Project Assignment 03 Report

CS415

3-29-2017

By: Jervyn Suguitan

Description of Assignment

Project 03's main purpose was to teach students about bucket sort and how to parallelize the code that creates the bucket sort. This report will only talk about running bucket sort sequentially and how to do it and discussion based on several variables. The timing function that was used to time calculation for this project was MPI_Wtime.

Part 1: Sequential

The first part of the project tasks students in figuring out how to code bucket sort sequentially. The first thing to do however is to create an input file with a specified amount of numbers, this is done using the createValues.cpp. After creating the input file with the set amount of numbers that will be sorted we sort the numbers in the sequential.cpp code. The code reads in the input file and places those numbers into a holder/buffer. This buffer puts all the numbers into buckets. The number of buckets is predetermined to 10 and each bucket has a size of the max amount of possible integers. I do this for the case where all the numbers that need to be sorted are all on the same bucket. When they are initially put into the buckets they are not sorted. Once the numbers are in the buckets I get the numbers from each bucket and sort them using a sort function in the algorithm library. After the buckets are sorted, I put them back into a large array that contains all the values in order. The sorting function I use from the algorithm library runs at a nlog(n) speed (this is documented here http://www.cplusplus.com/reference/algorithm/sort/).

Figure 1 and Figure 2 showcase the run time of Bucket Sort with multiple amount of integers. The only difference between Fig. 1 and Fig. 2 is that Fig. 1 includes the trials with 50000000 integers sorted while Fig. 2 does not. These numbers of integers sorted (1000, 100000, 500000000) were chosen to show times over a small, medium, and large amount of sorted integers. These graphs show that as more numbers that are needed to be sorted, the longer it takes to sort. To specify all of the trials in Fig. 1 and Fig.2 all use a number range of 0 to 10000. Thus the numbers that are sorted will only be between that range.

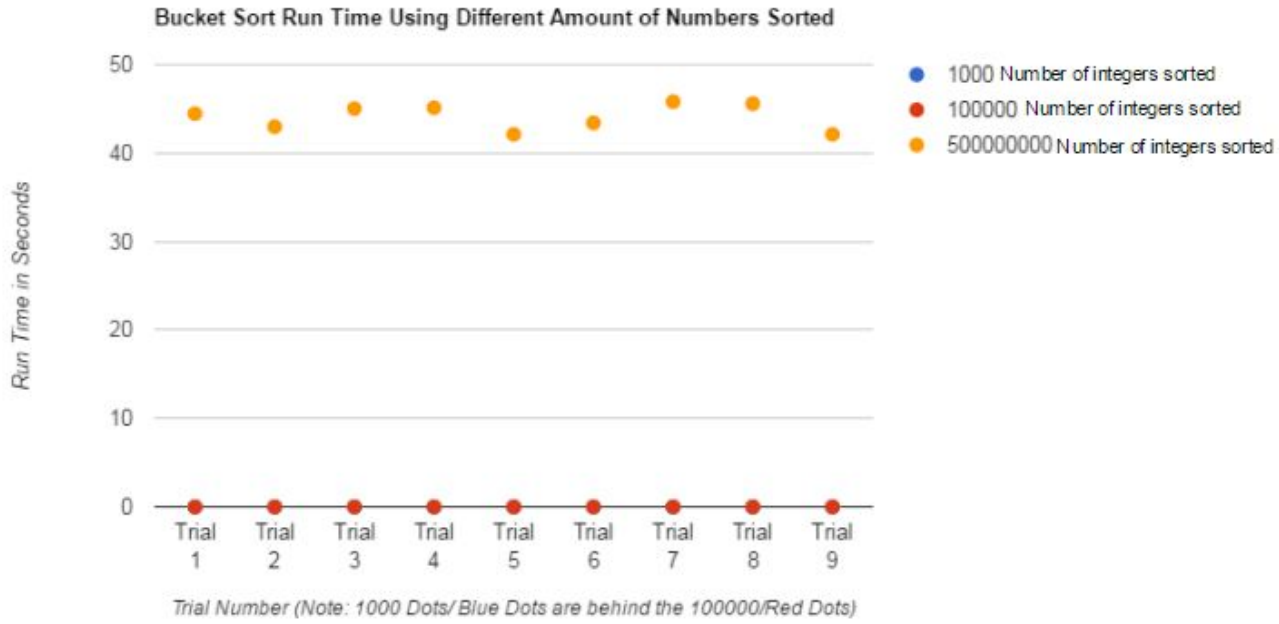

Fig. 1: Graphs that showcase the sequential times for different amount of integers sorted for Bucket sort. The Y axis shows the time in seconds and the x-axis shows the trial number for that specific resolution. This graph includes times for 500 billion integers sorted. The times do not vary large amounts since everything is sequential thus there can not be a large speed up. Speed up or slow down may occur due to the processor speed which could be different for every trial. The 1000 trials are behind the 100000 trials and are not seen in the picture

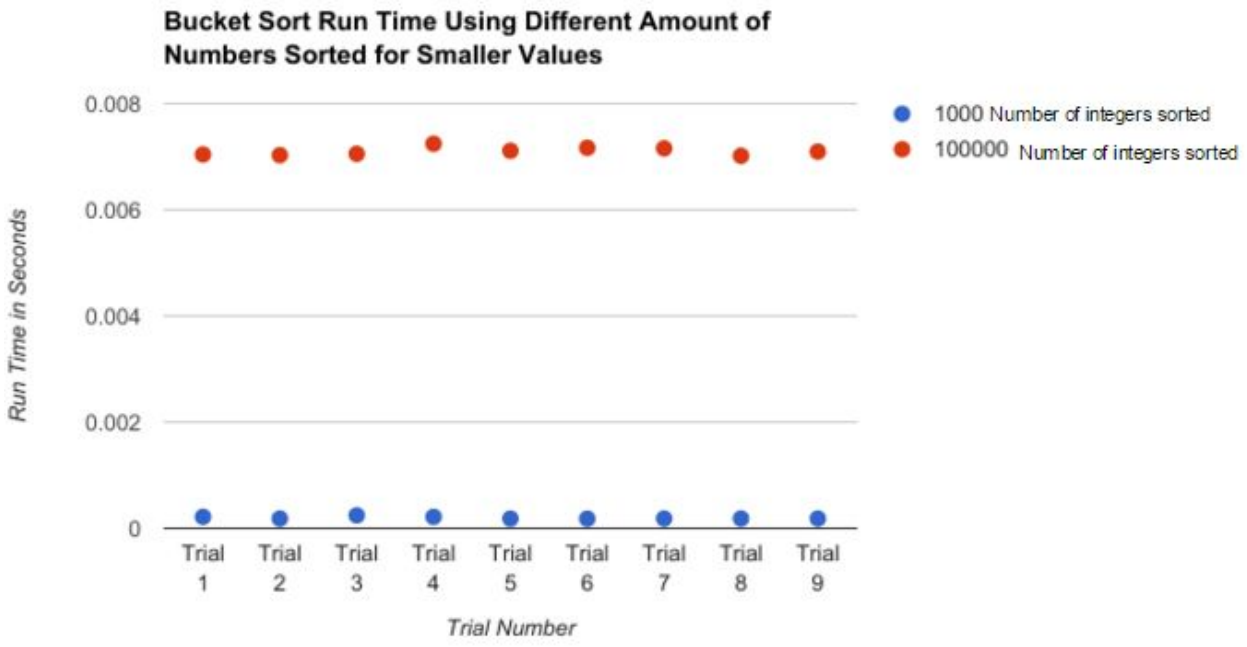

Fig. 2: Graphs that showcase the sequential times for different amount of integers sorted for Bucket sort. The Y axis shows the time in seconds and the x-axis shows the trial number for that specific resolution. This graph shows similar data to Fig. 1 but only showcases the trials that sort 1000 integers and 100 thousand integers.. The times do not vary large amounts since everything is sequential thus there can not be a large speed up. Speed up or slow down may occur due to the processor speed which could be different for every trial.

The times for each of the sorting do not vary large amounts, one reason this occurs is that there is no message passing between master and slave which takes up time. There is no message passing since it is sequential there is no need to use multiple processors thus the program uses a " -n1 " to show that we will only use one core. Also since everything is calculated a pixel at a time the times will not vary too much since there can not be any speed up and the only variation is the speed of the processor that works on this data.

Fig. 3 shows the average for sequential time shown as a chart. In the fifth column in the chart, we see the average time of 500 million integers however the range changed with the numbers being sorted could be between 0 - 1000000. This increased the average by around 7 whole seconds. The slowdown is most likely occurred due to the sort function in the algorithm library as the speed of the entire bucket sort is reliant on the speed of that sort.

Averages for Sequential Time

Number of Integers	1000	100000	5000000	5000000(million)
Average Times	0.0001970620444	0.007101324444	44.06544444	51.7652

Fig. 3: Chart that showcases the average times of different amount of sorted integers. The 5th column in the chart, we see the average time of 500 million integers however the range changed with the numbers being sorted could be between 0 - 1000000.

Problems Encountered in Sequential

Problems that I have encountered during this part of the project is that I didn't know that I still had to use MPI_Init even though there was no sending to slave functions at all. I needed it for the MPI_Wtime functions. I had to think of a way to use bucket sort without using a queue data structure or a vector. I also had to figure out a way to put the numbers in the buckets correctly.