Continuing with Introduction to MPI - Message Passing Interface

mpi4py - Collective Communication - Gather operation

Assignment

The task is to play around with the basic MPI commands in given examples and do corresponding coding exercises in meanwhile.

Collective communication (continued)

Gather operation

EXAMPLE - Gathering numpy arrays:

```python
from mpi4py import MPI
import numpy as np
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
sendbuf = np.zeros(100, dtype='i') + rank
recvbuf = None
if rank == 0:
    recvbuf = np.empty([size, 100], dtype='i')
comm.Gather(sendbuf, recvbuf, root=0)
if rank == 0:
    for i in range(size):
        assert np.allclose(recvbuf[i, :], i)
```
Task 4.1 {10 points}

4.1.1. {3 points} For startup, write a function \texttt{DigitSum(k)} which returns the sum of digits of the integer \(k\) \textit{recursively} (in base 10), until it reaches a \textit{single-digit number}. See e.g. http://www.applet-magic.com/Digitsum.htm

4.1.2. {7 points} Write a function called \texttt{Parallel_DigitSums(a,n,root)}, which on process \texttt{root} scatters (like in Assignment A3 Task 3.4) a given 2D integer numpy array of shape \((\text{size},n)\) (\textit{size} being the number of processes) to all processors (with ranks \([0,1,...,\text{size}-1]\)).

- \texttt{DigitSum} is calculated in parallel for the whole array
- The results get gathered back to the process with \texttt{rank=root}
- Process \texttt{root} returns the gathered array of DigitSums, other processes return \texttt{None}.

A HINT

For testing you may want to generate (on \texttt{root}) an integer array as follows:

\begin{verbatim}
arr = numpy.random.randint(1,10**16,[size,n]) # (size,n)-shaped array of random integers from the range [1, 10**16)
\end{verbatim}

- But make sure you create buffers for receiving arrays in this case with the same datatype:
  \begin{verbatim}
  dtyp='int64':
  recvbuf=numpy.empty(n,dtype='int64')
  \end{verbatim}

```python
from nose.tools import assert_equal
from mpi4py import MPI
import numpy
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
if size <= 2:
    print('Start at least 3 engines!')
else:
    if rank==0:
        a=numpy.random.randint(1,10**7,[size,2])
        #print(a)
    else:
        a=None
    s=Parallel_DigitSums(a,2,0)
    if rank==0:
        #print(a)
        for i in range(size):
            for j in range(2):
                #print(s[i,j],s[i,j])
                #We make use of this special property of DigitSums for performing the check:
                if s[i,j]<0:
                    assert_equal(s[i,j],s[i,j])
                else:
                    assert_equal(s[i,j],0)
        else:
            assert_equal(s,None)
```
EXAMPLE - Gathering Python objects:

```python
from mpi4py import MPI
import numpy as np
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
sendbuf = np.zeros(100, dtype='i') + rank
recvbuf = None
if rank == 0:
    recvbuf = np.empty([size, 100], dtype='i')
comm.Gather(sendbuf, recvbuf, root=0)
if rank == 0:
    for i in range(size):
        assert np.allclose(recvbuf[i, :], i)
```

Task 4.2 {10 points}

4.2.1 Modify your function `DigitSum(n)` from Task 4.1 into the function `DigitSumsList(n)`, which returns the list of integers `[n,...,DigitSum(n)]` -- the original number `n` followed by all the intermediate and the final value while calculating the single-digit answer.

4.2.2 Write a function `GenerateDigitSumsLists(a,root)`, which for a given positive integer array `a` with shape `(size,n)` -- defined on process `rank==root` -- scatters the array to all processes (like in A3 either Task 3.3 or 3.4); calculates in parallel the `DigitSumsList(a[rank,:])`; (note that here the array in the argument is can be interpreted as separate calls to the function with elements from the array) and gathers the generated lists back to `root` process.

Some HINTs that possibly might help

(read: that might possibly lead you to one possible way for doing this exercise, but you are totally free to choose your own way for achieving the goal...):

- A way to form a (1-element) list from a single number (which is not yet as a member of a list construct):

  ```python
  if type(a) is not list:
      a=[a]
  ```

- T append x to this list:

  ```python
  a. append(x)
  ```
• accessing the last element of the list:

\[ a[-1] \]

**Task 4.3 {10 points} Totally digit-prime numbers and their trees**

**Definition:**

A prime number \( x \) is said to be belonging to the set of **totally digit-prime numbers** if all the members in \( \text{DigitSumsList}(x) \) are prime numbers.

**For example:**

- Prime number 593 is not totally digit-prime (due to \( \text{DigitSumsList}(593) \) giving \([593, 17, 8]\)).
- But 599 is totally digit-prime (as \( \text{DigitSumsList}(599) \rightarrow [599, 23, 5] \)).

**Task: Building trees of totally digit-prime numbers**

Modify your solution to the previous task to make a function \( \text{HarvestTotalPrimes}(a,b) \), which searches in parallel for totally digit-primes \( x \), where \( x \in (a,b) \), where \( a \) and \( b \) are integer numbers; \( a,b > 1 \) and \( a<b \). You can distribute the work by simply splitting the interval \( (a,b) \) into \( \text{size} \) subintervals, where \( \text{size} \) is the number of processes. Corresponding \( \text{DigitSumsList}(x) \) should get gathered to the process \( h \) which is defined as follows:

- For each totally digit-prime number \( x \) denote \( D(x) = \text{DigitSumsList}(x) \)
- Let \( d(x) = D(x)[-1] \) -- the last element in the list (the one-digit prime number)
- The target process \( h \) (process rank to gather all corresponding branches of certain tree) ito be chosen as follows:

<table>
<thead>
<tr>
<th>( d(x) )</th>
<th>( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

**HINT:**

- You may want to check out and use the following:

```python
from sympy import prime, isprime, primerange
```
**BONUS exercise:**

Can you develop a nice way for visualization of created trees?