MPI - Continuação

Qual o output do programa:

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

if rank == 0:
    print ("Myrank: %s:" % rank)
    for proc in range(1, nprocs):
        msg = comm.recv( source = proc , tag=9 )
        print("Hello World from %s: " % proc)
else:
    msg = "Hello world from process" + str(rank)
    print ("Myrank: %s:" % rank)
    comm.send(msg, dest=0, tag=9);
```

MPI - Exemplos

Output:

```
>mpiexec -np 5 python msgTodos.py
```

```
Myrank: 4:
Myrank: 3:
Myrank: 2:
Myrank: 1:
Myrank: 0:
Hello World from  1:
Hello World from  2:
Hello World from  3:
Hello World from  4:
```

MPI - Exemplos

Deadlock

```
from mpi4py import MPI
comm=MPI.COMM_WORLD
rank = comm.rank

print("my rank is : " , rank)
if rank==1:
    data_send= "a"
    destination_process = 5
    source_process = 5
    data_received = comm.recv (source = source_process)
    comm.send( data_send, dest = destination_process )
    print ("sending data %s " %data_send + \
                                        "to process %d" %destination_process)
    print ("data received is = %s" %data_received)
```

MPI - Exemplos

Deadlock

```
if rank==5:
    data_send= "b"
    destination_process = 1
    source_process = 1
    data_received = comm.recv (source = source_process)
    comm.send( data_send, dest = destination_process )
    print ("sending data %s :" %data_send + \
                                        "to process %d" %destination_process)
    print ("data received is = %s" %data_received)
```

- O que acontece?

MPI - Exemplos

Deadlock

E se trocarmos as instruções:

```
if rank==1:
…
    comm.send( data_send, dest = destination_process )
    data_received = comm.recv (source = source_process)

if rank==5:

    data_received = comm.recv (source = source_process)
    comm.send( data_send, dest = destination_process )
```

- O que acontece?

MPI - Exemplos

Deadlock

```
>mpiexec -np 10 python deadLockProblems0.py
```

```
my rank is :  7
my rank is :  8
my rank is :  6
my rank is :  3
my rank is :  9
my rank is :  2
my rank is :  4
my rank is :  1
sending data a to process 5
data received is = b
my rank is :  5
sending data b :to process 1
data received is = a
my rank is :  0
```

MPI - Exemplos

Deadlock

E se for assim?

```
if rank==1:
    …
    comm.send( data_send, dest = destination_process )
    data_received = comm.recv (source = source_process)

if rank==5:
    …
    comm.send( data_send, dest = destination_process )
    data_received = comm.recv (source = source_process)
```

MPI - Exemplos

Instrução sendrecv:

```
Sendrecv(self, sendbuf, int dest, int
sendtag=0, recvbuf=None, int source=ANY_SOURCE, int
recvtag=ANY_TAG, Status status=None)
```

```
if rank==1:
    …
    data_received=comm.sendrecv(data_send,dest=destination_process,
                                source=source_process)
if rank==5:
    …
    data_received=comm.sendrecv(data_send,dest=destination_process,
                                source=source_process)
```

MPI

NumPY

Package para computação científica.

Permite declarar e manipular arrays:

```
import numpy as np
a = np.arange(20)
print (a)
print()
b= a.reshape(4,5)
print (b)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

MPI

Comunicar arrays:

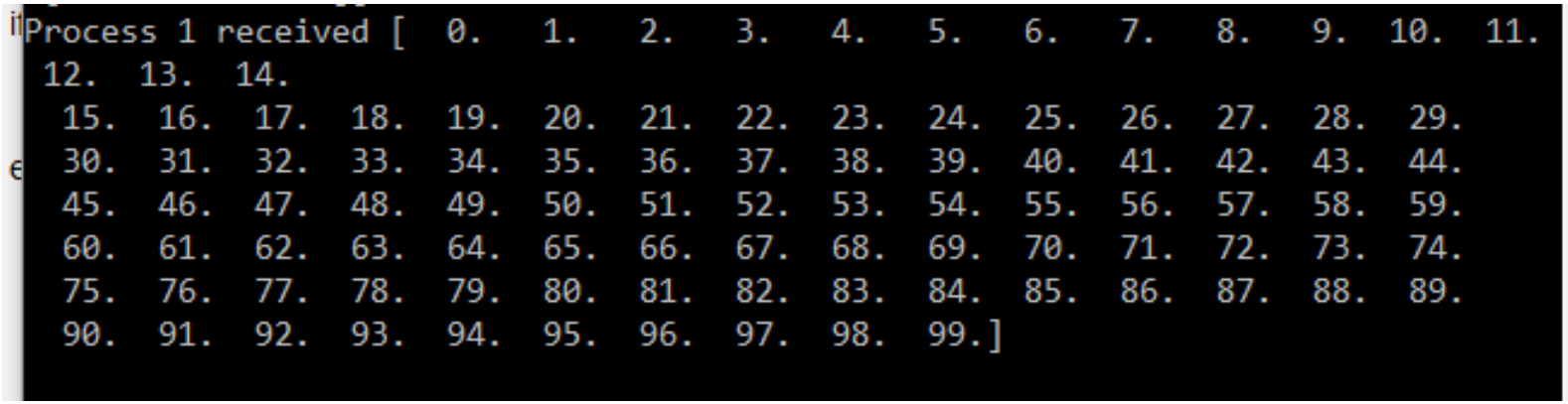
```
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
```

```
if rank ==0 :
    data = np.arange (100, dtype = np.float)
    comm.Send (data , dest=1, tag =10 )
elif rank == 1 :
    data = np.empty (100, dtype = np.float)
    comm.Recv ( data, source = 0, tag = 10)
    print ( "Process %s received %s " % (rank , data) )
```

MPI - Exemplos

Comunicar arrays:

Output



send / recv – objetos gerais python, lento

Send / Recv - arrays contínuos, mais rápido

MPI - Arrays

Comunicar arrays:

- Objetos python são convertidos para streams de bytes

antes do envio.

- No recetor as streams de bytes são novamente convertidas para objetos python.
- A conversão introduz overhead na comunicação.
- Arrays NumPy (contínuos) podem ser transmitidos com baixo overhead usando os métodos:

Send(data, dest, tag)

Recv(data, source, tag)

– Notar a diferença na receção: o array tem de já existir.

MPI - Arrays

Status

```
data = rank*np.ones(5,dtype = np.float64)  // 0 * (11111) = 0 0 0 0 0
                                           // 1 * (11111)= 1 1 1 1 1
print ("process: %s data: %s " % (rank, data))
if rank == 0:
    comm.Send([data,3,MPI.DOUBLE],dest=1,tag=1)
if rank == 1:
    info = MPI.Status()
    comm.Recv(data, MPI.ANY_SOURCE, MPI.ANY_TAG, info)
    source = info.Get_source()
    tag = info.Get_tag()
    count = info.Get_elements(MPI.DOUBLE) // nº de doubles
    size = info.Get_count() // nº de bytes
    print ("on %s process: source: %s tag: %s count: %s, size: %s" %
                                        (rank, source, tag, count, size))
    print (data)  ??
```

MPI - Arrays

Output:

```
process: 0 data: [ 0.  0.  0.  0.  0.]
process: 1 data: [ 1.  1.  1.  1.  1.]
on 1 process:  source: 0 tag: 1 count: 3, size: 24
[ 0.  0.  0.  1.  1.]
```

MPI - Arrays

Probe

```
if rank == 0:
    data = rank*np.ones(5,dtype = np.float64)
    comm.Send([data,3,MPI.DOUBLE],dest=1,tag=1)
if rank == 1:
     info = MPI.Status()
    comm.Probe(MPI.ANY_SOURCE, MPI.ANY_TAG, info)
    count = info.Get_elements(MPI.DOUBLE)
    data = np.empty(count, dtype = np.float64)  # !
    comm.Recv(data,MPI.ANY_SOURCE,MPI.ANY_TAG, info)
    print ("Process: %s data: %s " % (rank, data))
```

! Criar o buffer de receção com a dimensão dos dados a receber.

MPI
Operações de Comunicação Coletiva

broadcast

Um valor é enviado do processo root para todos os outros processos

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
    variable_to_share = 100
else:
    variable_to_share = None
variable_to_share = comm.bcast(variable_to_share, root=0)
print("process = %d" %rank + " variable shared = %d " %variable_to_share)
```

mpiexec –np 8 python exemplo.py Qual o output?

MPI
Operações de Comunicação Coletiva

```
process = 0 variable shared  = 100
process = 8 variable shared  = 100
process = 1 variable shared  = 100
process = 9 variable shared  = 100
process = 2 variable shared  = 100
process = 4 variable shared  = 100
process = 6 variable shared  = 100
process = 3 variable shared  = 100
process = 5 variable shared  = 100
process = 7 variable shared  = 100
```

MPI
Operações de Comunicação Coletiva

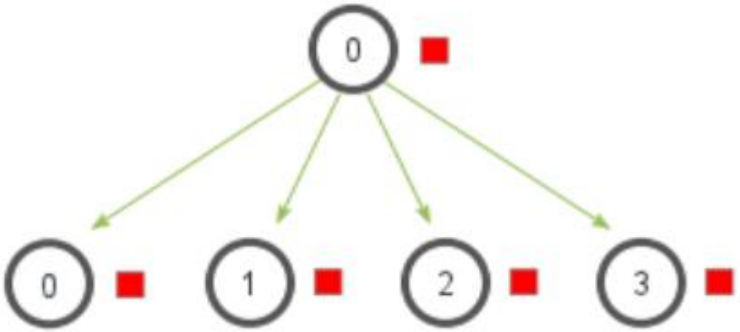
Broacast de um array:

```
data = np.empty(10, dtype = np.int )
if rank == 0:
    data = np.arange( 10, dtype = np.int)
comm.Bcast ( [data, 10 , MPI.INT] , root = 0)
print ("rank %s data: %s" % (rank, data))
```

output ?

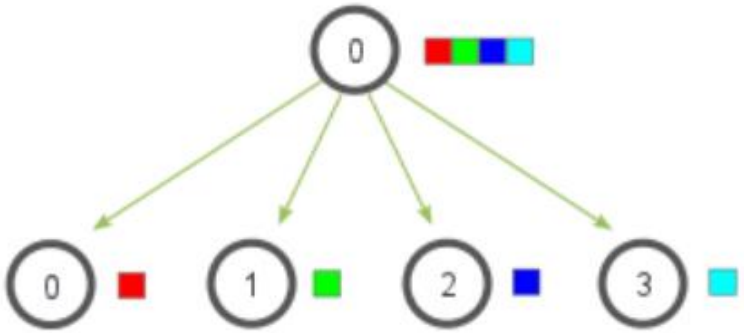
MPI
Operações de Comunicação Coletiva

MPI_Bcast



Difundir um elemento por todos os processos.

MPI_Scatter



Difundir os elementos de um array por todos os processos, seguindo a ordem do seu *rank* .

MPI
Operações de Comunicação Coletiva

scatter

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

if rank == 0:
    array_to_share = [1, 2, 3, 4 ,5 ,6 ,7, 8 ,9 ,10]
else:
    array_to_share = None
recvbuf = comm.scatter(array_to_share, root=0)
print("process = %d" %rank + " variable shared = %d " %recvbuf )
```

MPI
Operações de Comunicação Coletiva

mpiexec –np 10 python exemplo.py

```
process = 2 variable shared  = 3
process = 3 variable shared  = 4
process = 8 variable shared  = 9
process = 0 variable shared  = 1
process = 9 variable shared  = 10
process = 4 variable shared  = 5
process = 5 variable shared  = 6
process = 1 variable shared  = 2
process = 6 variable shared  = 7
process = 7 variable shared  = 8
```

MPI
Operações de Comunicação Coletiva

Scatter

```
…
size = comm.Get_size()
a_size = 4
recvdata = np.empty(a_size,dtype=np.float64)
senddata = None
if rank == 0:
    senddata = np.arange(size*a_size,dtype=np.float64)
comm.Scatter(senddata, recvdata,root=0)
print ("on process %s after Scatter: data = %s" % (rank, recvdata))
```

Output? **com mpiexec –np 2 python exemplo.py**

MPI
Operações de Comunicação Coletiva

Scatter

```
on process 0 after Scatter: data = [ 0.  1.  2.  3.]
on process 1 after Scatter: data = [ 4.  5.  6.  7.]
```

E com **mpiexec –np 4 python exemplo.py**

?

MPI
Operações de Comunicação Coletiva

Scatterv

```
size = comm.Get_size()
a_size = 4
senddata = None
recvdata = np.empty(a_size, dtype=np.float64)
counts = None
dspls = None
if rank == 0:
    senddata = np.arange(100,dtype=np.float64)
    counts=(1,2,3) <= tamanho de cada bloco
    dspls=(4,3,10)  <= posição inicial de cada bloco

comm.Scatterv([senddata,counts,dspls, MPI.DOUBLE],recvdata,root=0)
print ("X on process %s after Scatter: data = %s" % (rank, recvdata))
```

MPI
Operações de Comunicação Coletiva

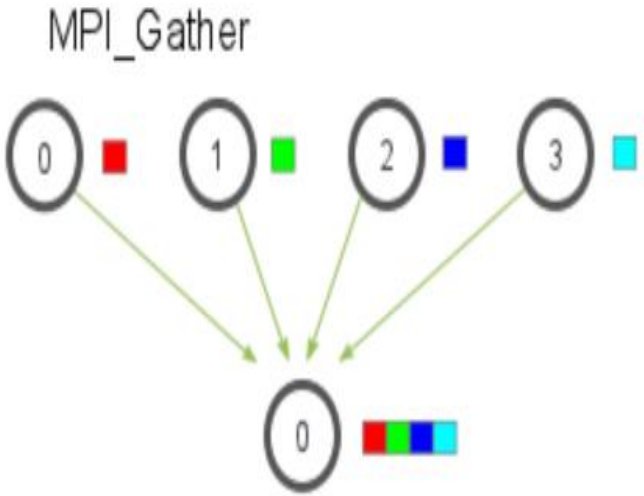
Scatterv

Output:

```
D:\_PPD_17_18-mestrado\pratica>mpiexec -np 3 python scatter.py
X on process 0 after Scatter: data = [ 4.  0.  0.  0.]
X on process 2 after Scatter: data = [ 10.  11.  12.   0.]
X on process 1 after Scatter: data = [ 3.  4.  0.  0.]
```

MPI
Operações de Comunicação Coletiva

Gather



Recolhe dados de todos os processo e reúne-os no processo *root*, ordenado pelo *rank* do processo origem.

MPI
Operações de Comunicação Coletiva

gather

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

data1 = (rank+1)**2

data = comm.gather(data1, root=0)
if rank == 0:
    print ("rank = %s " %rank + "...receiving data from other process")
    print (data)
```

mpiexec –np 10 python exemplo.py Qual o output?

MPI
Operações de Comunicação Coletiva

gather

Output:

```
rank = 0 ...receiving data from other process
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

MPI
Operações de Comunicação Coletiva

Gather (arrays)

```
from mpi4py import MPI
import numpy as np
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
```

Output com np = 3 ?

```
a_size = 4
recvdata = None
senddata = (rank+1)*np.arange(a_size, dtype=np.float64)
if rank == 0:
    recvdata = np.arange(size*a_size, dtype=np.float64)
comm.Gather(senddata,recvdata,root=0)
print ("on process %s, after Gather data = %s" % (rank, recvdata ))
```

MPI
Operações de Comunicação Coletiva

Gather (arrays)

Output: (com np = 3)

```
on process 1, after Gather  data = None
on process 2, after Gather  data = None
on process 0, after Gather  data = [ 0.  1.  2.  3.  0.  2.  4.  6.  0.  3.  6.  9.]
```

MPI
Operações de Comunicação Coletiva

Gatherv (arrays)

```
a_size = 4
recvdata = None
senddata = (rank+1)* np.arange(a_size,dtype=np.float64)
counts=(2,3,4)
dspls=(0,3,8)
if rank == 0:
    recvdata = np.zeros(size*a_size,dtype=np.float64)
sendbuf = [senddata, counts[rank] ]
recvbuf = [recvdata, counts, dspls, MPI.DOUBLE]
comm.Gatherv(sendbuf,recvbuf,root=0)
print ("on process %s, after Gather data = %s" % (rank, recvdata))
```

Output?

MPI
Operações de Comunicação Coletiva

Gatherv (arrays)

```
on process 1, after Gather data = None
on process 2, after Gather data = None
on process 0, after Gather data = [ 0.  1.  0.  0.  2.  4.  0.  0.  0.  3.  6.  9.]
```

MPI
Operações de Comunicação Coletiva

Alltoall (sendbuf, recvbuf)

O objeto da posição i, do processo j (sendbuf), é copiado para o objeto j do processo i (recvbuf).

MPI
Operações de Comunicação Coletiva

```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
a_size = 1
senddata = (rank+1)*numpy.arange(size,dtype=int)
recvdata = numpy.empty(size*a_size,dtype=int)
comm.Alltoall(senddata,recvdata)
print(" process %s sending %s receiving %s“  %(rank , senddata , recvdata))
```

MPI
Operações de Comunicação Coletiva

Alltoall (sendbuf, recvbuf)

```
D:\_PPD_17_18-mestrado\pratica>
D:\_PPD_17_18-mestrado\pratica>mpiexec -n 3 python alltoall.py
 process 2 sending [0 3 6] receiving [2 4 6]
 process 0 sending [0 1 2] receiving [0 0 0]
 process 1 sending [0 2 4] receiving [1 2 3]
```

Exemplos de aplicação:

- Transposição de matrizes;
- Transformadas de Fourier (Fast Fourier transform);
- Operações de ordenação;
- Operações de join em bases de dados;

MPI
Operações de Comunicação Coletiva

Explorar:

alltoallv (sendbuf, recvbuf)

e

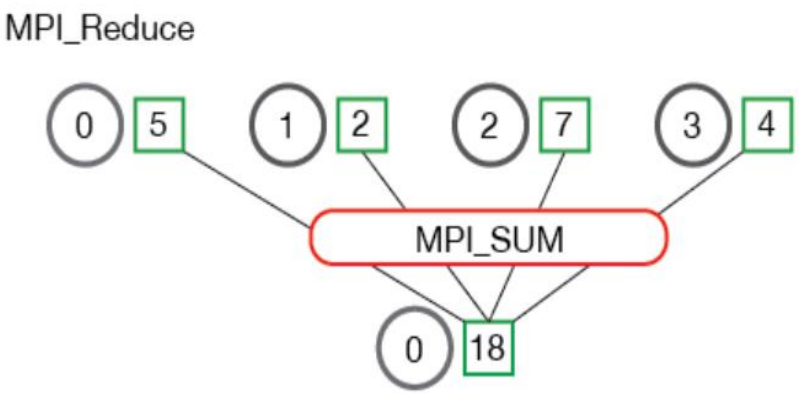
Alltoallv (sendbuf, recvbuf)

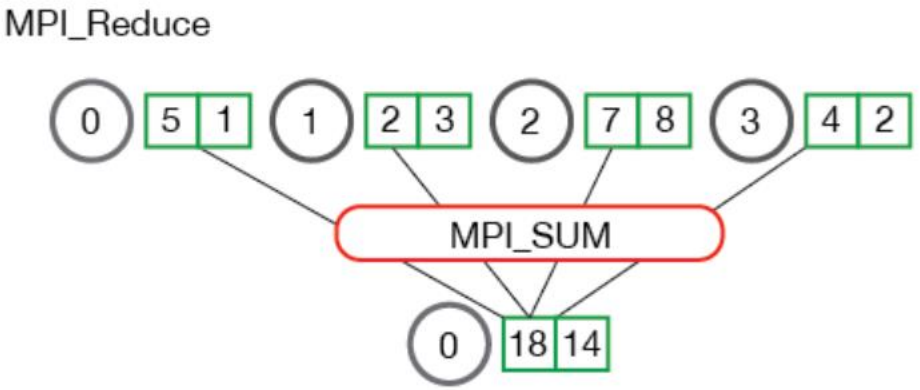
MPI
Operações de Comunicação Coletiva

Reduce

recvobj = comm.reduce(sendobj=None, op=MPI.SUM, root=0)

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





MPI
Operações de Comunicação Coletiva

Reduce - exemplo

```
import numpy as np
from mpi4py import MPI
comm = MPI.COMM_WORLD
size = comm.size
rank = comm.rank


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

MPI
Operações de Comunicação Coletiva

Reduce - exemplo

```
D:\_PPD_17_18-mestrado\pratica>mpiexec -np 4 python reduction.py
 process 3 sending [0 4 8]
on task 3 after Reduce:    data =  [0 0 0]
 process 1 sending [0 2 4]
on task 1 after Reduce:    data =  [0 0 0]
 process 2 sending [0 3 6]
on task 2 after Reduce:    data =  [0 0 0]
 process 0 sending [0 1 2]
on task 0 after Reduce:    data =  [ 0 10 20]
```

MPI
Operações de Comunicação Coletiva

Reduce – operações

MPI.MIN minimum
MPI.MAX maximum
MPI.SUM sum
MPI.PROD product
MPI.LAND logical and
MPI.BAND bitwise and
MPI.LOR logical or
MPI.BOR bitwise or
MPI.LXOR logical xor
MPI.BXOR bitwise xor
MPI.MAXLOC max value and location rank
MPI.MINLOC min value and location rank

MPI
Operações de Comunicação Coletiva

allreduce e Allreduce:

recvobj = comm.allreduce(sendobj=None, op=MPI.SUM)

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

(Distribuem os valores por todos os processos, equivalente a MPI_Reduce seguido de MPI_Bcast)

root?