Array Packing Implementation – A Parallel approach

By,
Abhishek Cumbakonam Desikan
Rajesh Balasubramanian
Ramalingam Sankaran
Aswin Gokulachandran

The Problem

Given an array X, having some labelled items. We want to rearrange the items so that all labelled items are listed before the unlabeled items.

It can be visualized as sorting of 0’s and 1’s where 1’s represent the labelled items, and 0’s represent the unlabeled items

Before:	1001101001011001101
After:	1111111111000000000

Serial Implementation

Can be achieved with the help of counting sort algorithm:

- Count the number of occurrences of 0’s by traversing through the entire data.
- Count the number of occurrences of 1’s by traversing through the entire data.
- Arrange the data in the preferred order.

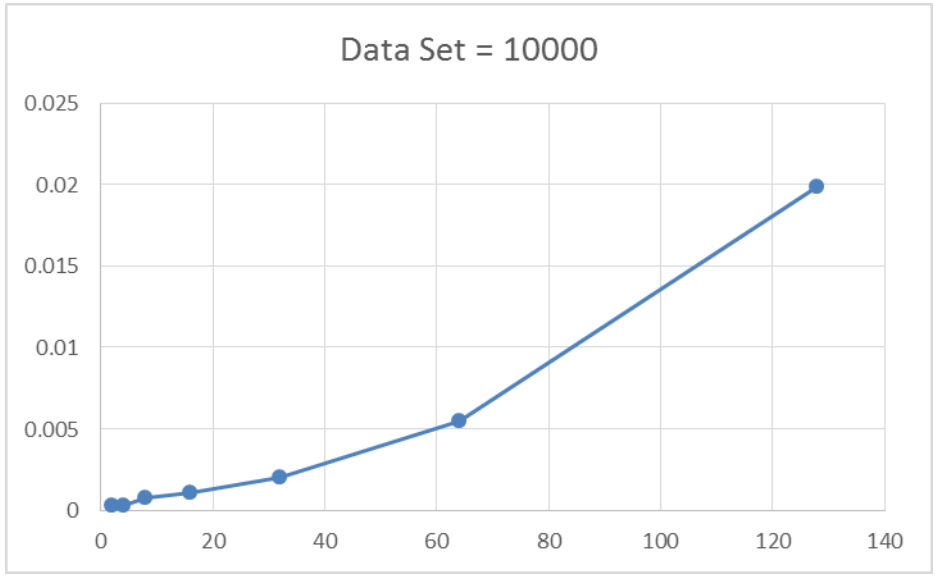
- Running time – O(n)

Parallel Implementation

Considering an input dataset of 0's and 1's:

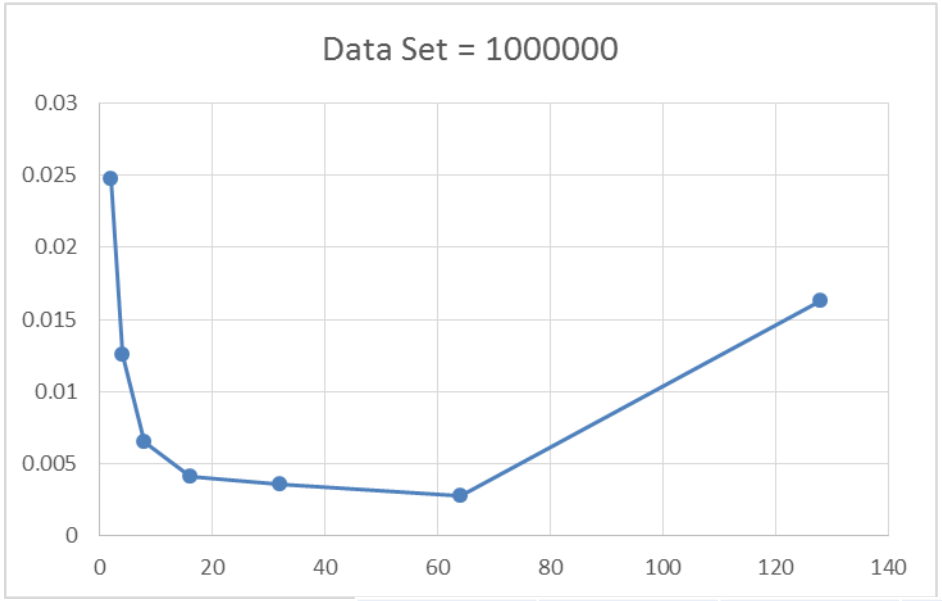
- We distribute the data evenly across different processors (depending on size of input data)
- Each processor individually performs a parallel prefix sum operation on 0's and then on 1's, both on its local set of data.
- Upon completion, the processors perform a prefix sum among all other processors on 0's and 1's.
- The final processor broadcasts the value of number of 0's and 1's among all processors. The data is rearranged accordingly.

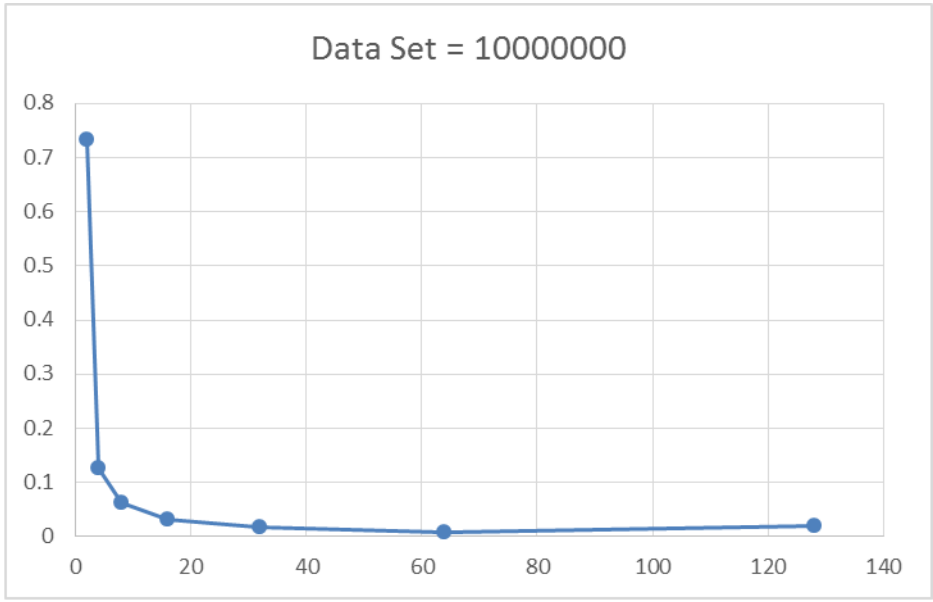
Results - array packing 1's and 0's (no of processors vs running time)



			time		
processors	data=100	data=10000	data=100000	data=10000000	data=30000000
2	0.000054	0.000299	0.024709	0.732996	0.735197
4	0.000127	0.000255	0.012578	0.125272	0.373667
8	0.000743	0.000788	0.00649	0.062512	0.184988
16	0.001071	0.00108	0.00412	0.031699	0.093299
32	0.00198	0.002016	0.003578	0.017145	0.047755
64	0.002136	0.00551	0.002809	0.008926	0.022727
128	0.01641	0.019855	0.016321	0.019312	0.027064

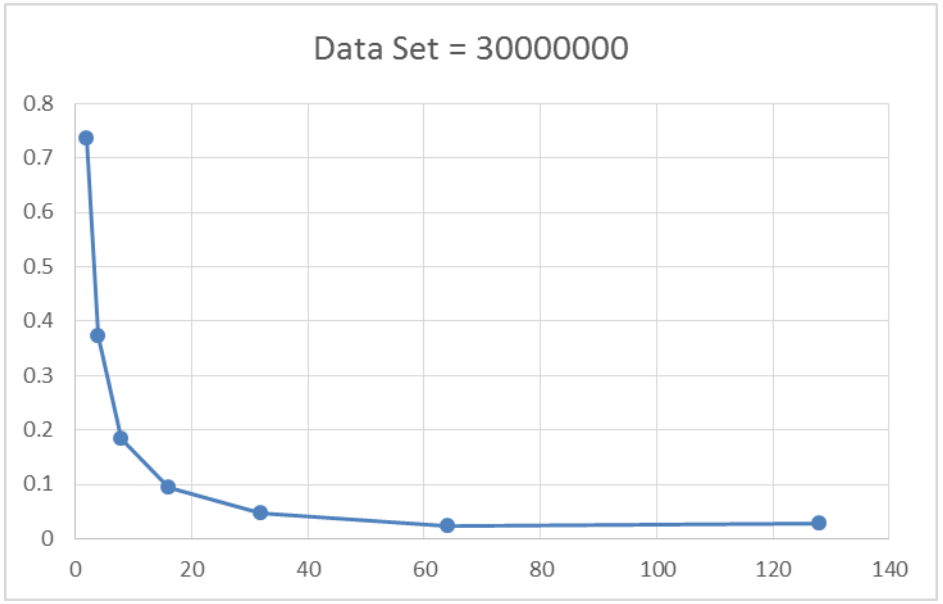
Results for array packing 1's and 0's (no of processors vs running time)





			time		
processors	data=100	data=10000	data=100000	data=10000000	data=30000000
2	0.000054	0.000299	0.024709	0.732996	0.735197
4	0.000127	0.000255	0.012578	0.125272	0.373667
8	0.000743	0.000788	0.00649	0.062512	0.184988
16	0.001071	0.00108	0.00412	0.031699	0.093299
32	0.00198	0.002016	0.003578	0.017145	0.047755
64	0.002136	0.00551	0.002809	0.008926	0.022727
128	0.01641	0.019855	0.016321	0.019312	0.027064

Results for array packing 1's and 0's (no of processors vs running time)



			time		
processors	data=100	data=10000	data=100000	data=10000000	data=30000000
2	0.000054	0.000299	0.024709	0.732996	0.735197
4	0.000127	0.000255	0.012578	0.125272	0.373667
8	0.000743	0.000788	0.00649	0.062512	0.184988
16	0.001071	0.00108	0.00412	0.031699	0.093299
32	0.00198	0.002016	0.003578	0.017145	0.047755
64	0.002136	0.00551	0.002809	0.008926	0.022727
128	0.01641	0.019855	0.016321	0.019312	0.027064

An extension to the problem- N labels

If our data set has more than just one label, the problem of array packing is changed into a more routine counting sort problem. Considering an array X to have items with labels from 0 to N, where 0 represents the unlabeled items and 1 to N represent the different labels. Here, we need to arrange the data items such that all labelled items are arranged in increasing order, and then followed by the unlabeled items.

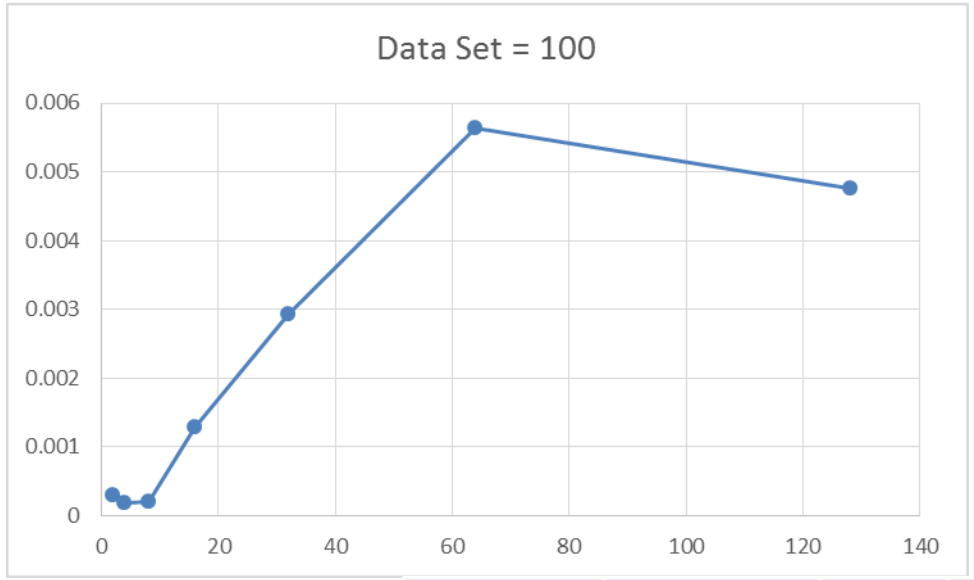
Before:	10223884444996671 75
After:	11223444456677889 90

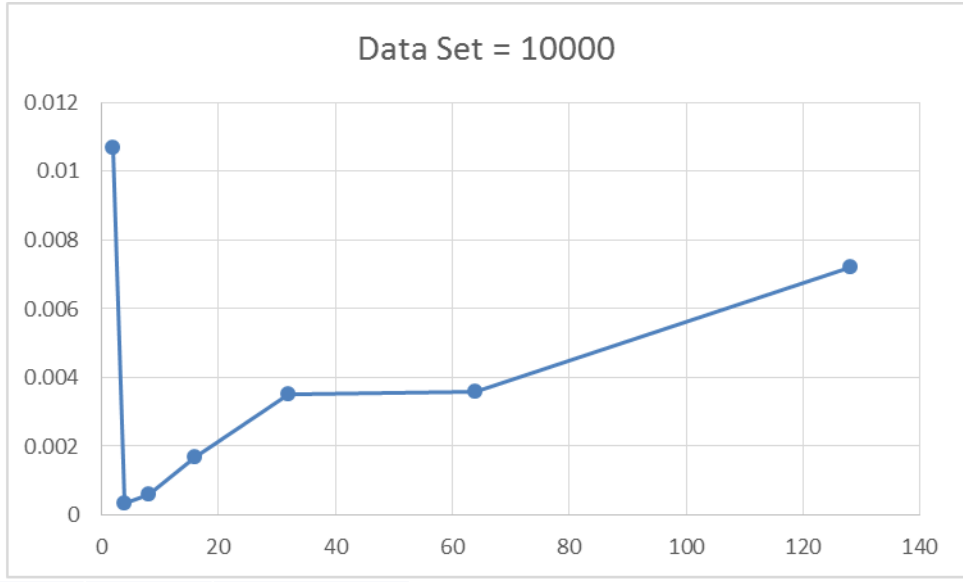
Parallel implementation for N labels

Considering an input dataset containing N labelled items:

- We distribute the data evenly across different processors(depending on size of input data).
- We initialize an array of size equal to the size of the input dataset with initial value 0 for all elements.
- We pass the array among each processor and increment the count of each label in its corresponding array element. Once all the local data in a processor has been traversed, the array is passed on to the next processor.
- This prefix sum operation on the array is continued till all data in all processors are traversed through.
- The last processor will hold an array with the count of all labels. The data is rearranged accordingly.

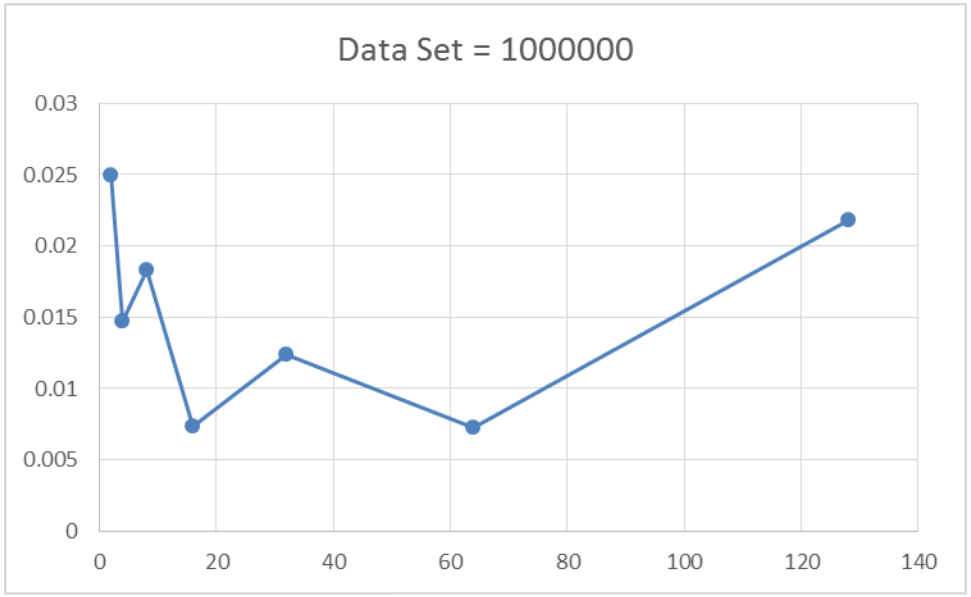
Array packing n labels
(no of processors vs running time)

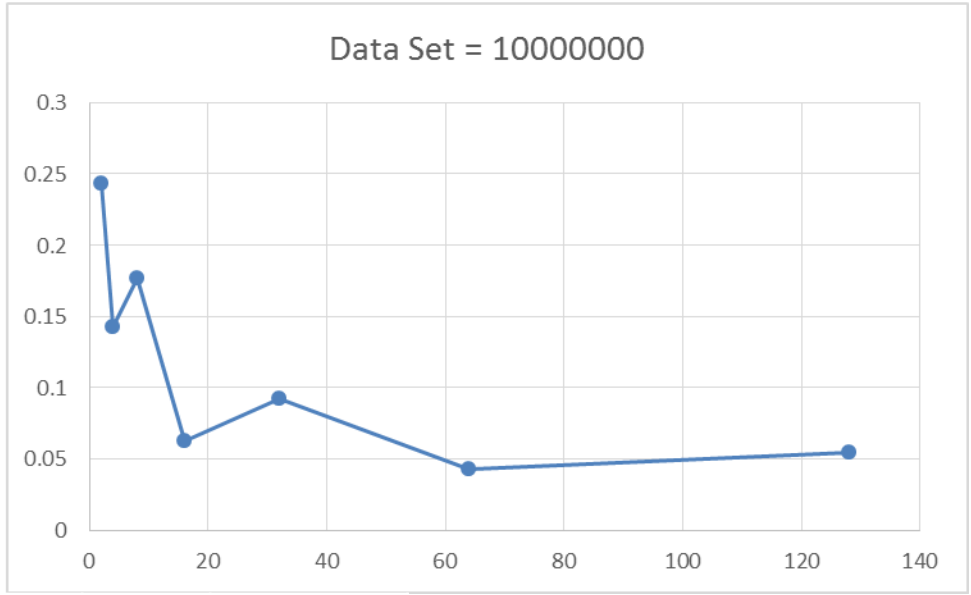




			time		
processors	data=100	data=10000	data=100000	data=10000000	data=30000000
2	0.000294	0.010678	0.024952	0.243225	0.740902
4	0.000192	0.000347	0.014706	0.142665	0.427468
8	0.0002	0.000595	0.018268	0.176992	0.51861
16	0.001297	0.001691	0.007344	0.062234	0.181338
32	0.002933	0.003527	0.012398	0.09193	0.262581
64	0.005641	0.003592	0.007206	0.042311	0.119937
128	0.004759	0.00721	0.021832	0.054439	0.113642

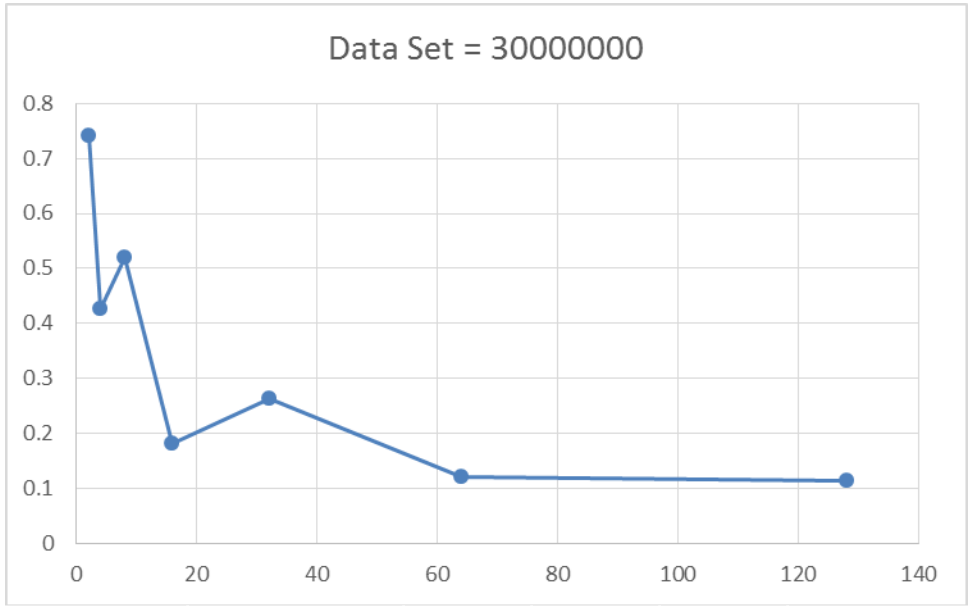
Array packing n labels
(no of processors vs running time)





			time		
processors	data=100	data=10000	data=100000	data=10000000	data=30000000
2	0.000294	0.010678	0.024952	0.243225	0.740902
4	0.000192	0.000347	0.014706	0.142665	0.427468
8	0.0002	0.000595	0.018268	0.176992	0.51861
16	0.001297	0.001691	0.007344	0.062234	0.181338
32	0.002933	0.003527	0.012398	0.09193	0.262581
64	0.005641	0.003592	0.007206	0.042311	0.119937
128	0.004759	0.00721	0.021832	0.054439	0.113642

Array packing n labels
(no of processors vs running time)



			time		
processors	data=100	data=10000	data=100000	data=10000000	data=30000000
2	0.000294	0.010678	0.024952	0.243225	0.740902
4	0.000192	0.000347	0.014706	0.142665	0.427468
8	0.0002	0.000595	0.018268	0.176992	0.51861
16	0.001297	0.001691	0.007344	0.062234	0.181338
32	0.002933	0.003527	0.012398	0.09193	0.262581
64	0.005641	0.003592	0.007206	0.042311	0.119937
128	0.004759	0.00721	0.021832	0.054439	0.113642

Using Quicksort – Parallel Implementation

- We randomly choose a pivot from one of the processes and broadcast it to every processor
- Divide the local list of each processor based on the pivot
- Each processor in the lower half of the processor list sends its "upper list" to processor with rank=own rank+s/2 and keep the lower half of the data
- Now, the upper-half processes have only values greater than the pivot, and the lower-half processes have only values smaller than the pivot
- Processor divides into 2 groups and the algorithm runs recursively

Using Quicksort – Parallel Implementation

- After log P recursions, every process has an unsorted list of values

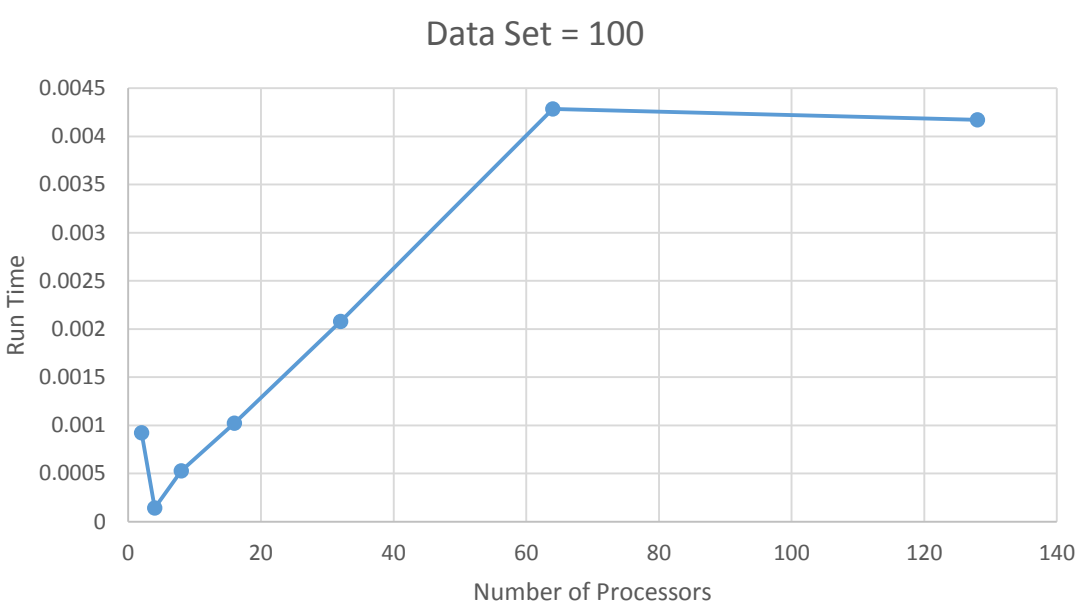
completely disjoint from the values held by the other processes

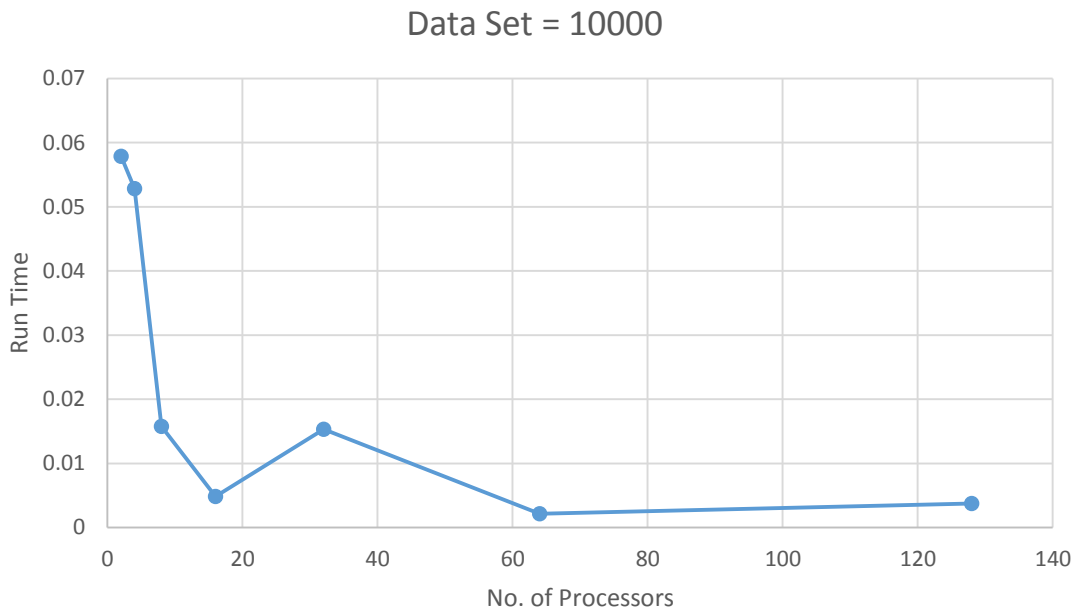
- The elements in each process will be sorted and that the highest element

in the Pi will be lesser than the lowest element in Pi+1

- Each process can sort its list using sequential quicksort
- Implementation is still in progress. Have executed the code using MPI but yet to execute it on SLURM

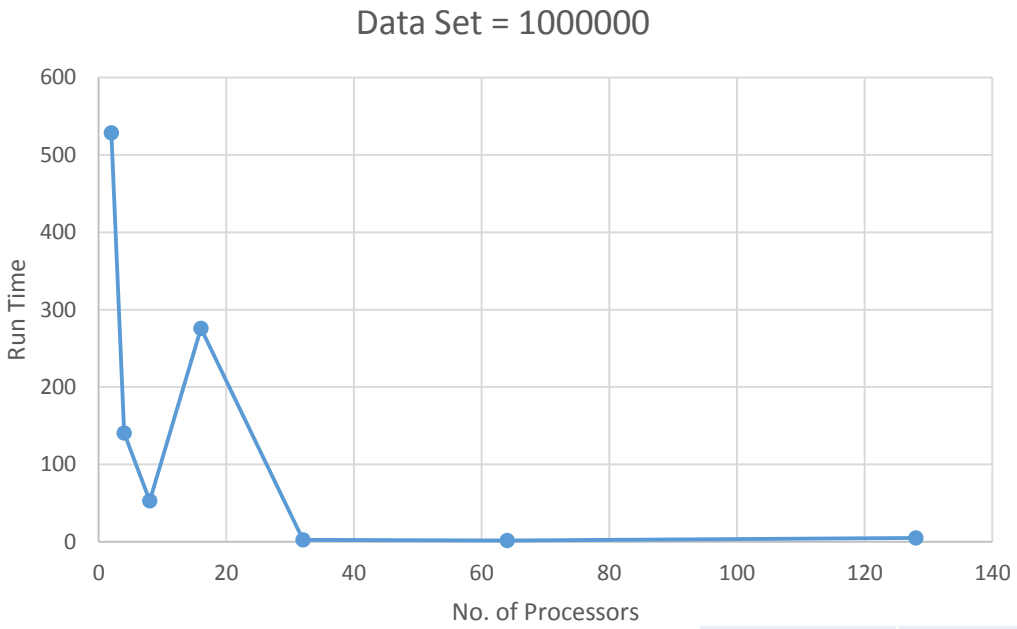
Array Packing N-labels – Quick Sort

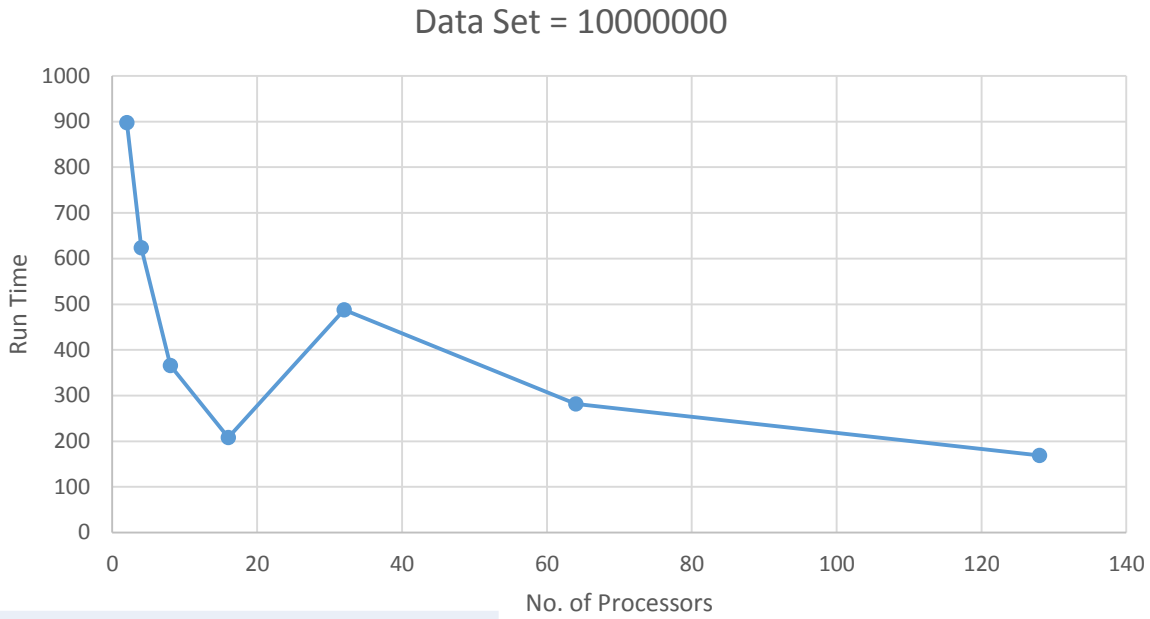




	Data Set				
Processors	100	10000	1000000	10000000	30000000
2	0.000924	0.057902	528.4385	897.6372	904.2461
4	0.000144	0.052861	140.5939	623.5641	682.4145
8	0.000529	0.015749	52.78392	365.9124	501.0876
16	0.001022	0.004815	275.6174	208.4245	452.4312
32	0.002078	0.015336	2.281963	487.6913	557.5093
64	0.004285	0.002132	1.463041	281.4132	433.1035
128	0.004172	0.003729	4.90777	169.0892	185.2132

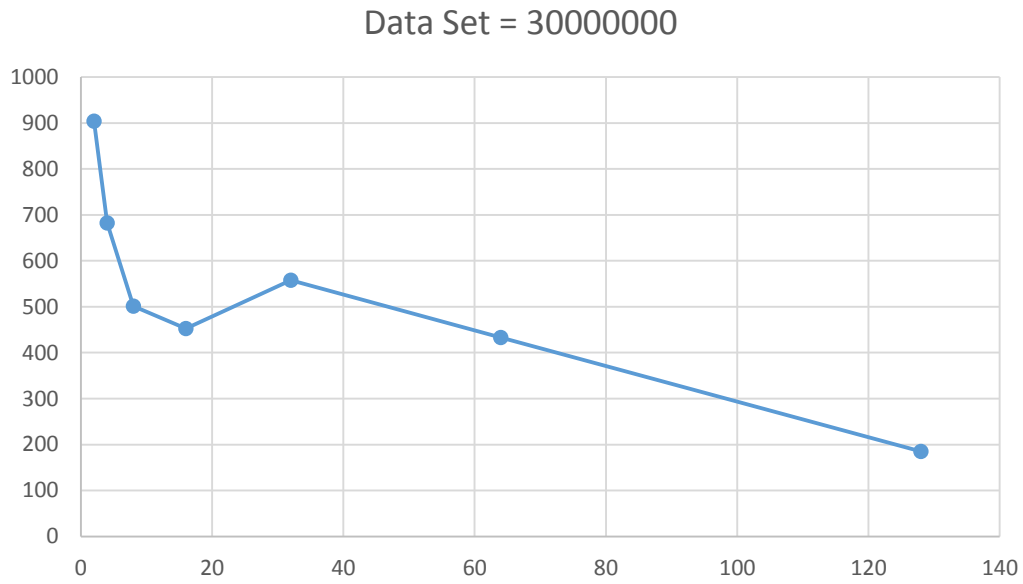
Array Packing N-labels – Quick Sort





	Data Set				
Processors	100	10000	1000000	10000000	30000000
2	0.000924	0.057902	528.4385	897.6372	904.2461
4	0.000144	0.052861	140.5939	623.5641	682.4145
8	0.000529	0.015749	52.78392	365.9124	501.0876
16	0.001022	0.004815	275.6174	208.4245	452.4312
32	0.002078	0.015336	2.281963	487.6913	557.5093
64	0.004285	0.002132	1.463041	281.4132	433.1035
128	0.004172	0.003729	4.90777	169.0892	185.2132

Array Packing N-labels – Quick Sort



	Data Set				
Processors	100	10000	1000000	10000000	30000000
2	0.000924	0.057902	528.4385	897.6372	904.2461
4	0.000144	0.052861	140.5939	623.5641	682.4145
8	0.000529	0.015749	52.78392	365.9124	501.0876
16	0.001022	0.004815	275.6174	208.4245	452.4312
32	0.002078	0.015336	2.281963	487.6913	557.5093
64	0.004285	0.002132	1.463041	281.4132	433.1035
128	0.004172	0.003729	4.90777	169.0892	185.2132

References

- Algorithms – Sequential and Parallel. A unified approach - By Russ Miller and Laurence Boxer

Thank you