Bitonic Sort (Parallel Execution)

Bitonic mergesort is a parallel algorithm for sorting. It is also used as a construction method for building a sorting network.

Pseudocode:
The bitonic merge sort algorithm that was used revolves around three main functions bitonicSort, merge, and compare.

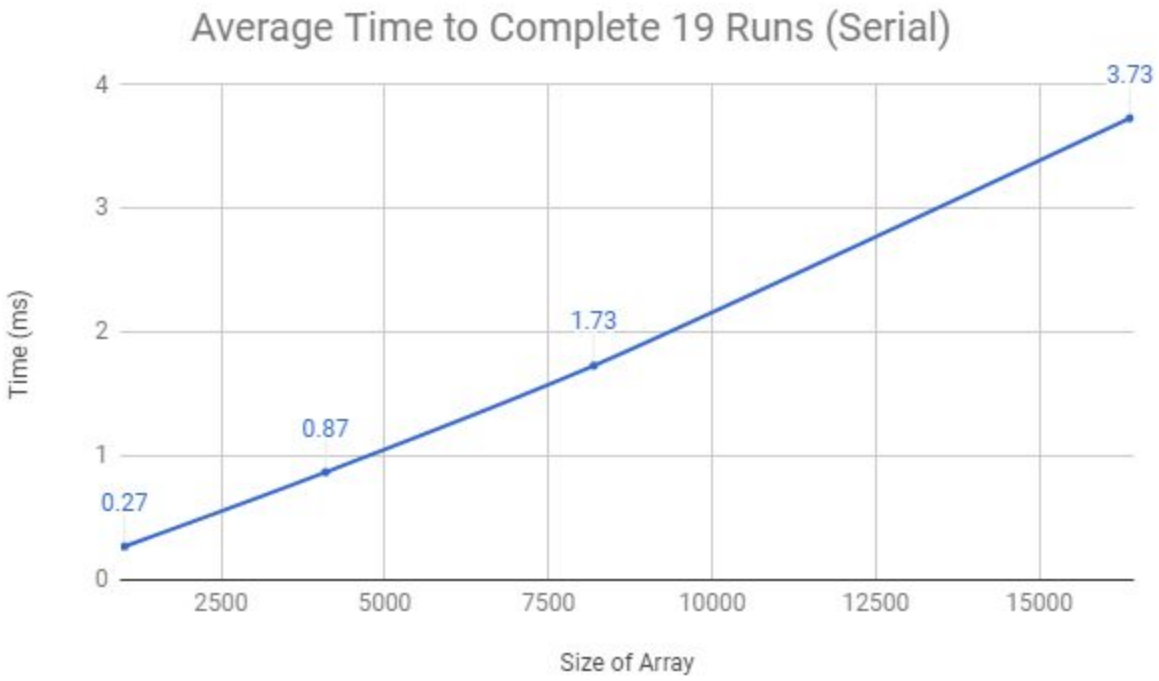
```
bitonicSort(list, lowIndex, count, direction){
	If (count > 1){
		K = count / 2
		bitonicSort(list, lowIndex, k, 1) // Ascending
		bitonicSort(list, lowIndex + k, k, 0) // Descending
		merge(list, lowIndex, count, direction)
	}
}

merge(list, lowIndex, count, direction){
	If (count > 1){
		K = count / 2
		For (i = lowIndex; i < lowIndex + k; i++){
			compare(list, i, i + k, direction)
		}
		merge(list, lowIndex, k, direction)
		merge(list, low + k, k, direction)
	}
}

compare(list, i, j, direction){
	If (list[i] > list[j] && direction == 1) || (list[i] < list[j] && direction == 0)){
		// Swap elements
		temp = list[i];
		list[i] = list[j];
		list[j] = temp;
	}
}
```

For executions where the number of processing nodes is greater than 1, the array will be split into N equal sections, where N is the number of processing nodes to use. Each subsection of the original array will be sorted with the bitonic merge sort. When each node has completed the sorting of its sub array, he N subarrays will be merged together for a complete sorted list.

Data:

Serial:

Array size	AVG of 19 runs in ms (Serial)
1024	0.27
4096	0.87
8192	1.73
16384	3.73



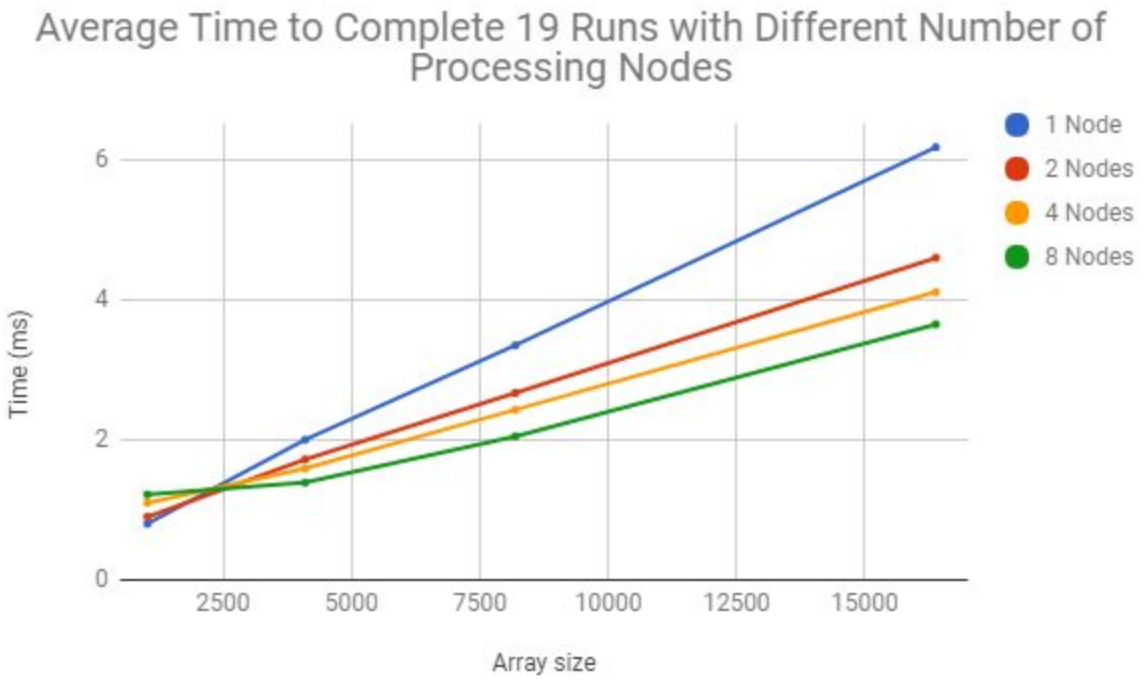
The serial implementation of the bitonic sorting algorithm seems to have a linear relationship between the size of the array and the time to sort the array.

Parallel:[1]

Array size	1 Node	2 Nodes	4 Nodes	8 Nodes
1024	0.8	0.9	1.1	1.22
4096	2	1.72	1.59	1.39
8192	3.35	2.67	2.43	2.05
16384	6.18	4.6	4.11	3.65

All data points are the average of 19 runs.

[1] All executions were ran on a i7-6700k @ 4.0 GHz



Using the parallel implementation of the algorithm small array sizes (N < ~2400) do not benefit from the increased number of computing nodes. The point around where all the lines cross, around an array size of 2400, is where there is a benefit for more nodes. This is due to the overhead that is required for spawning new threads.