CS326– Parallel and Distributed Computing

Assignment 4 [Total marks: 75]

The assignment is to be submitted at the google classroom as a single zip file (not .rar) containing code and output snippet[s] for each of the questions. The zip file should contain 4 folders named Q1, Q2, Q3, and Q4. Each folder shall contain data [code, input file, and output snippets] related to the questions (it will make it easy for the evaluator to execute your code). Moreover, an MS word document should also be attached containing code and output snippets of all the work. Note: Strictly follow the submission guidelines

Q#1: Implement a simple parallel linear search algorithm using MPI. The algorithm takes the total number of integers in a text file, the name of the text file, key-value to search, and the total number of processes at the command line. A process with rank#0, reads the file into a local array and scatters the data to other processes using a simple bock-distribution scheme. After the scatter, each process then searches for the key from their local portions. For every occurrence, the processes display massage with its own details: e.g., if process 0 found its occurrence at 6^{th} index, then it will print "process#0 found the key value at its local position:6".

Q#2: Implement the distributed algorithm for naïve pattern matching algorithm using MPI given at:

https://www.geeksforgeeks.org/naive-algorithm-for-pattern-searching/

Each process will read the whole sequence from a single file and calculate its portion (i.e., its start and end index to start searching) using its rank. You can use the block-distribution scheme to calculate the global start and global end index for each process. Here, you can safely assume that number of total characters in the sequence is always some multiple of the total number of processes.

- After calculating the start and endpoints, each process shall find and count occurrences of the sub-string (or pattern) from its start index to the end index.
- After the searching phase, every process shall send count and global locations of appearances to the P0. In the end, P0 shall generate a summary of the occurrences.

e.g., Assume that a text file seq.txt contains the following sequence of 16 characters: "ATGCGCATGCGCATGC". Further, assume that there are only two processes and pattern to be searched is "GC". Then following is the procedure and output:

CS326– Parallel and Distributed Computing

<table>
<tr><td>P0</td><td>P1</td></tr>
<tr><td>Seq:
<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>A</td><td>T</td><td>G</td><td>C</td><td>G</td><td>C</td><td>A</td><td>T</td><td>G</td><td>C</td><td>G</td><td>C</td><td>A</td><td>T</td><td>G</td><td>C</td></tr></table>
Start index=0
End index= 7

Start naïve pattern matching algorithm from the start index to end index. Keep track of total occurrences and the locations where the occurrences were found. Display own occurrences.</td><td>Seq:
<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>A</td><td>T</td><td>G</td><td>C</td><td>G</td><td>C</td><td>A</td><td>T</td><td>G</td><td>C</td><td>G</td><td>C</td><td>A</td><td>T</td><td>G</td><td>C</td></tr></table>
Start=8
End= 15

Start naïve pattern matching algorithm from start to end, keep track of total occurrences and the locations where the occurrence found. Now send this count and occurrence positions to process 0.</td></tr>
<tr><td>Receive counts and positions for the occurrences from each of the other processes, compute the total of received counts and print the information.</td><td></td></tr>
</table>

Program Output:

Sequence: ATGCGCATGCGCATGC

Pattern: GC

Total times the pattern found in the sequence is: 5

Positions for the occurrences are: 2, 4, 8, 10, 14

CS326– Parallel and Distributed Computing

Q#3: Histogram

A histogram is an accurate representation of the distribution of numerical data. To construct a histogram, the first step is to "bin" the range of values that is divide the entire range of values into a series of intervals and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent and are often (but are not required to be) of equal size.

The following example illustrates this concept,

We have an array of containing marks of students

Array of Marks	30	77	45	89	33	20	90	65	95	51	41	33	48	55	96

Then we have divided the marks into four bins:

0-33	4
34-50	3
51-75	3
76-100	5

Table 01

You have to give a parallel implementation of the histogram using MPI.

First processor 0 will create an array of n elements (n should be passed from command line arguments) and populate it with random data in range (0-100) then it will distribute it equally among p processors (if the number of elements is not divisible by the number of processors, then the maximum difference between each processor's chunk should be one). Use intervals defined above in Table 01 to calculate local results for the local bins.

Each processor will compute the number of elements in each bin from its local part of the array. Finally, an element-wise reduction step with processor 0 as a destination and sum as an associative operator will compute the globally correct results.

CS326– Parallel and Distributed Computing

Sample Output with n=12 and p =5

Total Elements = 12 Elements: 30,77,45,89,33,20,90,65,95,51,41,33 Processor 0's Part = 0-2 (3 elements) Processor 1's Part = 3-5 (3 elements) Processor 2's Part = 6-7 (2 elements) Processor 3's Part = 8-9 (2 elements) Processor 4's Part = 10-11 (2 elements)	After reducing the local bins at P0, the final distribution is: Bin 1(0-33) = 4 Bin 2(34-50) = 2 Bin 3(51-75) = 2 Bin 4(76-100) = 4

Happy Coding