Lab Course: distributed data analytics
Exercise Sheet 1

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

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. You will use python to solve problems mentioned below. If you like to solve it using Java you can also do it but only using Java Threads and not JOMP etc.

Topic: Multi-threading and Multi-processing

In this exercise sheet, you will solve problems using threading and multiprocessing API provided by Python. The lecture slides provides the basic introduction to the APIs. In the Annex section there are few useful resources that will help you further understand threading concepts and provide help in solving following exercises.

Exercise 0: Explain your system

Before we start with parallelizing our code, its always good to mention the system, which will be used for the experiments. It might happen that running time for your program differs on different hardware depending on different factors. Please create a table listing your hardware and software setup. Things you will mention will be processor, number of cores, RAM, OS, programming language version information, optimization flags (if used) etc (in future your network connection). This step is important as it gives a clear picture of the physical system used for the experiments.

Second important thing is to time your code. Python provides time package for this task. It is a good idea to just time a section of the code that is actually running in parallel.

Exercise 1: Basic Parallel Vector Operations with Threading/process (10 Points)

Suppose you are given a vector $\mathbf{v}$ having size n. Initialize your vector $\mathbf{v}$ with random numbers (can be either integers or floating points). You will experiment with three different sizes of vector, i.e. $n = \{10^7, 10^{12}, 10^{15}\}$. You have to parallelize vector operations mentioned below using threading or multiprocessing API. For each operation you will have to run experiment with varying number of threads, i.e. if your system has P threads than run experiments with threads $= \{1, 2, \ldots P\}$ for each size of vector

given above. You have to time your code for each operation and present it in a table. [Note: You have to define/explain your parallelization strategy i.e. how you assign task to each thread, how you divide your data etc.]

a) Add two vectors and store results in a third vector.

b) Find a minimum number in a vector.

c) Find an average of numbers in a vector.

Note: Please whenever you have to apply synchronization construct, explain what causes synchronization issue and what method you used to solve it. Another important thing is that your code should solve correct problem. To be sure you are doing everything correct, i.e. choose $n = 16$ to verify if your results are correct. Incorrect solution will not earn you any points.

Exercise 2: Parallel Matrix Operation using Threading/Multiprocessing (10 Points)

In this exercise you have to work with three matrices ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$) i.e each matrix having size $n \times n$. Initialize your matrices $\mathbf{A}$ and $\mathbf{B}$ with random numbers (can be either integers or floating points). Matrix $\mathbf{C}$ will store result of $\mathbf{A} \times \mathbf{B}$.

In case of matrix multiplication, you will experiment with three different sizes of matrices i.e. $n = \{10^2, 10^3, 10^4\}$. [note: your matrix will be $n \times n$, which means in case 1 you will have matrices with dimension 100x100]. You will have to run experiments with varying number of threads, i.e. if your system has P threads than run experiments with threads $= \{1, 2, \ldots P\}$ for each matrix size given above. You have to time your code and present it in a table.

a) Implement parallel matrix multiplication. You can look at the implementation provided in the lecture (slide 48) `https://www.ismll.uni-hildesheim.de/lehre/bd-17s/script/bd-01-parallel-computing.pdf`. Perform experiments with varying matrix sizes and varying number of threads.

b) In this exercise you will improve matrix multiplication by using a better approach then mentioned on slide 48. Either you can change your algorithm i.e. dijkstra's algorithm for matrix multiplication or use Tiled approach mentioned on slide 50. Repeat your experiment with varying matrices sizes and threads.

Note: If you are using python threads did you see any lift in performance, if not please explain why. Also try to use multiprocessing api provided by python. Also note that depending on your system RAM you might not be able to run experiment with matrix size $n = 10^4$.

Annex

1. Parallel Computing tutorial `https://computing.llnl.gov/tutorials/parallel_comp/`.
2. Palach, Jan. Parallel Programming with Python. Packt Publishing Ltd, 2014.
3. Matrix Multiply: A Case Study `https://ocw.mit.edu/courses/electrical-engineering-and-computer-scien 6-172-performance-engineering-of-software-systems-fall-2010/video-lectures/lecture-1-matrix-mu MIT6_172F10_lec01.pdf`
4. Python Threading tutorial `https://pymotw.com/2/threading/`
5. Python multiprocessing tutorial `https://pymotw.com/2/multiprocessing/basics.html`