Adam Montano

CS 415 – Spring 2017

PA3 – Bucketsort: Sequential Version

Introduction

Bucketsort is a sorting algorithm that separates data into “buckets”, sorts each bucket individually and then merges the buckets back together. Bucketsort is best when the input data is uniformly distributed. The run time relies on the amount of input data and the amount of buckets being used. In a parallel version, each processor can act as a bucket. Each bucket is sorted using a different sorting algorithm, usually insertion sort.

Sequential Implementation

The sequential implementation used a two dimensional array with the second dimension acting as each bucket. The data was “scattered” to each bucket. Then, each bucket was sorted individually in place. Once each bucket was sorted, the data was “gathered” back into a single array. Finally, the data had been sorted. Below is a table of various sizes of inputs, buckets and the resulting run times.

	2 Buckets	**4 Buckets**	**8 Buckets**	**16 Buckets**
Small (1000)	0.084359	0.023207	0.011391	0.003623
Medium (100,000)	0.161849	0.056677	0.023527	0.012971
Large (10,000,000)	7.54979	3.77899	1.93737	0.998498

Here is this table graphed to a surface plot using Excel.

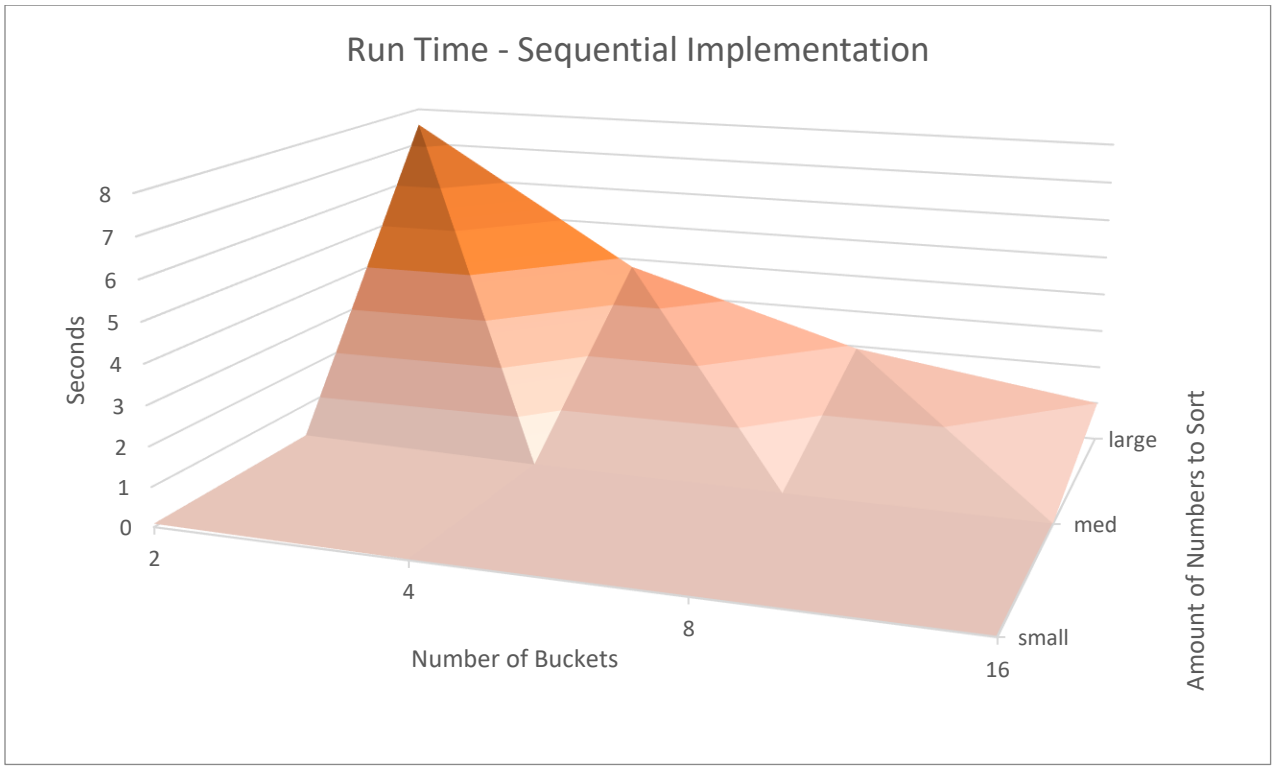

Figure 1 - Run Time Sequential

It is evident by the surface plot (Fig. 1) that an increased number of buckets allows for a general lower run time. The lower the number of buckets, the algorithm reverts closer to the sorting algorithm implemented for each bucket. A small amount of input gives near constant time regardless of the number of buckets.

Issues

The implementation of this program could have been much easier had it been developed using vectors instead of arrays due to the nature of the method of scattering. Also, when the input reached larger than ten million inputs, the program could not continue.

Conclusion

The sequential implementation was straightforward despite a few bugs. Clearly, the more buckets the more optimal the sorting becomes. My prediction for the parallelized version is that it will only become more optimal as each processor will most likely be able to sort the numbers much faster. The overhead will most likely come from communication.