Lab Course: distributed data analytics
Exercise Sheet 3

Nghia Duong-Trung, Mohsan Jameel
Information Systems and Machine Learning Lab
University of Hildesheim
Submission deadline: May 5th, Friday, 23:59PM (on LearnWeb, course code: 3117)

Topic: Distributed Computing with Message Passing Interface

Instructions

Please following these instructions for solving and submitting the exercise sheet.

1. You should submit a zip or a tar file containing two things a) python scripts and b) a pdf document.
2. In the pdf document you will explain your approach (i.e. how you solved a given problem), and present your results in form of graphs and tables.
3. The submission should be made before the deadline, only through learnweb.
4. If you are M.Sc. Data Analytics summer 2017 intake student, you should submit to "First term students" link on LearnWeb.
5. And if you are M.Sc. Data Analytics winter 2016 intake student, you should submit to "Second term students" link on LearnWeb.
6. If you are not M.Sc. Data Analytics student, you can submit to anyone of the links above.
7. All your solutions should be in python.

Exercise 1: Point to Point Communication (10 Points)

In this exercise you will write a parallel program using point-to-point communication routines. Suppose Process 0 has an integer array and it wants to send to all $P-1$ processes in COMM_WORLD, lets call this routine **sendAll**. A naive way to send this array is using a *for* loop at Process 0 and sequentially send it to all other processes i.e. it will take $P-1$ steps. Another possible ways is to use a *recursive doubling* algorithm, which will require $log(P)$ steps.

Short description of an efficient way: The *recursive doubling* algorithm is explained as follows:

Suppose you have P processes, where $P \geq 2^d$ i.e. if $P = 33$ than $d = 5$ and rank is the current process ID. Lets say your root process has rank 0. The root process sends to Process 1 and Process 2 only. All other processes first receive a message from recvProc, i.e. $recvProc = int((rank-1)/2)$ and sends to two more processes $destA = 2 * rank + 1$ and $destB = 2 * rank + 2$. But before sending, make sure $destA$ and $destB$ exist. To make sure every process has finished put a MPI.Barrier() just before returning from *sendAll* routine.

Your tasks are to write:

1. Implement the **sendAll** routine using the naive way i.e using a single for loop.
2. Implement the **sendAll** routine using the efficient way as explained above.
3. Compare performance of the two implementations by recording the time to finish each task. You can fix the number of process $P = 2^d$ i.e. 16 and 32, and send an array of size 10^3 or 10^5 or 10^7. [Hint: see which of the array sizes fit in your memory and even if you have less cores you should still use 16 and 32].

Exercise 2: Collective Communication (10 Points)

In this exercise you have to find an Image histogram. Images generally have RGB or gray scale values. Finding histogram is just calculating the frequency of occurrence of each gray scale or RGD value. You have to provide parallel implementation using collective routines only. Along with the code, please explain the following:

1. Pick an image with a high resolution i.e. resolution $\geq$ 2048x2048.
2. Data division strategy i.e. how you divide your data among processes.
3. How you assign tasks to different processes?
4. How you combine results from all the processes?
5. Did you implement for RGB or gray scale histogram? (2 points if it works with both)
6. Provide runtime analysis on varying number of processes i.e. $[1, 2, 3, 4]$. You have to show that your solution actually is reducing time to calculate histogram, if you add more processes. (3 points)

[Note:] If you dont know how to read an image in python, please see Annex below. (Install, use opencv and histogram tutorial)

Annex

1. Palach, Jan. Parallel Programming with Python. Packt Publishing Ltd, 2014.
2. For reading an image as gray scale or RGB you can use OpenCV.
   (a) Install opencv: conda install -c menpo opencv
   (b) A simple tutorial `http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html`
   (c) Histogram of an image in opencv `http://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_image_histogram_calcHist.php`
3. To time your program you have to use MPI.Wtime(): `http://nullege.com/codes/search/mpi4py.MPI.Wtime`
4. MPI tutorial (C/C++): `https://computing.llnl.gov/tutorials/mpi/`