1 Objective

To get familiar with Message Passing Interface using collective communication; broadcast and scatter.

2 Introduction

MPI, the Message Passing Interface, is a standardized, portable message-passing system designed to work on a wide variety of parallel computers and is most suitable for computers with Distributed memory architectures. In this section, we will look into MPI for python. MPI for python is a general-purpose and full-featured package targeting the development of parallel applications in python. Its implementation specification is completely based on MPI-2 C++ bindings. It is implemented with Cython. MPI for python contains the twofold, higher level being the Python and lower level C for handling MPI types and calls.

Below are the some of the features available in MPI for python

1. Blocking point-to-point communications using e Send(), Recv() and Sendrecv() methods.

2. Nonblocking point-to-point communications using Isend() and Irecv() methods.
3. Collective communications using Bcast(), Scatter(), Gather(), Allgather() and Alltoall() methods.

4. Dynamic process management.

5. One-sided operations and Parallel input/output operations (through instance of File).

**Collective Communication** is a communication that involves group of processors where each process executes the same operations. In this assignment, we will work on broadcast and scatter. The figure below explains what broadcast and scatter is.

![Figure 1: Collective Communication- broadcast, scatter, gather (Pic - mpi-forum.org)](image)

The python MPI library MPI4py supports two types of communication: generic python objects and python buffer like objects (Numpy array) which are allocated in contiguous memory. The lowercase methods like bcast,recv,scatter allows to communicate generic objects while the initial uppercase of the same (Send,Bcast) can communicate memory buffer and is advisable for buffer objects to avoid performance loss. A special binary representation is created with generic objects communication, introducing overhead.
Suggested Reading / References


Note: While executing task, run with at least 3 processors. The provided sample test cases are to help you and passing it does not mean everything is fine. We employ additional test to verify the correctness.

3 Assignment

Task 1 (5 Points): Below is an example of broadcast using python objects.

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

if mpi_rank == 0:
    data = {'key1' : [7, 2.72, 2+3j],
            'key2' : ('abc', 'xyz')}
else:
    data = None

data = comm.bcast(data, root=0)
```

Listing 1: Broadcasting Python Object

Now write a program with `broadcast_prime_list(root)` function that generates the list of all prime numbers less than 100, broadcast it to all other
processes and return list of primes on each process. Note - The root (i.e. rank = 0) generates list of primes.

Listing 2: Sample Test Case

**Task 2 (5 Points):** Below is an example to broadcast Numpy array.

Listing 3: Broadcasting Numpy array

In this task you should modify your `broadcast_prime_list(root)` from task 1 to broadcast Numpy array containing primes.
from nose.tools import assert_equal
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank_m = comm.Get_rank()
size_m = comm.Get_size()

if size_m <= 2:
    print('Start at least 3 engines!')
else:
    primes=broadcast_prime_array(0)
    if rank_m == 0:
        assert_equal(primes[0],2)

Listing 4: Sample Test Case

Task 3 (5 Points): Based on the example below of scattering python objects, write a program with function `scatter_prime_list(root)` which

- on root generates list containing the first prime numbers equal to length of nproc=comm.Get_size() i.e number of processor.

- And scatter the result to rest of the processor.

Note: If you are trying to scatter a different number of objects than nproc, the program is likely to deadlock.
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
nproc = comm.Get_size()
if rank == 0:
    data = [(i+1)**2 for i in range(nproc)]
    if len(data) != nproc:
        print('NB: we need exactly', nproc, 'objects here!')
        data = None
else:
    data = None
data = comm.scatter(data, root=0)
print(data)

Listing 5: Scatter Example

from nose.tools import assert_equal
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank_m = comm.Get_rank()
size_m = comm.Get_size()
if size_m <= 2:
    print('Start at least 3 engines!')
else:
    prime = scatter_prime_list(0)
    if rank_m == 0:
        assert_equal(prime, 2)

Listing 6: Sample Test Case - Scatter

Task 4 (5 Points): Based on the example provided below to scatter a Numpy array, modify your scatter_prime_list(root) from previous task to scatter the Numpy array. The Numpy array here should consist exactly 5*nproc prime numbers.
from mpi4py import MPI
import numpy as np

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

sendbuf = None
if rank == 0:
    sendbuf = np.empty([nproc, 10], dtype='i')
    sendbuf.T[:, :] = range(nproc)
recvbuf = np.empty(10, dtype='i')
comm.Scatter(sendbuf, recvbuf, root=0)
print(recvbuf, rank)

Listing 7: Example Scattering Numpy array

from nose.tools import assert_equal
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank_m = comm.Get_rank()
size_m = comm.Get_size()

if size_m <= 2:
    print('Start at least 3 engines!')
else:
    prime5 = scatter_prime_array5(0)
    if rank_m == 0:
        assert_equal(prime5[4], 11)

Listing 8: Sample Test Case - Scattering Numpy

Task 5 (10 Points): Parallel Matrix Multiplication

Given two matrices:
A $n \times k$ matrix) and $B$ ($k \times m$ matrix)

The goal is to write the function `par_matmul(A,B,root)`, which calculates matrix multiplication in parallel $C = AB$.

Actual values of matrices $A$ and $B$ are initially known only on process `root`. Matrix $B$ will be broadcasted to all processes. Matrix $A$ will be split between processes $0,...,\text{nproc-1}$ block-row-wise, where process $i$ will get $A[i/\text{nproc} : (i+1)/\text{nproc} , :]$. (This is actually a natural way of splitting to achieve with `comm.Scatter(...)`) The resulting matrix $C$ will remain distributed between the processes as shown on the figure for case $np=3$:

![Figure 2: C = AxB](image)

4 Submission

Each task should contain a separate file, each named with `User-ID_task-Number`. You should also include the test case along with your program. You can find your user id in ois which looks like BXXX

5 Testing Help

We do not use `nose tests` to run the test this time, rather run it as general MPI program with test included.
from nose.tools import assert_equal
from mpi4py import MPI

def parallel_test(root=0):
    comm = MPI.COMM_WORLD
    mpi_rank = comm.Get_rank()

    numberList = {"num":[1,2,3,4,5,6]}

    numberList = comm.bcast(numberList, root=0)
    return numberList

comm = MPI.COMM_WORLD
rank_m = comm.Get_rank()
size_m = comm.Get_size()

if size_m <= 2:
    print('Start at least 3 engines!')
else:
    parallel_test_result = parallel_test(0)["num"]

    if rank_m == 0:
        assert_equal(parallel_test_result[0], 2)

Figure 3: Running Test