MPI

Message Passing Interface

¿Qué es?

- Protocolo de comunicación para implementar algoritmos en paralelo.
- Puede ser usada en una computadora o en un cluster.
- Implementación principal en C++.
- Implementaciones alternas en otros lenguajes.

http://mpitutorial.com/tutorials/

Funcionalidad

- Distribución de trabajo en entidades llamadas nodos, identificadas por un rango.

- El nodo raíz tiene rango 0, este es quien generalmente genera y distribuye los datos, y recibe los resultados finales para su presentación.

- Provee primitivas de sincronización y paso de mensajes.

Ejemplo básico

Basic.py →

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print ("hello from process ",rank)
```

Para correrlo con 3 procesos:

```
mpiexec -n 3 python basic.py
hello from process  1
hello from process  0
hello from process  2
```

No se ejecutan el orden!!

Ejemplo con un condicional

basic2.py →

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
        print ("I am the root”)
else:
        print ("I am not the root, what do I get?”)
```

mpiexec -n 3 python basic2.py

I am not the root, what do I get?

I am the root

I am not the root, what do I get?

Comunicación punto a punto

Enviando / Recibiendo

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
        data = [1,2,3]
        comm.send(data, dest=1)
elif rank == 1:
        data = comm.recv(source=0)
        print (data)
```

mpiexec -n 1 python sendReceive.py. <-- error

mpiexec -n 2 python sendReceive.py. <-- ok
[1, 2, 3]

mpiexec -n 3 python sendReceive.py. <-- ok (+/-)
[1, 2, 3]

Enviando / Recibiendo

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
        data = [1,2,3]
        comm.send(data, dest=1)
        comm.send(data, dest=2)
else:
        data = comm.recv(source=0)
        print (“process ",rank," received ",data)
```

```
mpiexec -n 3 python sendReceive.py
process  1  received  [1, 2, 3]
process  2  received  [1, 2, 3]
```

Revisar la diferencia con iSend y iRecv
https://stackoverflow.com/questions/10017301/mpi-blocking-vs-non-blocking.
https://nyu-cds.github.io/python-mpi/03-nonblocking/

Enviando / Recibiendo con ssend

```
from mpi4py import MPI
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
        data = [1,2,3]
        print "I am process ",rank,"... going to send"
        comm.ssend(data, dest=1)
        data = [4,5,6]
        comm.ssend(data, dest=2)
        print  ("I am process ",rank,"... msg sent”)
elif rank == 1:
        data = []
        print "I am process ",rank,"... going to receive, but first wait"
        time.sleep(3)
        data = comm.recv(source=0)
        print ("I am process ",rank,"... msg received ", data)
elif rank == 2:
        data = []
        print "I am process ",rank,"... going to receive"
        data = comm.recv(source=0)
        print ("I am process ",rank,"... msg received ", data)
```

¿Cuál es la diferencia entre send y ssend?

Enviando / Recibiendo con tags

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
        data = [1,2,3]
        comm.send(data, dest=1,tag=1)
        data = [4,5,6]
        comm.send(data, dest=1,tag=2)
else:
        data = comm.recv(source=0,tag=2)
        print ("process ",rank," received ",data," with tag 2”)
        data = comm.recv(source=0,tag=1)
        print ("process ",rank," received ",data," with tag 1”)
```

```
mpiexec -n 2 python sendReceive.py
process  1  received  [4, 5, 6]  with tag 2
process  1  received  [1, 2, 3]  with tag 1
```

Comunicación colectiva

Sincronización

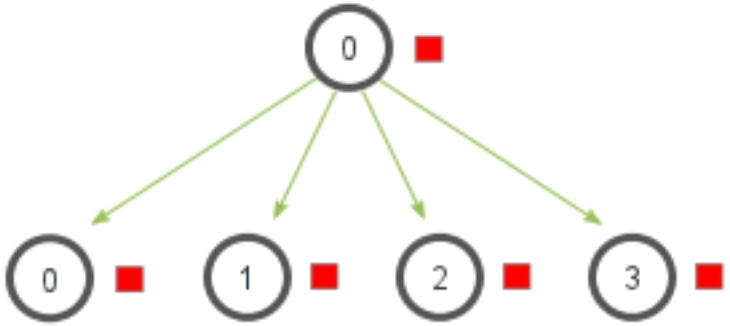
```
from mpi4py import MPI
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
        print (rank , " starting to wait”)
        comm.barrier()
        print (rank , " after barrier”)
elif rank == 1:
        print (rank , " starting to wait”)
        comm.barrier()
        print (rank , " after barrier”)
elif rank == 2:
        print (rank , " starting to wait”)
        time.sleep(4)
        comm.barrier()
        print (rank , " after barrier”)
```

El proceso que permite esperar hasta que todos los procesos lleguen a un punto.

```
mpiexec -n 3 python sincronizacion.py
2  starting to wait
1  starting to wait
0  starting to wait
1  after barrier
2  after barrier
0  after barrier
```

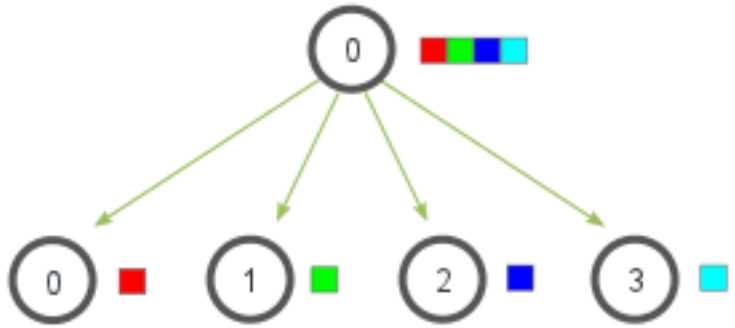
Sincronización

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
        data = {'key1' : [7, 2.72, 2+3j],
                'key2' : ( 'abc', 'xyz')}
else:
        data = None
data = comm.bcast(data, root=0)
print (rank, data)
```




```
mpiexec -n 3 python
broadcastPythonDictionary.py
(0, {'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]})
(1, {'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]})
(2, {'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]})
```

Scatter



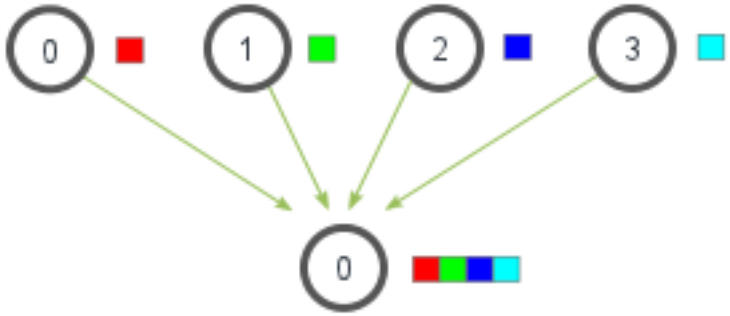

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
        data = [1, [3.5 , 5], "Hello"]
        print ('we will be scattering:',data)
else:
        data = None
data = comm.scatter(data, root=0)
print ('rank',rank,'has data:',data)
```

```
mpiexec -n 3 python scatteringPytgonObjects.py
we will be scattering: [1, [3.5, 5], 'Hello']
rank 0 has data: 1
rank 1 has data: [3.5, 5]
rank 2 has data: Hello
```

En este caso se necesitan 3 procesos ya que los data tiene 3 elementos

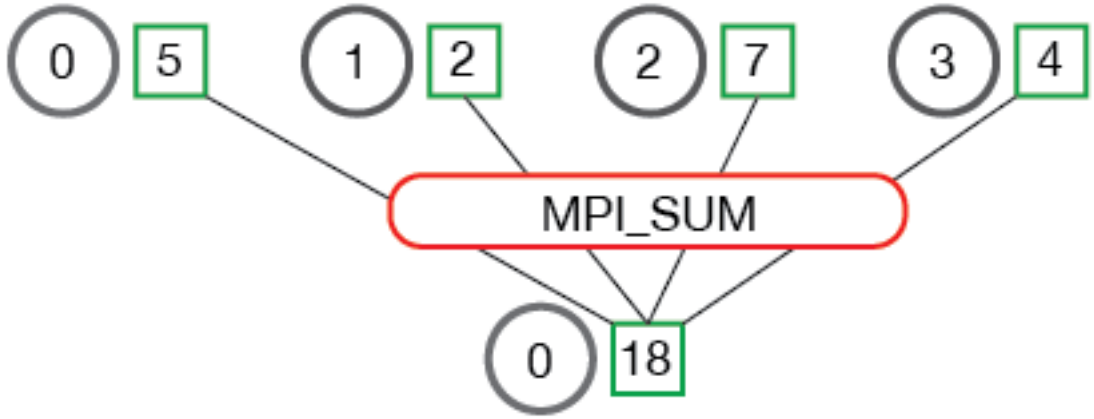
Gather

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

data = comm.gather(rank, root=0)
if rank == 0:
        print ('received ',data)
```




```
mpiexec -n 4 python gather.py
received  [0, 1, 2, 3]
```

Reduction



Reduce una lista de números distribuidos en diferentes nodos usando una función paralela.

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

a_size = 3
recvdata = np.zeros(a_size,dtype=np.int)
senddata = (rank+1)*np.arange(a_size,dtype=np.int)
print ('on task',rank,' sendData ',senddata)
comm.Reduce(senddata,recvdata,root=0,op=MPI.SUM)
print ('on task',rank,'after Reduce: data = ',recvdata)
```

```
mpiexec -n 4 python reduceSum.py
on task 2  sendData  [0 3 6]
on task 2 after Reduce:    data =  [0 0 0]
on task 1  sendData  [0 2 4]
on task 3  sendData  [0 4 8]
on task 0  sendData  [0 1 2]
on task 3 after Reduce:    data =  [0 0 0]
on task 1 after Reduce:    data =  [0 0 0]
on task 0 after Reduce:    data =  [ 0 10 20]
```

Sincronización

- Wait()

Espera que se complete una petición asíncrona.

- Barrier()

Espera a que todos los nodos llegen a este punto.

MPI4Py

Implementación de MPI en Python

http://pythonhosted.org/mpi4py/

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

Ejemplo simple

Envía desde el nodo 0, recibe en el nodo 1

```
from mpi4py import MPI

comm = MPI.COMM_WORLD  # Obtiene el objeto de comunicación
rank = comm.Get_rank()  # Obtiene el rango del nodo actual

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1, tag=11)  # Envía los dats al nodo 1
elif rank == 1:
    data = comm.recv(source=0, tag=11)  # Recibe datos del nodo 0
```

Ejemplo de broadcast

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'key1' : [7, 2.72, 2+3j],
                'key2' : ( 'abc', 'xyz')}
else:
    data = None
data = comm.bcast(data, root=0)
```

A tener en cuenta

Las funciones en minúscula trabajan con **objetos de Python.**

Las funciones en mayúscula trabajan con **buffers**.

Toda llamada a Send() debe tener una correspondiente llamada a Recv().