PA 3(Draft)
CS 615
By Samson Haile
March 29, 2017

Assignment Description

The objective of this assignment is to determine differences in computation time when executing the bucket sort algorithm. The computation involved organizes the data into one of one-hundred buckets representing a specific range of data. The execution time is evaluated on different sizes of data, utilizing subsets of a single set of an unorganized data set to represent the different sizes of data that are timed. In the case of a parallelized algorithm, the computation time is expected to decrease with an increase in the number of cores utilized. With respect to parallel computation, subtasks of the computation are divided by partitioning the data set into n sets, where n represents the number of cores used. The sequential algorithm will be plotted in a two dimensional array size versus time graph. The parallel algorithms will be plotted in a three dimensional array size versus cores used versus time graph. An assessment will then be made to explain trends and differences between the graphs.

Assignment Methodology

The assignment was implemented in the form of two c files and two batch files, as well as a makefile. Each c file and batch file match to designate one of the two implementation types of the bucket sort. This includes sequential and parallel computation of the bucket sort algorithm. The sequential code simply executes the computation code on a single core, iterating across the different array sizes and sorting subsets of a large data, increasing the subset size as the execution progresses. The parallel programs use the general heuristic the sequential does, but involves extra components for properly sending and receiving work between the master and slave nodes. The number of cores the parallel programs run on is modified through the -n and -N parameters of the batch scripts used to run the programs. This is to ensure that the programs request only as much resources as it uses.

The sequential code operates by first reading the code into a vector data structure. This operation has no impact on the execution time since it is done before any sorting is performed. An array is then initialized to handle receiving the sorted array for each subset iteration. 100 individual arrays are also specified at the start of each sort iteration to represent the buckets for the bucket sort. The bucket sort algorithm then operates, beginning by determining the largest value in the data set. This largest value will then implicitly define the numerical range for which each bucket represents. Each data value is then placed into the bucket matching the data value's respective numerical range. After this is complete, each bucket is sorted sequentially using an insertion sort algorithm. Upon sorting all of the buckets, they are inserted into the result array in increasing order of the arranged buckets, resulting in the sorted array.

The c code is compiled through the usage of makefile which is capable of producing an executable to run the code, as well as the ability to remove the produced executable from the directory. Once the executable is produced, one of three batch files can be run in the form of the command sbatch <file_name>. The batch file seqSort.sh measures the time it takes to bucket sort different sized data sets in a sequential manner. The batch file parSort.sh measures the time it takes to bucket sort different sized data sets in a parallel manner.

Data and Analysis

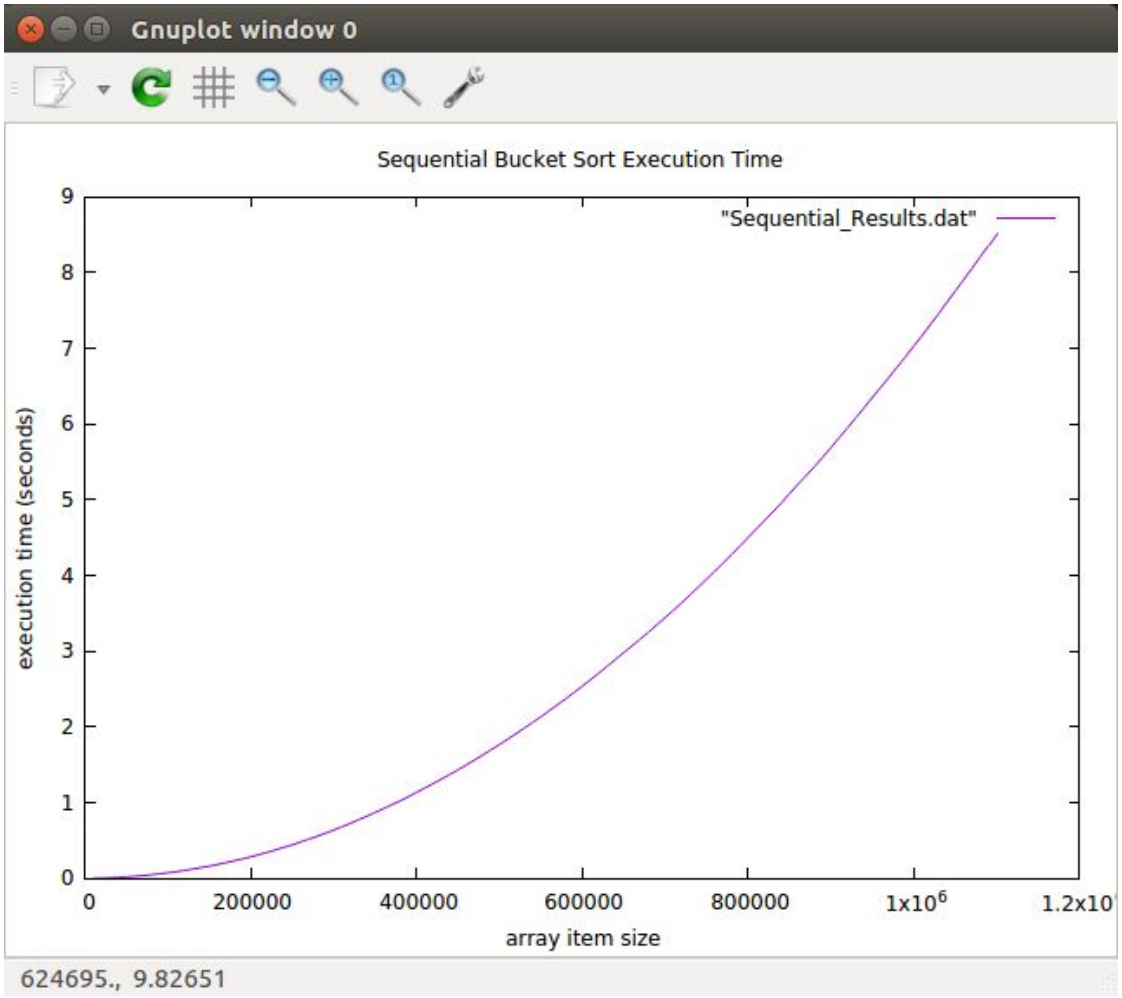

Figure 1: The graph shows the execution time of a sequential bucket sort algorithm over varying sizes of arrays. The graph pattern shows a quadratic increase in execution time, which can be attributed to the underlying insertion sort used in the bucket sort.