MPI4py crash course

Ariel Lozano and David Colignon

CÉCI training

October 25, 2017

MPI4py initializing and running

- Importing the library

  ```
  from mpi4py import MPI
  ```

- No need to call `MPI_Init()` and `MPI_Finalize()`
  - Importing mpi4py already triggers `MPI_INIT()`
  - `MPI_Finalize()` is called when all python processes exit
- Initializing the main variables

  ```
  comm = MPI.COMM_WORLD
  myrank = comm.Get_rank()
  nproc = comm.Get_size()
  ```

- Parallel execution

  ```
  mpirun -np <N> python mycode.py
  ```

MPI4py: important remarks

- The library supports two types of communication:
  - any kind of **generic python objects**
  - or **python buffer-like objects** allocated in contiguous memory
- The all-lowercase methods `send`, `recv`, `bcast`... allow to communicate generic objects
- Their initally upper case analogues `Send`, `Recv`, `Bcast`... can communicate memory buffers
- Communicating generic objects introduces an overhead, a special binary representation of the message is created to send and restored after received
- For buffer objects (e.g. NumPy arrays) **upper case methods must be used** to avoid unnecessary performance loss!!!

Going from serial to parallel with mpi4py

```python
if (__name__ == '__main__'):
    print("Hello, World !")
```

```
$ python 01_hello.py

 Hello, World !
```

```python
from mpi4py import MPI

if (__name__ == '__main__'):
    print("Hello, World !")
```

```
$ mpirun -np 3 python 01_hello_mpi4py.py

 Hello, World !
 Hello, World !
 Hello, World !
```

Using the Comm class to define communicator variables

```
from mpi4py import MPI

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()

    print("Hello, World ! from process {0} of {1} \n"
          .format(myrank, nproc))
```

```
$ mpirun -np 3 python 02_hello_mpi4py_details.py

 Hello, World ! from process 0 of 3

 Hello, World ! from process 1 of 3

 Hello, World ! from process 2 of 3
```

Point-to-point (P2P) communications

- Blocking communication
  - Python objects

```
comm.send(sendobj, dest=1, tag=0)
recvobj = comm.recv(None, src=0, tag=0)
```

  - Numpy buffer

```
comm.Send([sendarray, count, datatype], dest=1, tag=0)
comm.Recv([recvarray, count, datatype], src=0, tag=0)
```

- Nonblocking communication
  - Python objects

```
reqs = comm.isend(object, dest=1, tag=0)
reqr = comm.irecv(source=0, tag=0)
reqs.wait()
data = reqr.wait()
```

  - Numpy buffer

```
reqs = comm.Isend([sendarray, count, datatype], dest=1, tag=0)
reqr = comm.Irecv([recvarray, count, datatype], src=0, tag=0)
MPI.Request.Waitall([reqs, reqr])
```

Point-to-point (P2P) communications

- Blocking communication
  - Python objects

```
comm.send(sendobj, dest=1, tag=0)
recvobj = comm.recv(None, src=0, tag=0)
```

  - Numpy buffer

```
comm.Send([sendarray, count, datatype], dest=1, tag=0)
comm.Recv([recvarray, count, datatype], src=0, tag=0)
```

- Nonblocking communication

Note: `datatype` discovery is supported and `count` can be inferred with this and the buffer bite-size. Thus,

```
comm.Send(sendarray, dest=1, tag=0)
comm.Recv(recvarray, src=0, tag=0)
```

could be used equivalently.

For didactic purposes we'll follow here Zen of Python statement "Explicit is better than implicit" and always pass all the arguments.

P2P communication of generic object

```
from mpi4py import MPI

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()

    if (myrank == 0):
        for i in range(1, nproc):
            a = {"Day": "Monday", "Age": 20, "z": [90, 3, 1]}
            comm.send(a, dest=i, tag=7)
    else:
        a_recv = comm.recv(source=0, tag=7)
        print("I'm process {0} and received: {1}\n".format(
                                            myrank, a_recv))
```

```
$ mpirun -np 3 python 03_send_dict.py

I'm process 2 and received: {'Day': 'Monday', 'Age': 20, 'z': [90, 3, 1]}

I'm process 1 and received: {'Day': 'Monday', 'Age': 20, 'z': [90, 3, 1]}
```

P2P communication of numpy array

```
from mpi4py import MPI
import numpy as np

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()
    if (myrank == 0):
        for i in range(1, nproc):
            a = np.arange(10, dtype='i')
            comm.Send([a, 10, MPI.INT], dest=i, tag=7)
    else:
        a = np.zeros(10, dtype='i')
        comm.Recv([a, 10, MPI.INT], source=0, tag=7)
        print("I'm process {0} and received: {1}\n".format(
                                            myrank, a))
```

```
$ mpirun -np 3 python 04_send_np_array.py

I'm process 2 and received: [0 1 2 3 4 5 6 7 8 9]

I'm process 1 and received: [0 1 2 3 4 5 6 7 8 9]
```

Sum of the first N integers using P2P communications

```
from mpi4py import MPI
import numpy as np

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()
    N = 1000
    startval = int(N * myrank / nproc + 1)
    endval = int(N * (myrank+1) / nproc)
    partial_sum = np.array(0, dtype='i')
    for i in range(startval, endval+1):
        partial_sum += i
    if (myrank != 0):
        comm.Send([partial_sum, 1, MPI.INT], dest=0, tag=7)
    else:
        tmp_sum = np.array(0, dtype='i')
        for i in range(1, nproc):
            comm.Recv([tmp_sum, 1, MPI.INT], source=i, tag=7)
            partial_sum += tmp_sum
        print("The sum is {0}\n".format(partial_sum))
```

```
$ mpirun -np 3 python 05_sum_p2p.py
The sum is 500500
```

Collective communications

- Broadcast
  - Python objects:

    ```
    recvobj = comm.bcast(sendobj, root=0)
    ```

  - Numpy buffer:

    ```
    comm.Bcast(buf, root=0)    # buf = [array, count, datatype]
    ```

- Scatter, Gather, Allgather
  - Python objects: `sendobj` single value or `comm.size()` size

    ```
    recvobj = comm.scatter(sendobj, root=0) # return single value
    recvobj = comm.gather(sendobj, root=0) # return comm.size() list
    recvobj = comm.allgather(sendobj)      # return comm.size() list
    ```

  - Numpy buffer: `count` value of the message can be relevant here

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

- Reduce
  - Python objects:

    ```
    reducedobj = comm.reduce(sendobj, op=MPI.OPERATION, root=0)
    ```

  - Numpy buffer:

    ```
    comm.Reduce(sendbuf, reducedbuf, op=MPI.OPERATION, root=0)
    ```

Sum of the first N integers using MPI_Reduce

```
from mpi4py import MPI
import numpy as np

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()
    N = 1000
    startval = int(N * myrank / nproc + 1)
    endval = int(N * (myrank+1) / nproc)
    partial_sum = np.array(0, dtype='i')
    for i in range(startval, endval+1):
        partial_sum += i

    tot_sum = np.array(0, dtype='i')
    comm.Reduce([partial_sum, 1, MPI.INT],
                [tot_sum, 1, MPI.INT], op=MPI.SUM, root=0)

    if (myrank == 0):
        print("The sum is {0}\n".format(tot_sum))
```

Usage on CÉCI clusters

Depending if you wish a `python2` or `python3` environment:

- NIC4:

```
module load EasyBuild
module load Python/2.7.13-foss-2017a
or
module load Python/3.5.1-foss-2016a
```

- lemaitre2, hmem, hercules, vega:

```
module load Python/2.X.X-foss-2016x
or
module load Python/3.X.X-foss-2016x
```

- dragon1:

```
module load Python/2.7.3-goolf-1.4.10
or
module load Python/3.3.2-goolf-1.5.14
```

- vega:

```
module load Python/2.X.X-intel-201Xx
or
module load Python/3.X.X-intel-201Xx
```

Linux installation and useful links

If you have your own linux installation:

- Ubuntu, Debian

  ```
  apt-get install python-mpi4py python3-mpi4py
  ```

- Fedora, CentOS (EPEL repository)

  ```
  yum install mpi4py-openmpi
  ```

- ArchLinux (community repository)

  ```
  pacman -S python-mpi4py python2-mpi4py
  ```

- Tutorial on official Documentation

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

- MPI4py API reference

  https://mpi4py.scipy.org/docs/apiref/frames.html