MPI Summary for Python with mpi4py

The mpi4py package contains several subpackages, the most important of which is MPI. For full details see

https://mpi4py.readthedocs.io/en/stable

for details. We will discuss only the MPI subpackage in this Guide.

The MPI subpackage in turn contains a set of top-level parameters and methods, plus a number of classes. The only class we will discuss in this Guide is the Comm (communicator) class.

A short tutorial is at

https://mpi4py.readthedocs.io/en/stable/tutorial.html

Importing

Begin by importing the subpackage. Usually we omit the mpi4py prefix.

```
from mpi4py import MPI
```

The MPI constants are top-level attributes in this package. Among the most widely used are

```
MPI.COMM_WORLD

MPI.PROC_NULL

MPI.ANY_TAG

MPI.ANY_SOURCE
```

Predefined MPI types are also attributes of the MPI package.

```
MPI.DOUBLE

MPI.FLOAT

MPI.INT

MPI.BOOL

MPI.COMPLEX
```

Note that these correspond to NumPy types. There are many more; see the API reference for a list.

Usage

The mpi4py package can be used with arbitrary (mutable) Python objects. In this case the MPI routines are invoked with all *lower-case* letters and the object is pickled and unpickled automatically. However,

the simplest and fastest way to use mpi4py is to use NumPy arrays, even if the array has only one element. In this case the initial letter of the method is *capitalized*. This Guide will focus on use with NumPy arrays. See the tutorial for more examples. All examples below will assume

```
import numpy as np
```

It is generally a good idea to create your arrays with an explicit dtype argument.

The Essential Procedures

```
MPI.Init()
```

This can be automatically invoked when a Comm object is created, so is less necessary than for compiled languages.

Finalization is automatic upon exit from the interpreter.

Comm methods

Most of mpi4py can be used through a Comm object.

Instantiating

```
comm=MPI.COMM_WORLD
```

Rank. Note that ranks are relative to a communicator.

```
comm.Get_rank()
```

```
myrank=MPI.COMM_WORLD.Get_rank()
```

Number of processes in a communicator:

```
nprocs=comm.Get_size()
```

Broadcasting:

```
comm.Bcast(buffer,root=0)
```

If a NumPy array is sent, the package can autodiscover the number of items and type, though if not type double it may be safest to send it explicitly.

Example

Broadcast from root (default rank 0) using default MPI.COMM_WORLD explicitly.

```
N=np.array([42],dtype=int)
```

```
MPI.COMM_WORLD.Bcast([N,MPI.INT])
```

Reduction

```
comm.Reduce(sendbuf,recvbuf,op=MPI.SUM,root=0)
```

Send array sendbuf to root in recvbuf, which applies the specified operation (default MPI.SUM). The "buffers" are the arrays.

Commonly-used available operations:

```
MPI.SUM

MPI.PROD

MPI.MAX

MPI.MIN

MPI.MAXLOC

MPI.MINLOC
```

Example:

```
myval=np.array([myrank])

product=np.zeros(1)

MPI.COMM_WORLD.Reduce(myval,product,MPI.PROD)
```

With Reduce only the root has the value. If all ranks are to know the result use

```
comm.Allreduce(sendbuf,recvbuf,op=MPI.SUM)
```

Barrier

```
comm.Barrier()
```

All processes will pause until all members of the specified communicator have invoked the method. This can be used for explicit synchronization.

Send a message, by default to root with tag 0.

```
comm.Send(sendbuf,dest=0,tag=0)
```

Receive a message, by default from root with tag 0 and without creating a Status variable.

```
comm.Recv(recvbuf,source=0,tag=0, Status=None)
```

Send/receive

Write on one line:

```
comm.Sendrecv(sendbuf, dest=0, sendtag=0, recvbuf=None, source=0,
recvtag=0, Status status=None)
```

Send a message sendbuf to dest while receiving a message from source.

Example

```
MPI.COMM_WORLD.Sendrecv([halobuf,MPI.FLOAT], myrank+1, 0,
[bcbuf,MPI.FLOAT],myrank-1,0,stat)
```

Gather

```
Comm.Gather(sendbuf,recvbuf,root=0)
```

Gather items from each process to the specified process (usually process 0) into a larger buffer recvbuf in rank order. Gather assumes all sendbufs are the same size.

Example

```
my_N=np.array([myrank])
all_N=np.empty(nprocs)
MPI.COMM_WORLD.Gather([my_N,MPI.DOUBLE],[all_N,MPI.DOUBLE])
```

With Gather only the root knows the values. If all processes must know them use `comm.Allgather(sendbuf,recvbuf)`

Scatter

```
comm.Scatter(sendbuf,recvbuf,root=0)
```

Distribute items from sendbuf to recvbuf in rank order. Scatter assumes the same quantity of data sent to each process.

Example

```
Narr=10
if myrank==0:
   data=np.arange(nprocs*Narr)
else:
    data=np.empty(nprocs*Narr)
my_data=np.empty(Narr)
MPI.COMM_WORLD.Scatter(data,my_data)
```

Hello, World!

```
from mpi4py import MPI
import sys
myrank=MPI.COMM_WORLD.Get_rank()
```

```
nprocs=MPI.COMM_WORLD.Get_size()
if myrank==0:
   sys.stdout.write(“Running on %d processes\n”%nprocs)
sys.stdout.write(“Greetings from process %d\n”%myrank)
MPI.COMM_WORLD.Finalize()
```