MPI4py crash course

Ariel Lozano and Orian Louant

CÉCI trainings

October 29, 2020

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 !
```

MPI4py initializing and running

- Importing the library

  ```
  from mpi4py import MPI
  ```

- Initializing the main parallel workflows variables

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

- No need to call `MPI_Init()` and `MPI_Finalize()`
  - Importing mpi4py already triggers `MPI_INIT()`
  - `MPI_Finalize()` is called when all python processes exit
- Code 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!!!

Using the `Comm` class to define communicator variables

```python
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

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

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.

But we'll follow here Zen of Python statement ”Explicit is better than implicit” and always pass all the arguments.

P2P communication of generic object

```python
from mpi4py import MPI

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

    if (myrank == 0):
        a = {"Day": "Monday", "Age": 20, "z": [90, 3, 1]}
        for i in range(1, nproc):
            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

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

```python
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)    # with buf = [array, count, datatype]
    ```

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

    ```
    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 collective comms

```python
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))
```

Scatter a python object

```python
from mpi4py import MPI

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()
    assert nproc == 3     #this basic example works only in 3 proc
    if myrank == 0:
        #object to scatter MUST be tuple or list of size comm.Get_size
        fulldata = [ 23, "AB", ["z", 22]]
        print("I'm {0} fulldata is: {1}".format(myrank,fulldata))
    else:
        fulldata = None     #all the procs must have a value for fulldata

    mydata = comm.scatter(fulldata, root=0)
    print("After Scatter, I'm {0} and mydata is: {1}".format(myrank,mydata))
```

```
$ mpirun -np 3 python 09_scatter_pyobj.py

I'm 0 fulldata is: [23, 'AB', ['z', 22]]
After Scatter, I'm 1 and mydata is: AB
After Scatter, I'm 0 and mydata is: 23
After Scatter, I'm 2 and mydata is: ['z', 22]
```

Scatter a Numpy array

```python
from mpi4py import MPI
import numpy as np

if (__name__ == '__main__'):
    comm = MPI.COMM_WORLD
    myrank = comm.Get_rank()
    nproc = comm.Get_size()
    assert nproc == 3
    if myrank == 0:
        fulldata = np.arange(9, dtype='i')
        print("I'm {0} fulldata is: {1}".format(myrank,fulldata))
    else:
        fulldata = None

    count = 3
    mydata = np.zeros(count, dtype='i')
    comm.Scatter([fulldata, count, MPI.INT],[mydata, count, MPI.INT],root=0)
    print("After Scatter, I'm {0} and mydata is: {1}".format(myrank,mydata))
```

```
$ mpirun -np 3 python 09_Scatter_np.py
I'm 0 fulldata is: [0 1 2 3 4 5 6 7 8]
After Scatter, I'm 0 and mydata is: [0 1 2]
After Scatter, I'm 1 and mydata is: [3 4 5]
After Scatter, I'm 2 and mydata is: [6 7 8]
```

Usage on CÉCI clusters

- NIC4:

```
module load EasyBuild Python/3.5.2-foss-2016b
```

- lemaitre3, dragon2:

```
module load releases/2018b
module load Python/3.6.6-intel-2018b

or

module load releases/2019b
module load Python/3.7.4-GCCcore-8.3.0
module load SciPy-bundle/2019.10-foss-2019b-Python-3.7.4
```

- dragon1, hercules2:

```
module load Python/3.5.2-foss-2016b
```

Usage on CÉCI clusters

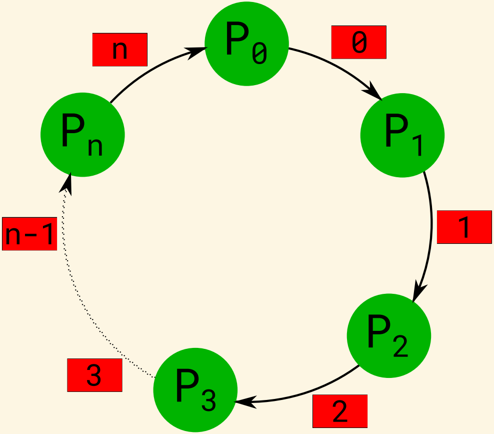
All examples shown before available on CÉCI clusters at:

```
$ ls /CECI/proj/training/MPI/Python/examples/

01_hello_mpi4py.py          07_Bcast_pyobj.py     11_ssend_pyobject.py
01_hello.py                 08_Gather_np_2.py     12_Isend_Irecv_np.py
02_hello_mpi4py_details.py  08_Gather_np.py       13_async_ring.py
03_send_dict.py             09_Scatter_np_ex2.py  13_async_ring_pyobjects.py
04_send_np_array.py         09_Scatter_np_ex3.py  14_Scatterv_np.py
05_sum_p2p.py               09_Scatter_np.py      job_test_lm3.sh
06_sum_reduce.py            09_scatter_pyobj.py
07_Bcast_np.py              10_Alltoall_np.py
```

Excercise: implement a communication ring

Each processor sends a message containing its rank to the following one. The last sends it to processor 0.



Excercise: implement a communication ring

```python
from mpi4py import MPI
import numpy as np

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

    msg = np.array(100 + myrank, dtype='i')
    msg_recv = np.array(0, dtype='i')

    left = TODO
    right = TODO

    comm.Send( TODO , dest=right)
    comm.Recv( TODO , source=left)

    print("I'm myrank {0}: received {1} from processor {2}”
          .format(myrank, msg_recv, left))
```

Excercise: implement a communication ring

```
$ mpirun -np 5 python msg_chain.py

I'm myrank 1: received 100 from processor 0
I'm myrank 2: received 101 from processor 1
I'm myrank 3: received 102 from processor 2
I'm myrank 0: received 104 from processor 4
I'm myrank 4: received 103 from processor 3
```

Useful references

- Tutorial on the official Documentation
  http://mpi4py.readthedocs.io/en/stable/tutorial.html
- Install mpi4py in some linux distros:
  - Ubuntu, Debian
    ```
    apt-get install python3-mpi4py
    ```
  - Fedora, CentOS 8 (EPEL repository)
    ```
    yum install python3-mpi4py-openmpi
    ```
  - ArchLinux (community repository)
    ```
    pacman -S python-mpi4py
    ```