CMSC 603 – High Performance Distributed Systems

Assignment 1: KNN on MPI — Virginia Commonwealth University, Fall 2020

Due date: September 13, 2020

Big data mining involves the use of datasets with millions of instances. The computational complexity of machine learning methods limits their scalability to large-scale datasets. The simplest classifier is the k-nearest-neighbor classifier (KNN) but its computational complexity is O(n^2), where n is the number of instances, for classifying the n instances in a dataset with respect to the n-1 distances to other instances.

The KNN algorithm computes for each data instance the distances to the other instances and predicts the data class by majority voting from the k-nearest neighbors. The accuracy of the classifier is measured as the relation between the number of successful predictions and the number of instances. You may run the code using the datasets provided and analyze how the runtime increases with the size of the dataset.

The assignment consists on implementing the KNN algorithm, first a single-threaded version, and second a distributed version using MPI. Conduct all the code optimizations you consider relevant to speed up its execution, as long as the output is correct (accuracy of sequential and parallel must be the same). A skeleton is provided for the single-thread version to facilitate the reading of the dataset file and accuracy calculation. The number of k should be defined by an input parameter in the function and provided by the user. Should you consider critical sections in your code, be aware of data races and employ the tools to guarantee the mutual exclusion.

Execute both the single-thread and MPI codes on maple.cs.vcu.edu to:

1. Compare the runtime of the single-thread and distributed versions considering the datasets provided having different sizes. Calculate the speedup using 1, 2, 4, 8, and 16 processes.
2. Estimate the proportion of parallel code by relating the speedup you get and the number of threads employed. What's the maximum speedup you would be able to obtain using an infinite number of threads and cores?
3. Force the code to run using 1024 processes. Recalculate the speedup. What's your conclusion?
4. Deliverables:
    a. Source code of the single-thread and MPI implementations. Use a private github repository to upload and commit your code as you progress. The name of the repository **must be** "2020-603-A1-LASTNAME", replace LASTNAME with your last name. Add my github username (canoalberto) to the repository. I'll monitor your commits and timing. Start pushing your code since the moment you start working on the assignment. Do not wait to push the code only in the last minute. You don't need to upload the code to Blackboard.
    b. Documentation (2 pages max). Use the 2-column LaTeX template from IEEE (download). Include: Abstract, Introduction, Methodology (description of the implementation), Results (tables and figures with runtimes, speedups) and conclusions. Submit the PDF to Blackboard.

Extra credit: Run on AWS/Azure/Google cloud with 2+ nodes for extra credit (requires setup)

Remember: no code from previous years / internet may be employed here, your code must be your original contribution. I DO check your code. Plagiarism will be reported.