Introduction to MPI and Domain decomposition

Ji-Hoon Kang, jhkang@kisti.re.kr

Korea Institute of Science and technology (KISTI)

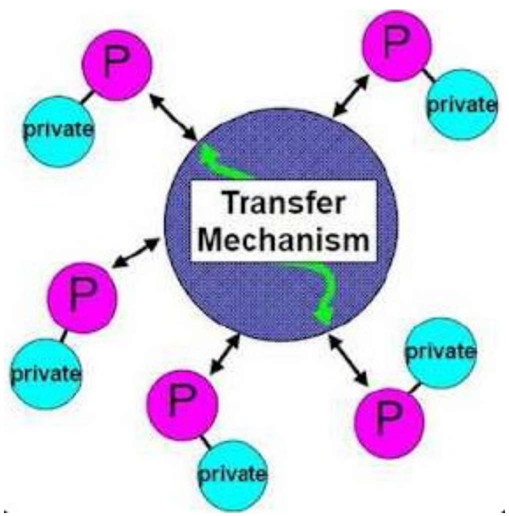
Contents

KISTI 한국과학기술정보연구원 Korea Institute of Science and Technology Information www.kisti.re.kr

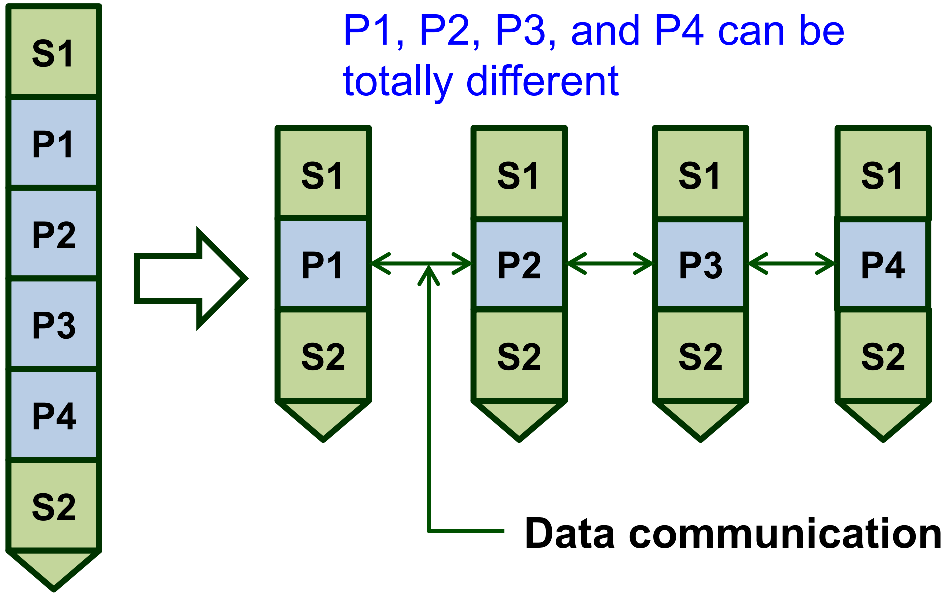
Review with exercise

MPI programming model

- **Message passing parallelism**



MPI: multi-processes



- Process based
  - Independent processes runs on many multi-core processors and work together using their own memory and resources **through message-passing communication**.
- Distributed memory model
  - Each process does its own work using its own memory and resources
  - In order to work together, data in memory are passed through communication

- **MPI only provides the tools of communication**

mpi4py

MPI for Python

- Python bindings for the Message Passing Interface (MPI) standard
- Allowing Python applications to exploit multiple processors
- Providing an object oriented interface resembling the MPI-2 C++ bindings
- Supporting P2P and collective communications.
- Handling python objects serialized with pickle module, as well as exposed to Python buffer interface of array data (e.g. NumPy arrays and built-in bytes/array/memory view objects).

MPI-2 bindings for C++ to Python

- Anyone using the standard C/C++ MPI bindings is able to use mpi4py module without need of learning a new interface.

Six (Four) key functions

Function	Functions
~~**MPI_INIT**~~	~~**Register communicator (address system)**~~
~~**MPI_FINALIZE**~~	~~**Destroy communicator**~~
MPI.COMM_WORLD.Get_size()	Return communicator size (size of address site)
MPI.COMM_WORLD.Get_rank()	Return process number (address)
MPI.COMM_WORLD.Send()	Send data/message to target process Send: numpy array, send: python object
MPI.COMM_WORLD.Recv()	Recv data/message from source process Recv: numpy array, recv: python object

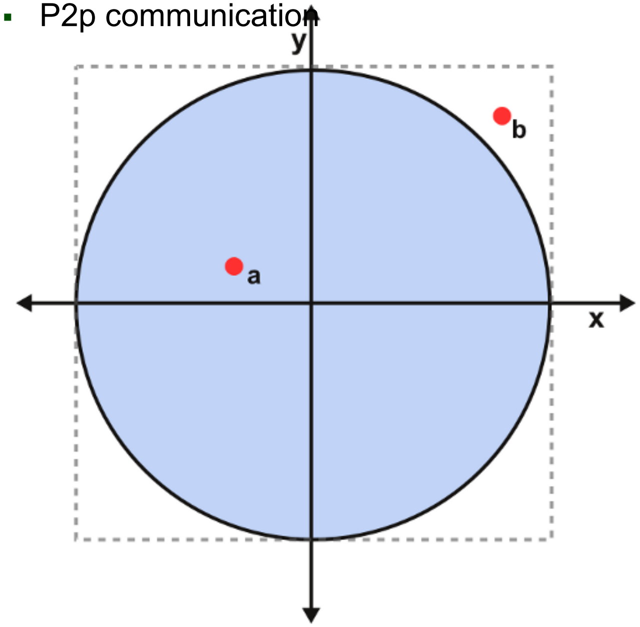
Exercise 1 – Estimating Pi with Monte Carlo method

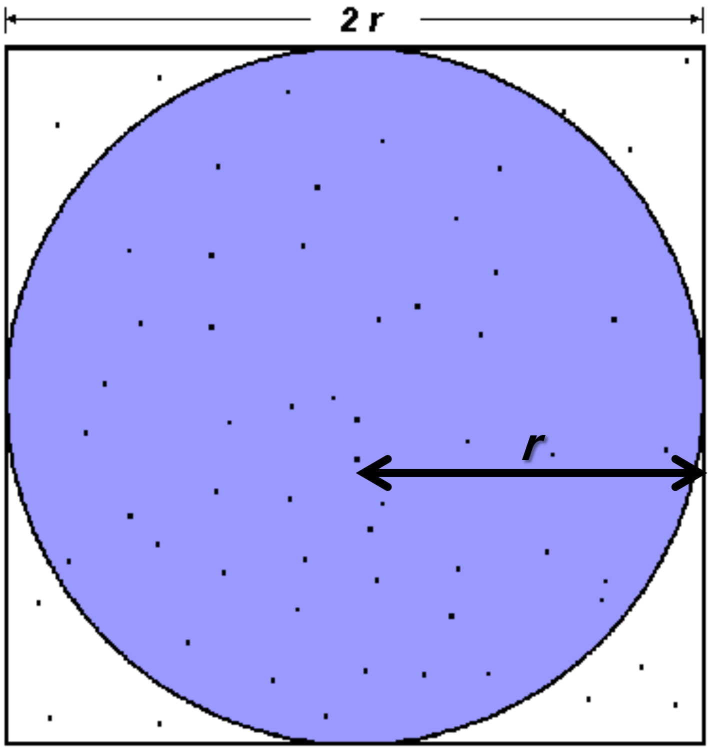
<Problem>

- Monte carlo simulation
- Random number use
- ***PI = 4 x Ac/As***

<Requirement>

- N's processor(rank) use
- P2p communication





$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \ x \ \frac{A_C}{A_S}$$

Exercise 1 – Serial code

```
import numpy as np

SCOPE = 1000000

count = 0

for i in range(SCOPE) :
    x = np.random.rand( )
    y = np.random.rand( )
    z = (x*x + y*y)**(0.5)
    if z < 1 :
        count += 1

print('Count = %d, Pi = %f'%(count,count/SCOPE*4))
```

Exercise 1– Parallel code

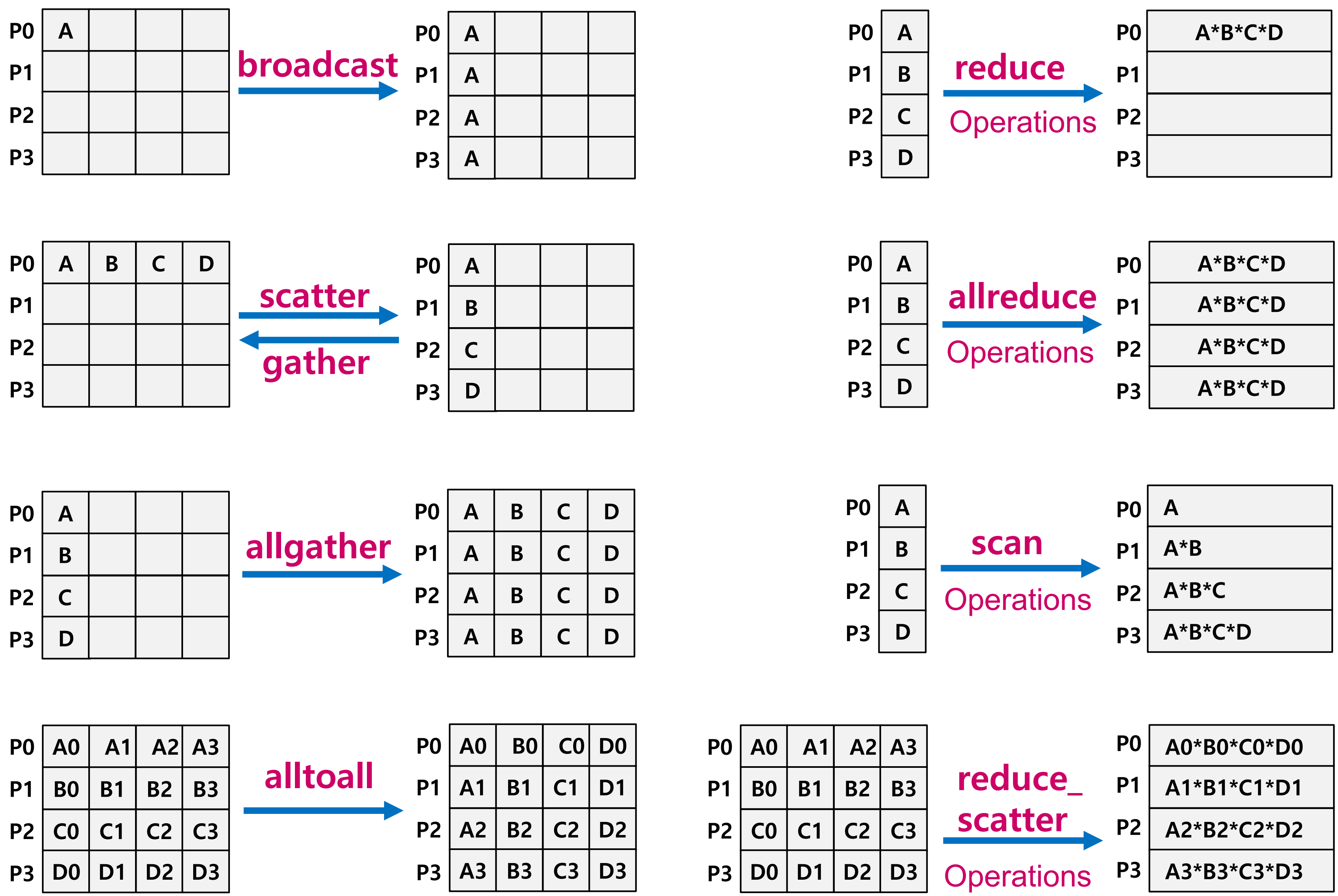
```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

SCOPE = 1000000

mycount = 0
for i in range(SCOPE) :
    x = np.random.rand()
    y = np.random.rand()
    z = (x*x + y*y)**(0.5)
    if z < 1 :
        mycount += 1

# Send mycount to rank 0 and sum

if rank == 0 :
    print('Rank : %d, Count = %d, Pi = %f'%(rank,count,count/SCOPE/size*4))
```

Collective communication

Collective communication

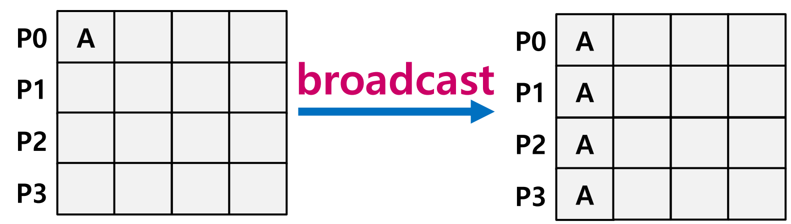
- **A group of processes participate in the communication**
- **Based on Point to Point communication**
- **More efficient, better performance than P2P communications**
- **Special feature**
  - All processes in the communicator group must be called
  - No message tag

Collective communication – schematics



Broadcast (lab9)

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

comm = MPI.COMM_WORLD

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

ROOT = 0

buf =  np.zeros(4, dtype = int)
buf2 = np.zeros(4, dtype = int)

if rank == ROOT :
    buf = np.array([5, 6, 7, 8])

if rank == (size - 1) :
    buf2 = np.array([50, 60, 70, 80])

print('Before : rank = {0}, buf = {1}'.format(rank, buf))

comm.Bcast(buf, ROOT)
comm.Bcast(buf, size-1)

print('After  : rank = {0}, buf = {1}'.format(rank, buf))
print('After  : rank = {0}, buf2 = {1}'.format(rank, buf2))
```



comm.Bcast(*buf, root=0*)

```
$ mpirun -np 4 python3 lab9_bcast.py
Before : rank = 0, buf = [5 6 7 8]
Before : rank = 3, buf = [0 0 0 0]
Before : rank = 2, buf = [0 0 0 0]
Before : rank = 1, buf = [0 0 0 0]
After  : rank = 1, buf = [5 6 7 8]
After  : rank = 1, buf2 = [50 60 70 80]
After  : rank = 2, buf = [5 6 7 8]
After  : rank = 2, buf2 = [50 60 70 80]
After  : rank = 3, buf = [5 6 7 8]
After  : rank = 3, buf2 = [50 60 70 80]
After  : rank = 0, buf = [5 6 7 8]
After  : rank = 0, buf2 = [50 60 70 80]
```

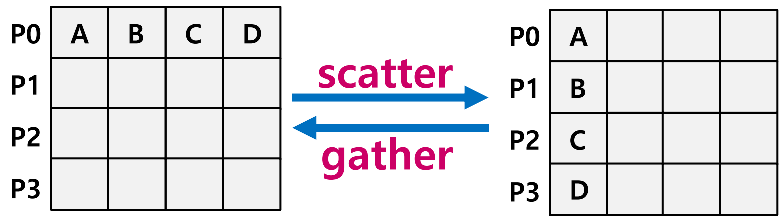
Gather (lab10)

comm.Gather(*sbuf, rbuf, root=0*)

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

comm = MPI.COMM_WORLD

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

isend = np.array([rank + 1], dtype = int)
irecv = np.zeros(size, dtype = int)

print('rank = {0}, isend = {1}'.format(rank, isend))
ROOT = 0
comm.Gather(isend, irecv, ROOT)
print('rank = {0}, irecv = {1}'.format(rank, irecv))
```

```
$ mpirun -np 4 python3 lab10_gather.py
rank = 2, isend = [3]
rank = 0, isend = [1]
rank = 1, isend = [2]
rank = 3, isend = [4]
rank = 3, irecv = [0 0 0 0]
rank = 1, irecv = [0 0 0 0]
rank = 2, irecv = [0 0 0 0]
rank = 0, irecv = [1 2 3 4]
```

Scatter (lab15)



comm.Scatter(*sbuf, rbuf, root=0*)

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

isend = np.zeros(size, dtype = int)
irecv = np.empty(1, dtype = int)

if rank == 0 :
    isend = np.arange(0, size, dtype = int)

print('sbuf : rank = {0}, irecv = {1}'.format(rank, isend))
comm.Scatter(isend, irecv, 0)
print('rbuf : rank = {0}, irecv = {1}'.format(rank, irecv))
```

```
$ mpirun -np 4 python3 lab15_scatter.py
sbuf : rank = 0, irecv = [0 1 2 3]
rbuf : rank = 0, irecv = [0]
sbuf : rank = 3, irecv = [0 0 0 0]
sbuf : rank = 2, irecv = [0 0 0 0]
sbuf : rank = 1, irecv = [0 0 0 0]
rbuf : rank = 2, irecv = [2]
rbuf : rank = 1, irecv = [1]
rbuf : rank = 3, irecv = [3]
```

Allgather (lab13)

comm.Allgather(*sbuf, rbuf*)

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

comm = MPI.COMM_WORLD

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

isend = np.array([rank + 1])
irecv = np.zeros(size, dtype = int)

print('rank = {0}, isend = {1}'.format(rank, isend))
comm.Allgather(isend, irecv)
print('rank = {0}, irecv = {1}'.format(rank, irecv))
```

```
mpirun -np 4 python3 lab13_allgather.py
rank = 0, isend = [1]
rank = 1, isend = [2]
rank = 2, isend = [3]
rank = 3, isend = [4]
rank = 3, irecv = [1 2 3 4]
rank = 2, irecv = [1 2 3 4]
rank = 1, irecv = [1 2 3 4]
rank = 0, irecv = [1 2 3 4]
```

Alltoall (lab18)

P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

alltoall →

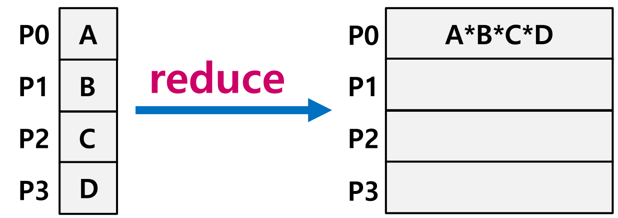
P0	A0	B0	C0	D0
P1	A1	B1	C1	D1
P2	A2	B2	C2	D2
P3	A3	B3	C3	D3

comm.Alltoall(*sbuf, rbuf*)

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

isend = np.arange(1 + size * rank, 1 + size * rank + size, dtype = int)
irecv = np.zeros(size, dtype = int)
print('Rank({0}) : isend = {1}'.format(rank, isend))

comm.Alltoall(isend, irecv)

print('Rank({0}) : irecv = {1}'.format(rank, irecv))
```

```
mpirun -np 4 python3 lab18_alltoall.py
Rank(1) : isend = [5 6 7 8]
Rank(3) : isend = [13 14 15 16]
Rank(2) : isend = [ 9 10 11 12]
Rank(0) : isend = [1 2 3 4]
Rank(0) : irecv = [ 1  5  9 13]
Rank(3) : irecv = [ 4  8 12 16]
Rank(2) : irecv = [ 3  7 11 15]
Rank(1) : irecv = [ 2  6 10 14]
```

Reduce



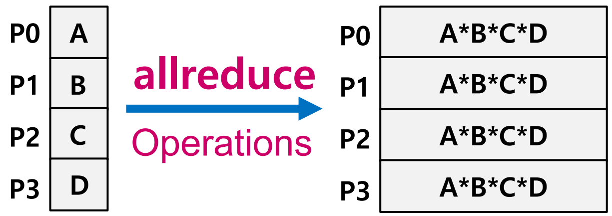
comm.Reduce(*sbuf, rbuf,* op, root=0)

Operation
MPI_SUM(sum), MPI_PROD(product) MPI_MAX(maximum), MPI_MIN(minimum)
MPI_MAXLOC(max value and location), MPI_MINLOC(min value and location)
MPI_LAND(logical AND), MPI_LOR(logical OR), MPI_LXOR(logical XOR)
MPI_BAND(bitwise AND), MPI_BOR(bitwise OR), MPI_BXOR(bitwise XOR)

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

comm = MPI.COMM_WORLD

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

a = np.zeros(9, dtype = int)

ista = rank * 3
iend = ista + 3

a[ista:iend] = np.arange(ista + 1, iend + 1)
sum = a.sum()
print('Rank({0}) : local_sum = {1}'.format(rank, sum))
tsum = np.zeros_like(sum)

comm.Reduce(sum, tsum, MPI.SUM, 0)

if rank == 0 :
    print('Rank({0}) : sum = {1}'.format(rank, tsum))
```

```
mpirun -np 3 python3 lab17_reduce.py
Rank(1) : local_sum = 15
Rank(0) : local_sum = 6
Rank(2) : local_sum = 24
Rank(0) : sum = 45
```

Allreduce

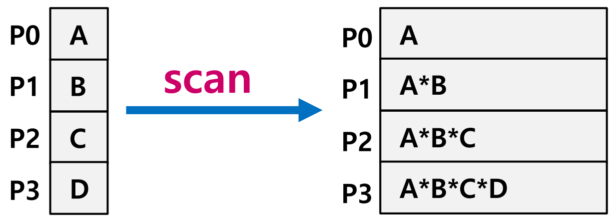


comm.AllReduce(*sbuf, rbuf,* op)

Operation
MPI_SUM(sum), MPI_PROD(product) MPI_MAX(maximum), MPI_MIN(minimum)
MPI_MAXLOC(max value and location), MPI_MINLOC(min value and location)
MPI_LAND(logical AND), MPI_LOR(logical OR), MPI_LXOR(logical XOR)
MPI_BAND(bitwise AND), MPI_BOR(bitwise OR), MPI_BXOR(bitwise XOR)

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

a = np.zeros(9, dtype = int)

ista = rank * 3
iend = ista + 3

a[ista:iend] = np.arange(ista + 1, iend + 1)
sum = a.sum()
tsum = np.zeros_like(sum)

comm.Allreduce(sum, tsum, MPI.SUM)

if rank == 1 :
    print('Rank({0}) : sum = {1}'.format(rank, tsum))
```

Scan (lab22)



comm.Scan(*sbuf, rbuf,* op)

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

comm = MPI.COMM_WORLD

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

local = random.randint(2, 5)

print("rank: {}, local: {}".format(rank, local))

scan = comm.scan(local, MPI.SUM)

print ("rank:", rank, "sum: ", scan)
```

```
mpirun -np 3 python3 lab22_scan.py
rank: 0, local: 3
rank: 2, local: 4
rank: 1, local: 3
rank: 1 sum:  6
rank: 0 sum:  3
rank: 2 sum:  10
```

Reduce & scatter

P0	A0	A1	A2	A3
P1	B0	B1	B2	B3
P2	C0	C1	C2	C3
P3	D0	D1	D2	D3

reduce_scatter →

P0	A0*B0*C0*D0
P1	A1*B1*C1*D1
P2	A2*B2*C2*D2
P3	A3*B3*C3*D3

comm.Reduce_scatter(*sbuf, rbuf, rcounts,* op)

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

comm = MPI.COMM_WORLD

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

sendbuf = np.array([1, 2, 3], dtype = int)
recvbuf = np.zeros(1, dtype = int)
RECVBUF = sendbuf * 2

comm.Reduce_scatter(sendbuf, recvbuf, None, MPI.SUM)

print('Rank({0}) : recvbuf = {1}'.format(rank, recvbuf))
```

```
mpirun -np 3 python3 lab20_reduce_scatter.py
Rank(0) : sendbuf = [1 2 3]
Rank(2) : sendbuf = [1 2 3]
Rank(1) : sendbuf = [1 2 3]
Rank(1) : recvbuf = [6]
Rank(2) : recvbuf = [9]
Rank(0) : recvbuf = [3]
```

Exercise 1 again

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

SCOPE = 1000000

mycount = 0
for i in range(SCOPE) :
    x = np.random.rand()
    y = np.random.rand()
    z = (x*x + y*y)**(0.5)
    if z < 1 :
        mycount += 1

# Reduce results to rank 0

if rank == 0 :
    print('Rank : %d, Count = %d, Pi = %f'%(rank,count,count/SCOPE/size*4))
```

Mores on collective communication

comm.Gatherv(*sbuf, rbuf=(rbuf, scount*), root)

comm.Scatterv(*sbuf=(sbuf,scount), rbuf*, root)

comm.Allgatherv(*sbuf, rbuf=(rbuf, scount*))

comm.Alltoallv(*sbuf=(sbuf,scount), rbuf=(rbuf, scount*))

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

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

isend = np.array([1, 2, 2, 3, 3, 3])
irecv = np.zeros(3 * (rank + 1), dtype = int)
iscnt = np.array([1, 2, 3])
ircnt = np.full(3, rank + 1, dtype = int)
isend += size * rank

comm.Alltoallv((isend, iscnt), (irecv, ircnt))
print('Rank({0}) : isend = {1}'.format(rank, isend))
print('Rank({0}) : irecv = {1}'.format(rank, irecv))
```

Loop parallelization

- Loop is frequently found in computer algorithm.

```
for i in range(1, N - 1) :
    for j in range(1, M - 1) :
        psi_new[i, j] = beta_1 * (psi_old[i, j + 1] + psi_old[i, j - 1] \
                     + beta * beta * (psi_old[i + 1, j] + psi_old[i - 1, j]))
```

- Loop parallelization is essential for most computational problem.

iteration	1	2	3	4	5	6	7	8	9	10	11	12
rank	0	0	0	1	1	1	2	2	2	3	3	3

Loop parallelization type

- **Block distribution**

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	0	0	1	1	1	1	2	2	2	3	3	3

- **Cyclic distribution**

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	1	2	3	0	1	2	3	0	1	2	3	0	1

```
DO i = n1, n2
   computation
ENDDO
```

```
DO i = n1+myrank, n2, nprocs
   computation
ENDDO
```

- **Block-cyclic distribution**

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	1	1	2	2	3	3	0	0	1	1	2	2

iblock

Block distribution – para_range

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Rank	0	0	0	0	1	1	1	1	2	2	2	3	3	3

- **Suppose when you divide n by p, the quotient is q and the remainder is r.**
  - n = p X q + r
- **Processes 0..r-1 are assigned q + 1 iterations each. The other processes are assigned q iterations.**
  - n = r(q+1) + (p-r)q

```
def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1

    return ista, iend
```

Exercise 2 – Numerical integration

<Problem>

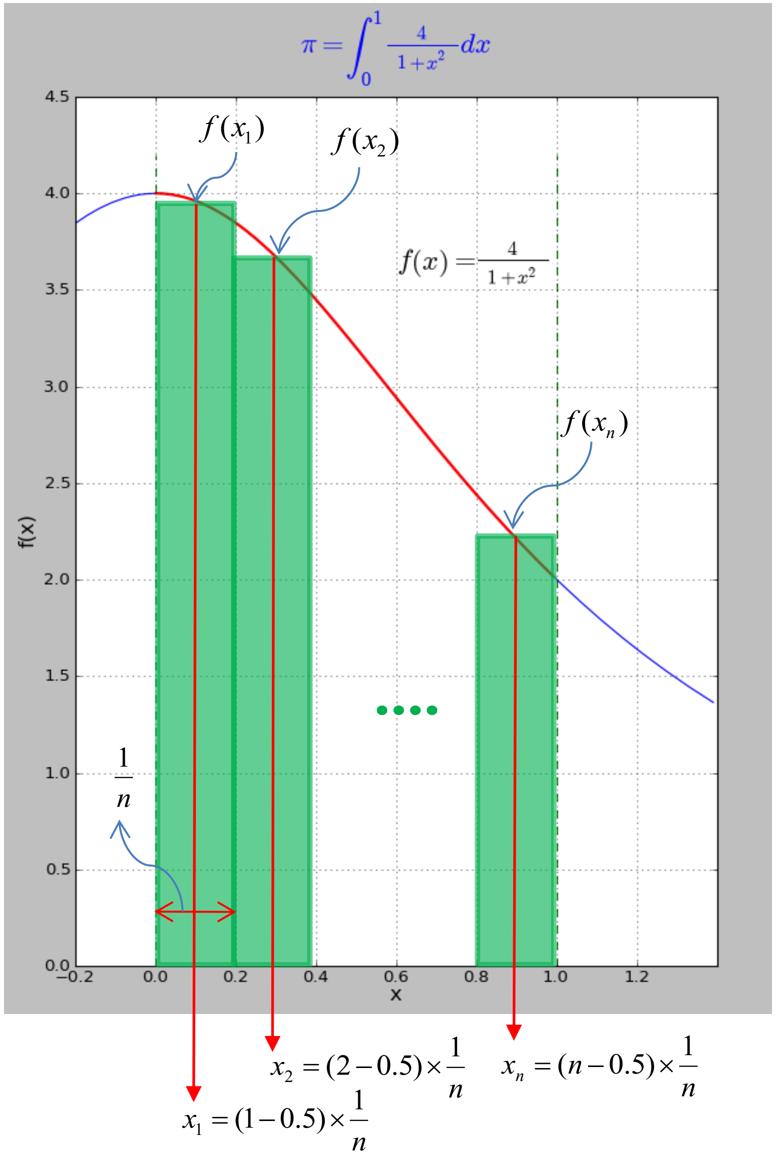
- Get PI using Numerical integration

$$\int_0^1 \frac{4.0}{(1+x^2)}\, dx = \pi$$

<Requirement>

- Point to point communication

$$\pi \approx \sum_{i=1}^{n} \frac{4}{1+((i-0.5)\times\frac{1}{n})^2} \times \frac{1}{n}$$



Exercise 2 – Serial code

```
num_step = 1000000

dx = 1.0 / num_step


sum = 0.0
for i in range(0, num_step) :
x= (i + 0.5) * dx
sum += 4.0/(1.0 + x*x)

pi = dx * sum
print('Numerical pi = %f'%pi)
```

Exercise 2 – Parallel code

```
from mpi4py import MPI

comm = MPI.COMM_WORLD

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

num_step = 1000000

dx = 1.0 / num_step

# Block distribution : ista, iend

print('Rank = %d, (ista, iend) = (%d, %d)'%(rank, ista, iend))

sum = 0.0
for i in range(ista, iend+1) :
    x= (i + 0.5) * dx
    sum += 4.0/(1.0 + x*x)

# Reduce results to rank 0

if rank == 0 :
    pi = dx * total_sum
    print('Numerical pi = %f'%pi)
```

Exercise 3 – matrix multiplication

```python
import numpy as np
import random as rd

NP = 5

matrixA = np.zeros((NP, NP), dtype = np.int32)
matrixB = np.zeros((NP, NP), dtype = np.int32)
matrixC = np.zeros((NP, NP), dtype = np.int32)
matrixT = np.zeros((NP, NP), dtype = np.int32)

for i in range(NP) :
    for j in range(NP) :
        matrixA[i][j] = rd.randrange(1, 10)
        matrixB[i][j] = rd.randrange(1, 10)

print(matrixA)
print(matrixB)

for k in range(NP) :
    for j in range(NP) :
        for i in range(NP) :
            matrixC[k][j] = matrixC[k][j] + matrixA[k][i] * matrixB[i][j]

print('Matrix C =')
print(matrixC)
print('Matrix A * B = ')
print(matrixA@matrixB)
```

Exercise 3 – parallelization with 5 procs.

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

comm = MPI.COMM_WORLD

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

NP = 5

matrixA = np.zeros((NP, NP), dtype = np.int32)
matrixB = np.zeros((NP, NP), dtype = np.int32)
matrixC = np.zeros((NP, NP), dtype = np.int32)
matrixT = np.zeros((NP, NP), dtype = np.int32)

# only for rank 0
if rank == 0 :
    for i in range(NP) :
        for j in range(NP) :
            matrixA[i][j] = rd.randrange(1, 10)
            matrixB[i][j] = rd.randrange(1, 10)

    print(matrixA)
    print(matrixB)

# Broadcast A and B
```