
    m=numpy.array(range(comm.size*comm.size),dtype=float)
    m.shape=(comm.size,comm.size)
    print(m)
    sendbuf=m
v=comm.scatter(sendbuf,root)
print("I got this array:")
print(v)
v=v*v
recvbuf=comm.gather(v,root)
if comm.rank==0:
    print numpy.array(recvbuf)
if MPI.COMM_WORLD.rank==0:
        sendbuf="done"
recvbuf=MPI.COMM_WORLD.bcast(sendbuf,root)
print(recvbuf)
```

Reduce

- `reduce` performs a parallel reduction operation
  - The default is summation, but many other operators are available.

reduce.py

```
#!/usr/bin/env python
from mpi4py import MPI
import numpy
comm = MPI.COMM_WORLD
sendbuf=[]
root=0
if comm.rank==0:
    m=numpy.array(range(comm.size*comm.size),dtype=float)
    m.shape=(comm.size,comm.size)
    print(m)
    sendbuf=m

v=comm.scatter(sendbuf,root)
print("I got this array:")
print(v)
v=v*v
recvbuf=comm.reduce(v,root)
if comm.rank==0:
    print numpy.array(recvbuf)
```

References

- http://mpi4py.scipy.org/docs/usrman/index.html
- https://computing.llnl.gov/tutorials/parallel_comp/