IntroPDP: Assignment 1

Shriram Ravi
shriramr
shriramr@buffalo.edu
50419944

Q1 – A brief:

Given a range of keys, the best sequential sorting algorithm is Counting Sort, which can be done with time complexity of O(n+k) and space complexity of O(k) where n is the number of keys to be sorted and k is the range of keys.
Given, the keys are from –(p-1) to +(p-1), which makes k = 2p-1
Hence, our sequential solution will have the time complexity O(n+2p-1) and space complexity of O(2p-1)

Q2 – Performance Discussion:

Assumptions:

1. Short int ranges from -32768 to +32767, which is 65536 in total. Hence, maximum number of processing elements up to which this can be scaled up is 32768.
2. -32768 will never be present in the keys.
3. I've made changes to a1.cpp to take n as **unsigned long long int.** The parallel solution is handled for scaling upto n = 10^10.
4. Size of long long int is 8 bytes and size of short int is 2 bytes.
5. This problem virtually excludes running on single processing element. The keys will just be n times 0s and the vector need not be sorted. So, I've ballparked **T1 = 2 * T2** which looks right considering the graphs and times we got.

Sequential:

- This cannot be run on single processing element as per assumption 5.

Parallel:

- The n elements of the given problem get split into p partitions each of size max(n/p)
- The vector of max(n/p) elements can be sorted with counting sort on each node.
- The complexity of the sorting comes down to O((n/p) +2p-1)
- After counting sort, the counters must be reduced across the nodes. The cost of MPI_Allreduce is almost twice as MPI_Bcast. We know All reduce can be implemented using a reduce and a broadcast. So, this should cost us at most O (2*log p).
- Since, every node must have an element, and the least number of distinct keys expected is p, we can't pin the processing nodes to generate sorted keys based on the range of keys. So, we need to equally scatter the keys to be generated over the nodes. This may take at most O(p*p) for sparse vectors to lower bound of O(p) for densely populated vectors.
- As we've used all reduce, the counters are present in all the nodes. We need to repopulate the vectors with the key value and counter. Every node will have at most 2 distinct elements as the range is –(p-1) to +(p-1).
- Calculated times and time, speedup, efficiency graph are given below.

Design considerations:

For communication, that is for transferring counter and key values, I've made the following considerations before proceeding with just MPI_Allreduce ().

- AllReduce:
  - Positives: Nodes won't be idle doing this since all nodes will calculate the proper distribution of the key values in the case of small n values.
  - Negatives: Redundant or unwanted work done by all the nodes.
- Reduce & Scatter / Reduce & Broadcast: (Refer Experiments section for performance metrics)
  - Positives: Comparatively cheaper than AllReduce when we have lesser number of processors.
  - Negatives: More load on a single load for calculating the distribution. Scattering takes more hit when p is greater.
- Reduce_Scatter:
  - Positives: This is seemingly best of all the options considering the speed and scalability.
  - Negatives: Distribution cannot be done after doing in the case of sparse keys.

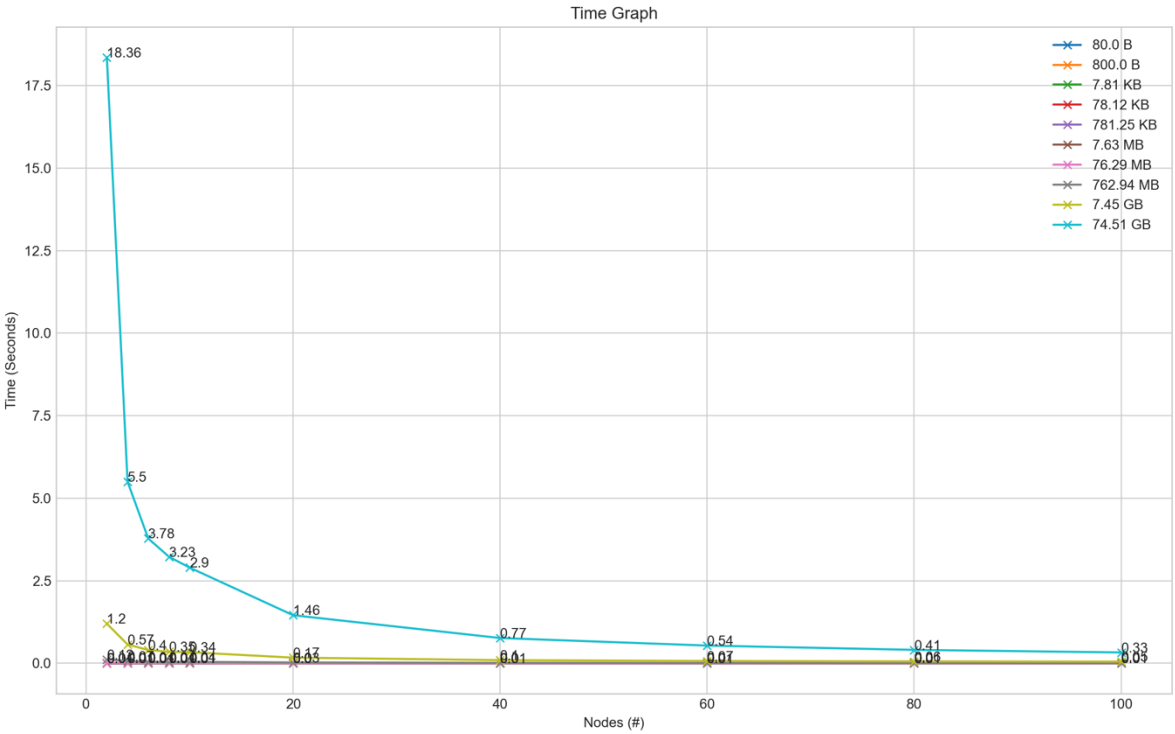
Surprisingly, the seemingly costlier operation, Allreduce gave the best results from the lot. It's not as good as others in lesser number of keys. But it scales up well for big chunk of keys.

Calculated times for Allreduce:

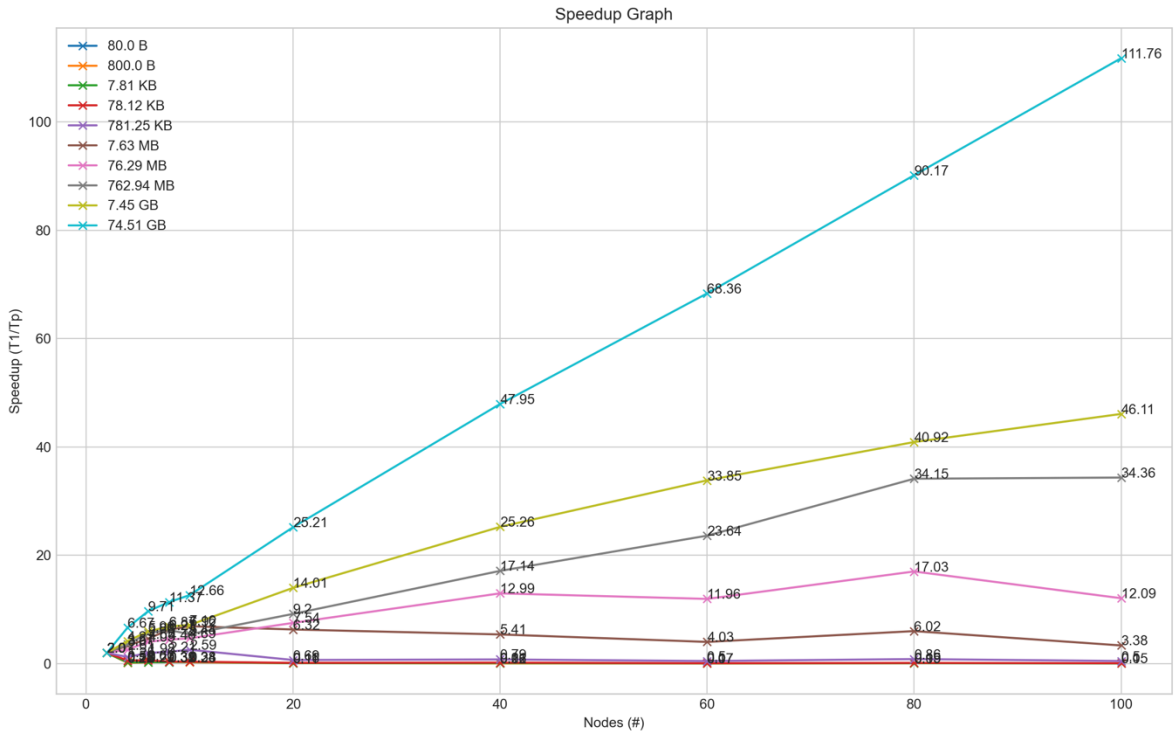
Processing Elements \ Keys (10 ^ X)	1	2	3	4	5	6	7	8	9	10
1*2 = 2	4e-05	4e-05	5e-05	7e-05	0.00033	0.00255	0.01273	0.12036	1.20044	18.35659
1 * 4 = 4	0.00037	0.00049	0.00037	0.00024	0.00053	0.00155	0.00974	0.06742	0.57134	5.50277
1 * 6 = 6	0.00028	0.00032	0.00032	0.00022	0.00034	0.0011	0.00629	0.04252	0.40272	3.7808
1 * 8 = 8	0.00026	0.00022	0.0003	0.00037	0.0003	0.00081	0.00571	0.03863	0.34956	3.22811
1 * 10 = 10		0.00031	0.00027	0.00037	0.00026	0.00074	0.00543	0.04423	0.33549	2.89888
2 * 10 = 20		0.0008	0.00087	0.00088	0.00097	0.00081	0.00338	0.02617	0.17138	1.45653
4 * 10 = 40		0.00147	0.00078	0.00067	0.00084	0.00094	0.00196	0.01404	0.09504	0.7657
6 * 10 = 60		0.00131	0.00139	0.00142	0.00133	0.00127	0.00213	0.01018	0.07093	0.53708
8 * 10 = 80		0.00099	0.00095	0.00092	0.00078	0.00085	0.0015	0.00705	0.05868	0.40718
10 * 10 = 100			0.00168	0.00149	0.00133	0.00151	0.00211	0.00701	0.05207	0.3285

The above graph doesn't have few entries populated since there's a base condition that the number of keys to be sorted cannot be lesser than the number of processing nodes used.

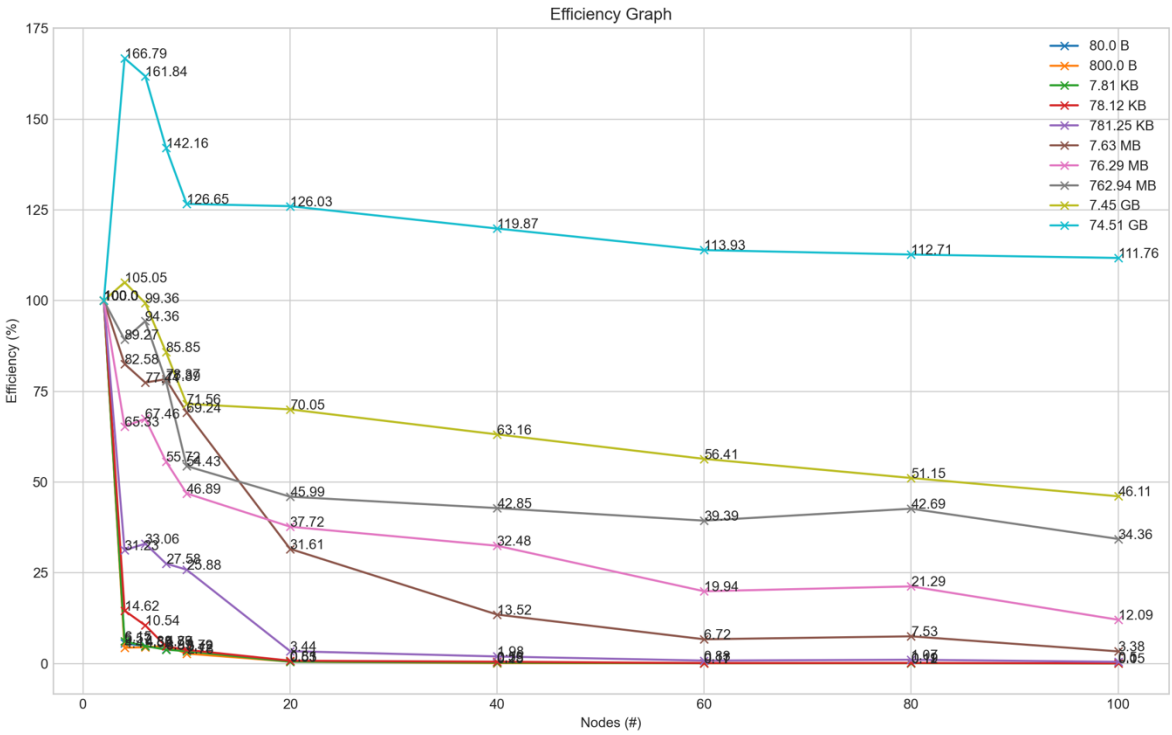
Time graph:



Speedup graph:



Efficiency graph:



Code Scalability:

Is it scalable and efficient?

This looks certainly scalable and efficient. We may want to use optimum number of processing elements based on the size of the keys that need to be sorted. Honestly, sorting 75 MB doesn't require more than 4 processing nodes, until which it's efficient. Sorting 74.51 GB of data, which is greater than the main memories of the nodes used, scales up very well till 100 processing nodes.

How can we improve?

- Ideally, Reduce & scatter (or broadcast) can be used instead of AllReduce for small set of keys. (Check Design Considerations section)
- Counting in lines 31-32 can ideally be done using parallel reduction with multithreading
- Extra usage of memory in line 41 with the array key values. It's required still, but alternative ways shall be used to cut the cost of O(2p-1) there.
- The distribution logic can be optimized. It'll hit us as the size of p increases.
- Resizing vector in line 74. Does it cost more? Needs to be checked.
- Using less barriers

Is it always scalable?

No, it's not always scalable for all combinations of n and p. There's a cutoff p for every n beyond which parallelizing makes no sense because of communication costs.

Scalable in what sense?

This is strongly scalable and it's visible going down the columns of the table. Is it weakly scalable? No. The diagonal where n/p increases doesn't indicate that this is weekly scalable.

What's causing it not to scale?

This is not scalable for lesser number of keys on larger number of processing elements. The communication costs hinder it to scale.

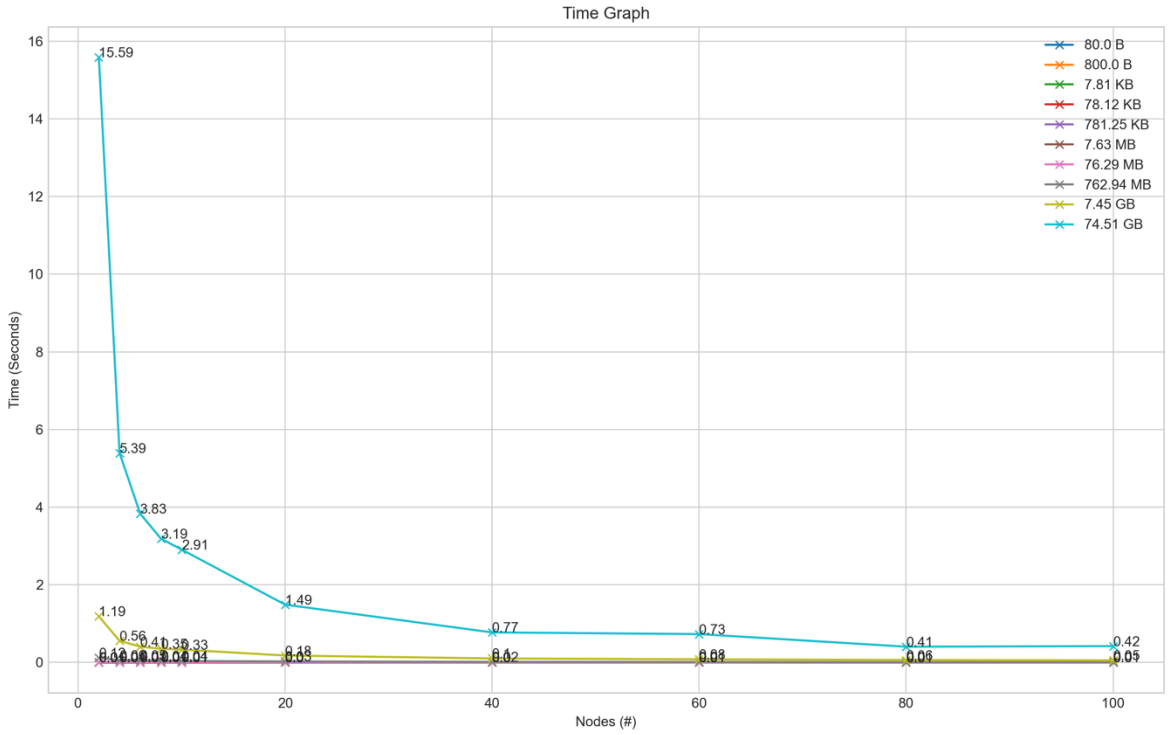
Also, there's an irregularity in scaling down the columns for some number of processing elements even after removing data outliers. This can be explained with few of the processors being sluggish. As we use barriers to sync the processors, this will certainly hit us in overall parallel execution.

Experiments (Not for grading, substantiates few above mentioned claims):

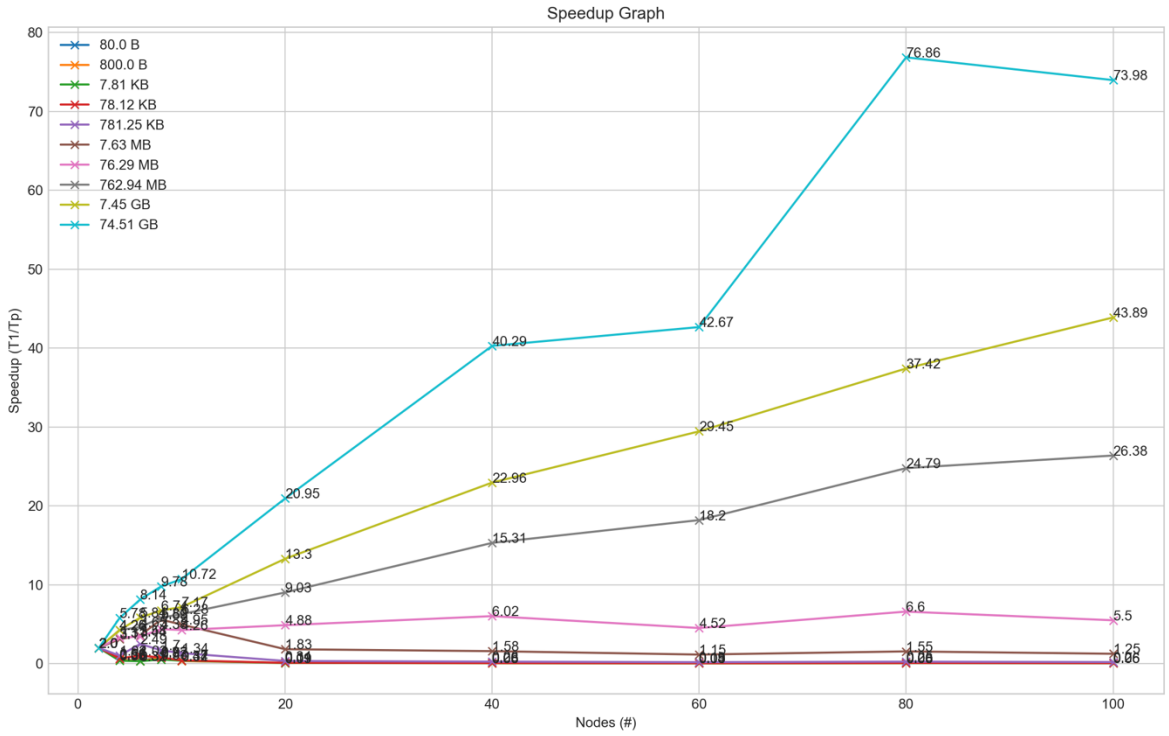
Calculated times for Reduce & Scatter:

Processing Elements \ Keys (10 ^ X)	1	2	3	4	5	6	7	8	9	10
1*2 = 2	0.0001	0.0001	0.0001	0.00012	0.0004	0.00252	0.01209	0.11953	1.18772	15.58785
1 * 4 = 4	0.00035	0.00037	0.00052	0.00037	0.00078	0.00162	0.00731	0.05626	0.55707	5.39331
1 * 6 = 6	0.00063	0.00064	0.00063	0.00024	0.00032	0.00141	0.00703	0.05146	0.40705	3.82786
1 * 8 = 8	0.00022	0.00024	0.00037	0.00032	0.00047	0.0009	0.00552	0.04107	0.3522	3.1884
1 * 10 = 10		0.00057	0.00053	0.00059	0.00059	0.00102	0.00567	0.03806	0.33128	2.90777
2 * 10 = 20		0.00221	0.00223	0.00232	0.00232	0.00275	0.00496	0.02648	0.17863	1.48832
4 * 10 = 40		0.00323	0.00331	0.00328	0.00311	0.00318	0.00401	0.01561	0.10347	0.77383
6 * 10 = 60		0.00419	0.00436	0.00462	0.00423	0.00437	0.00535	0.01313	0.08066	0.73061
8 * 10 = 80		0.0033	0.00321	0.00304	0.00316	0.00324	0.00366	0.00964	0.06348	0.4056
10 * 10 = 100			0.00387	0.00382	0.00391	0.00402	0.0044	0.00906	0.05412	0.42142

Time graph:



Speedup graph:



Efficiency graph: