Distributed Computing and Introduction to High Performance Computing

Imad Kissami[1]

[1]Mohammed VI Polytechnic University, Benguerir, Morocco

Outline of this lecture

Collective Communications

- Types of collective communications
- Global synchronization
- Global Reduction

General concepts

- Collective communications allow making a series of point-to-point communications in one single call.
- A collective communication always concerns all the processes of the indicated communicator.
- For each process, the call ends when its participation in the collective call is completed, in the sense of point-to-point communications (therefore, when the concerned memory area can be changed).
- The management of tags in these communications is transparent and system-dependent. Therefore, they are never explicitly defined during calls to subroutines. An advantage of this is that collective communications never interfere with point-to-point communications.

Collective Communications

Types of collective communications

- One which ensures global synchronizations : MPI_BARRIER()
- Ones which only transfer data :
  - Global distribution of data : MPI_BCAST()
  - Selective distribution of data : MPI_SCATTER()
  - Collection of distributed data : MPI_GATHER()
  - Collection of distributed data by all the processes : MPI_ALLGATHER()
  - Collection and selective distribution by all the processes of distributed data : MPI_ALLTOALL()
- Ones which, in addition to the communications management, carry out operations on the transferred data :
  - Reduction operations (sum, product, maximum, minimum, etc.), whether of a predefined or personal type : MPI_REDUCE()
  - Reduction operations with distributing of the result (this is in fact equivalent to an MPI_REDUCE() followed by an MPI_BCAST()) : MPI_ALLREDUCE()

Collective Communications

Global synchronization : MPI_BARRIER()

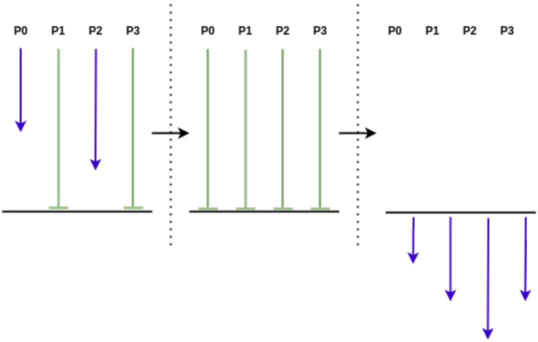

Figure: Global Synchronization : MPI_BARRIER()

```
COMM.Barrier()
```

Collective Communications

Global synchronization : MPI_BCAST()

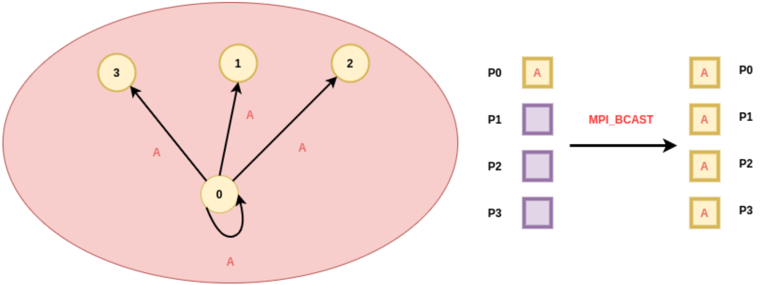

Figure: Global distribution : MPI_BCAST()

Collective Communications

Global synchronization : MPI_BCAST()

```fortran
MPI_BCAST(buffer, count, datatype, root, comm, code)

<type> :: buffer
integer :: count, datatype, root, comm, code
```

- Send, starting at position buffer, a message of count element of type datatype, by the root process, to all the members of communicator comm.
- Receive this message at position buffer for all the processes other than the root.

```python
Comm.bcast(self, obj, int root=0)
# or
Comm.Bcast(self, buf, int root=0)
```

Collective Communications

Global synchronization : MPI_BCAST() Full example 1

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

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

if RANK == 0:
  sendbuf = {'key1' : [7, 2.72, 2+3j]}
else:
  sendbuf = None

recvbuf = COMM.bcast(sendbuf, root=0)

print("I am the process {rank}, I received data {data} from 2".format(rank=RANK
    , data=recvbuf))
```

```
mpirun -n 4 python bcast.py
```

```
I am the process 0, I received data {'key1': [7, 2.72, (2+3j)]} from 2
I am the process 2, I received data {'key1': [7, 2.72, (2+3j)]} from 2
I am the process 1, I received data {'key1': [7, 2.72, (2+3j)]} from 2
I am the process 3, I received data {'key1': [7, 2.72, (2+3j)]} from 2
```

Collective Communications

Global synchronization : MPI_BCAST() Full example 2

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

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

if RANK == 0:
    sendbuf = np.arange(10, dtype='i')
else:
    sendbuf = np.empty(10, dtype='i')

COMM.Bcast(sendbuf, root=0)

print("I am the process {rank}, I received data {data} from 2".format(rank=
    RANK, data=sendbuf))
```

```
mpirun -n 4 python bcast.py

I am the process 1, I received data [0 1 2 3 4 5 6 7 8 9] from 2
I am the process 3, I received data [0 1 2 3 4 5 6 7 8 9] from 2
I am the process 0, I received data [0 1 2 3 4 5 6 7 8 9] from 2
I am the process 2, I received data [0 1 2 3 4 5 6 7 8 9] from 2
```

Collective Communications

Global synchronization : MPI_SCATTER()

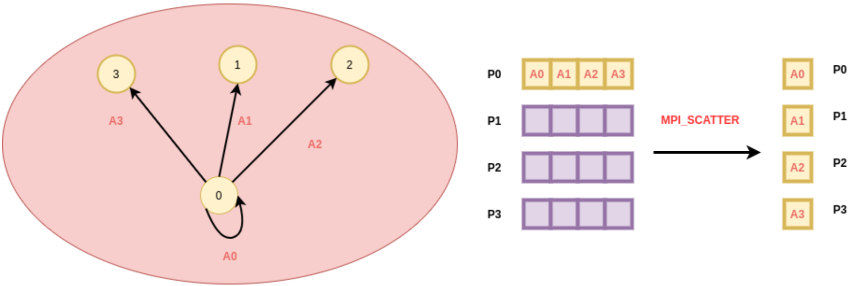

Figure: Selected distribution : MPI_SCATTER()

Collective Communications

Global synchronization : MPI_SCATTER()

```
MPI_SCATTER(message_a_repartir,longueur_message_emis,↩
     type_message_emis,
            message_recu,longueur_message_recu,type_message_recu,↩
                 rang_source,comm,code)

<type> :: sendbuf, recvbuf
integer :: sendcount, recvcount
integer :: sendtype, recvtype
integer :: root, comm, code
```

- Scatter by process root, starting at position sendbuf, message sendcount element of type sendtype, to all the processes of communicator comm.
- Receive this message at position recvbuf, of recvcount element of type recvtype for all processes of communicator comm.
- ➠ The couples (sendcount, sendtype) and (recvcount, recvtype) must represent the same quantity of data.
- ➠ Data are scattered in chunks of same size ; a chunk consists of sendcount elements of type sendtype.
- ➠ The i-th chunk is sent to the i-th process.

```
Comm.scatter(self, sendobj, int root=0)
#or
Comm.Scatter(self, sendbuf, recvbuf, int root=0)
```

Global synchronization : MPI_SCATTER() Full example

```python

from mpi4py import MPI
import numpy as np


COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()

if RANK == 0:
  sendbuf = [(i+1)**2 for i in range(SIZE)]
  print("The data will be scattered is", sendbuf)
else:
  sendbuf = None

recvbuf = COMM.scatter(sendbuf, root=0)
assert recvbuf == (RANK+1)**2
```

```
mpirun -n 4 python scatter.py
```

```
The data will be scattered is [1, 4, 9, 16]
I am the process 0, I received data 1 from 0
I am the process 3, I received data 16 from 0
I am the process 1, I received data 4 from 0
I am the process 2, I received data 9 from 0
```

Collective Communications

Global synchronization : MPI_GATHER()

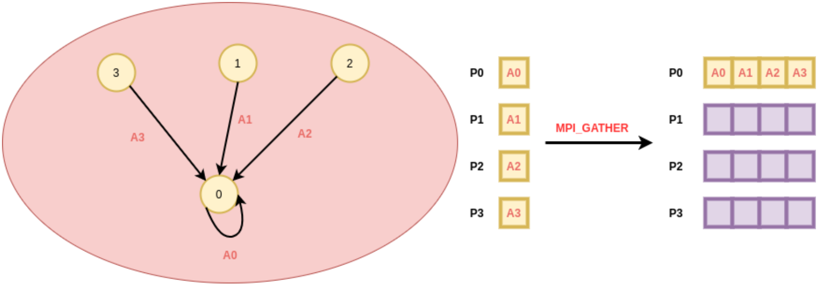

Figure: Collection : MPI_GATHER()

Collective Communications

Global synchronization : MPI_GATHER()

```
MPI_GATHER(sendbuf,sendcount,sendtype,
           recvbuf,recvcount,recvtype,root,comm,code)

<type> :: sendbuf, recvbuf
integer :: sendcount, recvcount
integer :: sendtype, recvtype
integer :: root, comm, code
```

- Send for each process of communicator comm, a message starting at position sendbuf, of sendcount element type sendtype
- Collect all these messages by the root process at position recvbuf, recvcount element of type recvtype.
- ➠ The couples (sendcount, sendtype) and (recvcount, recvtype) must represent the same size of data.
- ➠ The data are collected in the order of the process ranks.

```
Comm.gather(self, sendbuf, root=0)
#or
Comm.Gather(self, sendbuf, recvbuf, int root=0)
```

Collective Communications

Global synchronization : MPI_GATHER() Full example

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


COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()

sendbuf = (RANK+1)**2
print("I am the process {rank}, I send data {data} to 0".format(rank = RANK, data=sendbuf))

recvbuf = COMM.gather(sendbuf, root=0)

if RANK == 0:
  for i in range(SIZE):
      assert recvbuf[i] == (i+1)**2

  print("I am the process 0, I received data {data}".format(data=recvbuf))
```

Collective Communications

Global synchronization : MPI_GATHER() Full example

```
mpirun -n 4 python gather.py

I am the process 1, I send data 4 to 0
I am the process 0, I send data 1 to 0
I am the process 2, I send data 9 to 0
I am the process 3, I send data 16 to 0

I am the process 0, I received data [1, 4, 9, 16]
```

Collective Communications

Global synchronization : MPI_ALLGATHER()

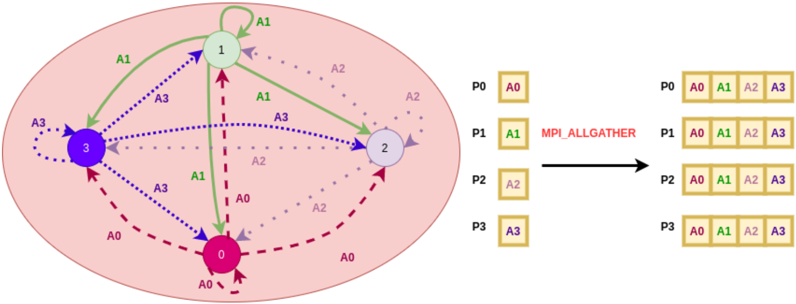

Figure: Gather-to-all : MPI_ALLGATHER()

Collective Communications

Global synchronization : MPI_ALLGATHER()

```
MPI_ALLGATHER(sendbuf,sendcount,sendtype,
              recvbuf,recvcount,recvtype,comm,code)

<type> :: sendbuf, recvbuf
integer :: sendcount, recvcount
integer :: sendtype, recvtype
integer :: comm, code
```

- Corresponds to an MPI_GATHER() followed by an MPI_BCAST()
- Send by each process of communicator comm, a message starting at position sendbuf, of sendcount element, type sendtype.
- Collect all these messages, by all the processes, at position recvbuf of recvcount element type recvtype.

➠ The couples (sendcount, sendtype) and (recvcount, recvtype) must represent the same data size.

➠ The data are gathered in the order of the process ranks.

```
Comm.Allgather(self, sendbuf, recvbuf)
#or
Comm.allgather(self, sendobj)
```

Global synchronization : MPI_ALLGATHER() Full example

```
from mpi4py import MPI
import numpy as np


COMM = MPI.COMM_WORLD
SIZE = COMM.Get_size()
RANK = COMM.Get_rank()

sendbuf = np.array((RANK+1)**2)
print("I am the process {rank}, I send data {data} to all".format(rank = RANK, data=sendbuf))

recvbuf = np.zeros(SIZE, dtype=np.int)

COMM.Allgather([sendbuf, MPI.INT], [recvbuf, MPI.INT])


print("I am the process {rank}, I received data {data}".format(rank = RANK, data=recvbuf))
```

Collective Communications

Global synchronization : MPI_ALLGATHER() Full example

```
I am the process 0, I send data 1 to all
I am the process 3, I send data 16 to all
I am the process 1, I send data 4 to all
I am the process 2, I send data 9 to all

I am the process 1, I received data [ 1  4  9 16]
I am the process 2, I received data [ 1  4  9 16]
I am the process 3, I received data [ 1  4  9 16]
I am the process 0, I received data [ 1  4  9 16]
```

Collective Communications

Global synchronization : MPI_GATHERV()

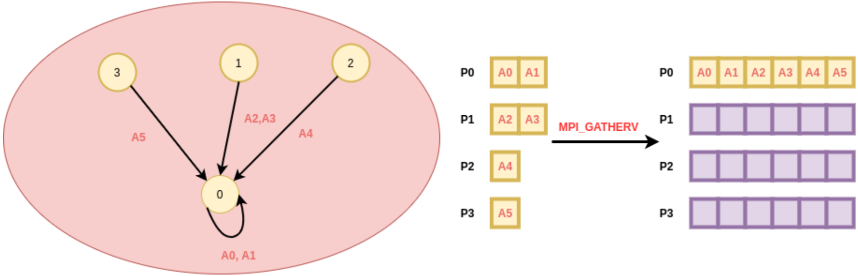

Figure: Extended gather : MPI_GATHERV()

Collective Communications

Global synchronization : MPI_GATHERV()

```
MPI_GATHERV(sendbuf,sendcount,sendtype,
            recvbuf,recvcounts,displs,recvtype,
            root,comm,code)

<type > :: sendbuf, recvbuf
integer :: sendcount
integer :: sendtype, recvtype
integer, dimension(:) :: recvcounts, displs
integer :: root, comm, code
```

- This is an MPI_GATHER() where the size of messages can be different among processes
- The i-th process of the communicator comm sends to process root, a message starting at position sendbuf, of sendcount element of type sendtype, and receives at position recvbuf, of recvcounts(i) element of type recvtype, with a displacement of displs(i)
- ➠ The couples (sendcount,sendtype) of the i-th process and (recvcounts(i), recvtype) of process root must be such that the data size sent and received is the same.

```
Comm.Gatherv(self, sendbuf, recvbuf, int root=0)
```

Collective Communications

Global synchronization : MPI_GATHERV() Full example

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

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

local_array = [RANK] * random.randint(2, 5)
print("rank: {}, local_array: {}".format(RANK, local_array))

sendbuf = np.array(local_array)

# Collect local array sizes using the high-level mpi4py gather
sendcounts = np.array(COMM.gather(len(sendbuf), root=0))

if RANK == 0:
    print("sendcounts: {}, total: {}".format(sendcounts, sum(sendcounts)))
    recvbuf = np.empty(sum(sendcounts), dtype=int)
else:
    recvbuf = None

COMM.Gatherv(sendbuf=sendbuf, recvbuf=(recvbuf, sendcounts), root=0)
if RANK == 0:
    print("Gathered array: {}".format(recvbuf))
```

Collective Communications

Global synchronization : MPI_GATHERV() Full example

```
mpirun -n 4 python gatherv.py

rank: 0, local_array: [0, 0, 0, 0, 0]
rank: 3, local_array: [3, 3, 3, 3, 3]
rank: 1, local_array: [1, 1, 1]
rank: 2, local_array: [2, 2, 2]

sendcounts: [5 3 3 5], total: 16
Gathered array: [0 0 0 0 0 1 1 1 2 2 2 3 3 3 3 3]
```

Collective Communications

Global synchronization : MPI_ALLTOALL()

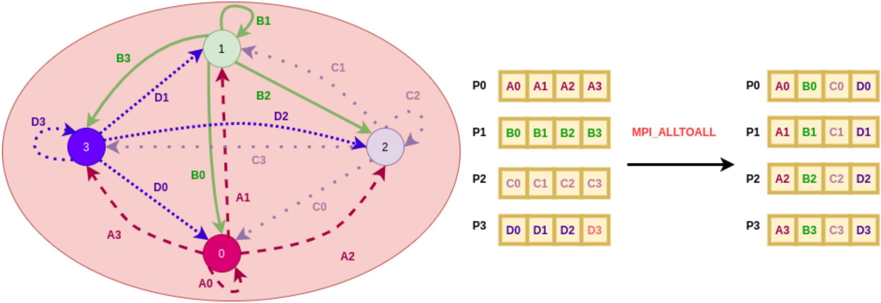

Figure: Collection and distribution : : MPI_ALLTOALL()

Collective Communications

Global synchronization : MPI_ALLTOALL()

```fortran
MPI_ALLTOALL(sendbuf,sendcount,sendtype,
             recvbuf,recvcount,recvtype,comm,code)

<type> :: sendbuf, recvbuf
integer :: sendcount, recvcount
integer :: sendtype, recvtype
integer :: comm, code
```

- the i-th process sends its j-th chunk to the j-th process which places it in its i-th chunk.
- The couples (sendcount, sendtype) and (recvcount, recvtype) must be such that they represent equal data sizes.

```python
Comm.Alltoall(self, sendbuf, recvbuf)
#or
Comm.alltoall(self, sendobj)
```

Collective Communications

Global synchronization : MPI_ALLTOALL() Full example

```python
from mpi4py import MPI

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

nb_values=4
sendbuf = [1000.+RANK*nb_values+i for i in range(nb_values)]
print("I am the process {rank}, I send data {data} ".format(rank = RANK, data=sendbuf))

recvbuf = COMM.alltoall(sendbuf)
print("I am the process {rank}, I received data {data}".format(rank = RANK, data=recvbuf))
```

```
mpirun -n 4 python alltoall.py

I am the process 1, I send data [1004.0, 1005.0, 1006.0, 1007.0]
I am the process 2, I send data [1008.0, 1009.0, 1010.0, 1011.0]
I am the process 0, I send data [1000.0, 1001.0, 1002.0, 1003.0]
I am the process 3, I send data [1012.0, 1013.0, 1014.0, 1015.0]

I am the process 2, I received data [1002.0, 1006.0, 1010.0, 1014.0]
I am the process 1, I received data [1001.0, 1005.0, 1009.0, 1013.0]
I am the process 0, I received data [1000.0, 1004.0, 1008.0, 1012.0]
I am the process 3, I received data [1003.0, 1007.0, 1011.0, 1015.0]
```

Collective Communications

Global Reduction

- A reduction is an operation applied to a set of elements in order to obtain one single value. Typical examples are the sum of the elements of a vector (SUM(A(:))) or the search for the maximum value element in a vector (MAX(V(:))).
- MPI proposes high-level subroutines in order to operate reductions on data distributed on a group of processes. The result is obtained on only one process (MPI_REDUCE()) or on all the processes (MPI_ALLREDUCE() , which is in fact equivalent to an MPI_REDUCE() followed by an MPI_BCAST()).
- If several elements are implied by process, the reduction function is applied to each one of them (for instance to each element of a vector).

Collective Communications

Global Reduction

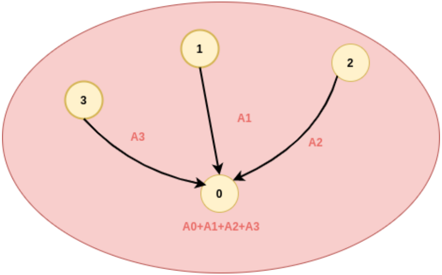

Figure: Distributed reduction (sum)

Collective Communications

Global Reduction: Operations

Name	Operation
MPI_SUM	Sum of elements
MPI_PROD	Product of elements
MPI_MAX	Maximum of elements
MPI_MIN	Minimum of elements
MPI_MAXLOC	Maximum of elements and location
MPI_MINLOC	Minimum of elements and location
MPI_LAND	Logical AND
MPI_LOR	Logical OR
MPI_LXOR	Logical exclusive OR

Table: Global Reduction available operations

Collective Communications

Global Reduction: MPI_REDUCE()

```
MPI_REDUCE(sendbuf,recvbuf,count,datatype,op,root,comm,code)

<type> :: sendbuf, recvbuf
integer :: count, datatype, root
integer :: op, comm, code
```

- Distributed reduction of count elements of type datatype, starting at position sendbuf, with the operation op from each process of the communicator comm,
- Return the result at position recvbuf in the process root

```
Comm.Reduce(self, sendbuf, recvbuf, Op op=SUM, int root=0)
#or
Comm.reduce(self, sendobj, op=SUM, int root=0)
```

Collective Communications

Global Reduction: MPI_REDUCE() Full example

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

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

np.random.seed(RANK)

data = np.random.randint(2, 10)

print("The data of rank {rank} is {data}".format(rank=RANK, data=data))

max_reduced_data = COMM.reduce(data, op=MPI.MAX, root=0)
min_reduced_data = COMM.reduce(data, op=MPI.MIN, root=0)
sum_reduced_data = COMM.reduce(data, op=MPI.SUM, root=0)
prod_reduced_data = COMM.reduce(data, op=MPI.PROD, root=0)


if RANK==0:
    print("I, process",RANK,"the max of data is :", max_reduced_data)
    print("I, process",RANK,"the min of data is :", min_reduced_data)
    print("I, process",RANK,"the sum of data is :", sum_reduced_data)
    print("I, process",RANK,"the product of data is :", prod_reduced_data)
```

Global Reduction: MPI_REDUCE() Full example

```
mpirun -n 4 python reduce.py

The data of rank 0 is 6
The data of rank 1 is 7
The data of rank 2 is 2
The data of rank 3 is 4

I, process 0 the max of data is : 7
I, process 0 the min of data is : 2
I, process 0 the sum of data is : 19
I, process 0 the product of data is : 336
```

Collective Communications

Global Reduction: MPI_ALLREDUCE()

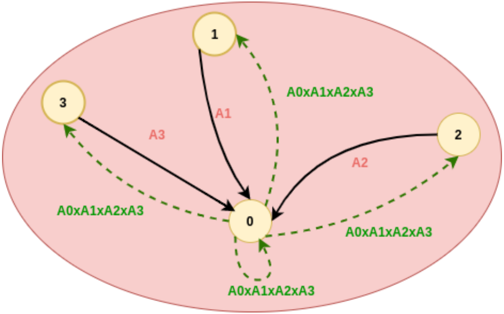

Figure: Distributed reduction (product) with distribution of the result

Collective Communications

Global Reduction: MPI_ALLREDUCE()

```
MPI_ALLREDUCE(sendbuf,recvbuf,count,datatype,op,comm,code)

<type> :: sendbuf, recvbuf
integer :: count, datatype
integer :: op, comm, code
```

- Distributed reduction of count elements of type datatype starting at position sendbuf, with the operation op from each process of the communicator comm,
- Write the result at position recvbuf for all the processes of the communicator comm.

```
Comm.Allreduce(self, sendbuf, recvbuf, Op op=SUM)
#or
Comm.allreduce(self, sendobj, op=SUM)
```

Collective Communications

Global Reduction: MPI_ALLREDUCE() Full example

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

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()

np.random.seed(RANK)

data = np.random.randint(2, 10)

print("The data of rank {rank} is {data}".format(rank=RANK, data=data))

max_reduced_data = COMM.allreduce(data, op=MPI.MAX)

print("I, process {RANK}, The max of data is {max_reduced_data}:".format(RANK=RANK, max_reduced_data=max_reduced_data))
```

Collective Communications

Global Reduction: MPI_ALLREDUCE() Full example

```
mpirun -n 4 python allreduce.py

The data of rank 1 is 7
The data of rank 3 is 4
The data of rank 0 is 6
The data of rank 2 is 2

I, process 3, The max of data is 7
I, process 0, The max of data is 7
I, process 1, The max of data is 7
I, process 2, The max of data is 7
```