Parallel Sorting by Regular Sampling

CSE-702
Programming Massively Parallel Systems
Final Report

Pranav Jain
pjain4
50208349

Introduction

Parallel Sorting by Regular Sampling (PSRS) is a parallel partition based sorting algorithm. This sorting routine can be used to sort a large quantity of numbers that can not fit in a single processor. The main idea of taking sampling regularly is that it partitions the data into ordered subsets of approximately equal size which is crucial for partition based sorting routines. Sampling has the advantage that the gathering of the sample is normally highly parallelizable and that the sample, being small, can be processed and communicated with relatively little regard for efficiency.

Algorithm

The code for PSRS is implemented in MPI using C[1]. The algorithm is detailed below

1. Data of size T is generated on each of the N processes randomly, this makes the total data X = T * N
2. The data on each process is sorted locally using quick sort[2]
3. The local data segment in each processor is sampled regularly at indices - 0, X/N^2,2X/N^2, ..., (N-1)X/N^2 and then these samples are sent to a particular processor (last in our case).
4. The last processor on receiving all the samples (N*N) from other processors merges the sorted samples using a priority queue and selects N-1 pivots regularly from the sorted values and broadcasts the pivots to all the processors.
5. Each processor on receiving the pivot values divides it local data to N disjoint partitions (N-1 pivots) and each processor i keeps the i'th partition of the data and sends the jth partition to process j, $\forall j \neq i$.
6. Each processor on receiving all the individually sorted N partitions merges them using a min heap.

Figure 1 illustrates an example[3] of the sorting algorithm for three processes each containing 9 unsorted numbers.

[1]Code can be found at - `https://github.com/pranav9056/PSRS`
[2]Any other optimal algorithm like merge sort etc can also be used
[3]Example taken from Parallel Programming in C with MPI and OpenMP

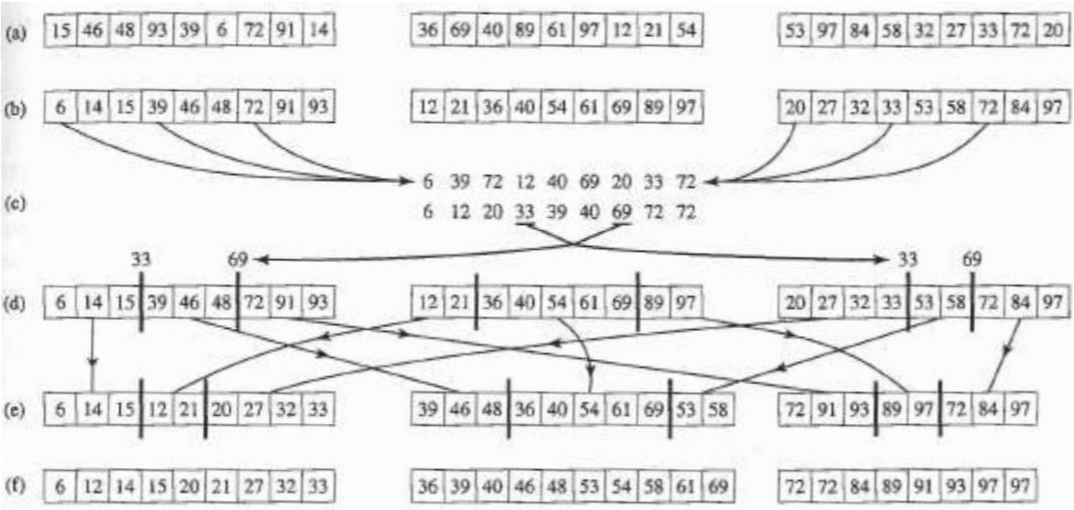


Figure 1) This example illustrates how three processes would sort 27 elements using the PSRS algorrthm. (a) Original unsorted list of 27 elements is divided among three processes. (b) Each process sorts its share of the list using sequential quicksort (c) Each process selects regular samples from its sorted subllst. A single process gathers these samples, sorts them, and broadcasts pivot elements from the sorted list ot samples to the other processes. (d) Processes use pivot elements computed in step (c) to divide their sorted sublists into three parts. (e) Processes perform an all-to-all communication to migrate the sorted sublist parts to the correct processes (f) Each process merges its sorted sublist

Results

The algorithm was run on the CCR cluster for nodes ranging from 2 - 128, nodes here represent a single server and only 1 task was assigned per node to take into account the latency of communication.

PSRS with Constant data

In this experiment,a total data of 512M numbers was sorted. As can be seen from table 1 as the number of nodes increase by a factor of 2 , the running time goes down by a factor of approximately half. However the decrease in running time from 32 - 64 is just by 2 seconds and as you go to 128 the time increases, this shows that at some point, the speedup that you get from paralleization is outweighed by the communication overhead incurred. Figure 2 shows a graph of running time vs nodes.

Nodes	Data per Node(M)	Total Data(M)	Time(s)
2	256	512	119.276
4	128	512	64.335
8	64	512	31.078
16	32	512	20.079
32	16	512	12.591
64	8	512	10.002
128	4	512	11.008

Table 1) Running Times for constant data.

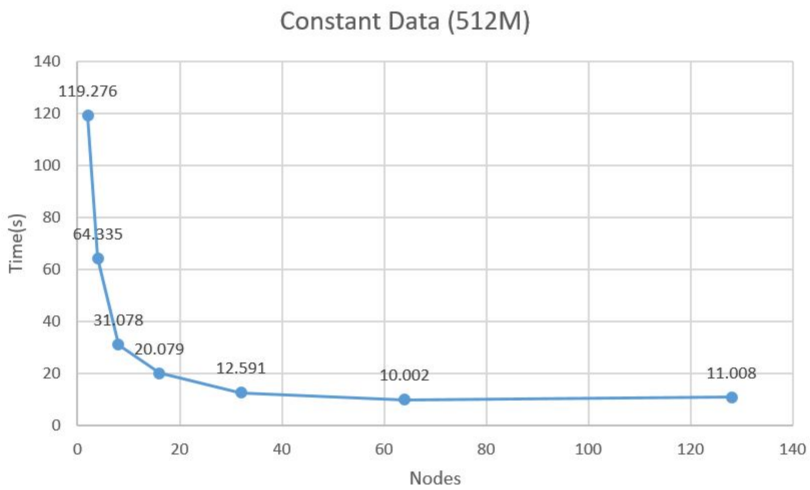


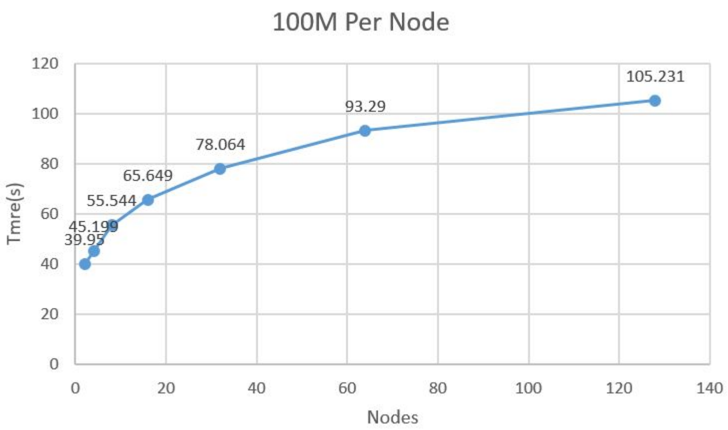
Figure 2) Nodes v/s Time for constant data of 512M

PSRS with increasing data

In this experiment,each node has 100M numbers. As can be seen from table 2 as the number of nodes increase so does the amount of data that is to be sorted,thus the running time also increases. The increase in time gradually increases as the number of nodes are increased. Figure 3 shows a graph of running time vs nodes

Nodes	Data per Node(M)	Total Data(M)	Time(s)
2	100	200	39.951
4	100	400	45.199
8	100	800	55.544
16	100	1600	65.649
32	100	3200	78.064
64	100	6400	93.291
128	100	12800	105.231

Running Times for increasing data.



Nodes v/s Time for increasing data

Advantages of PSRS

- Good Load balancing is achieved generally.
- Number of processors don't need to be in powers of 2 as with other algorithms like hypercube quick sort etc.
- Repeated communications of a same value are avoided as the pivots are sent only once.

Resources

- https://www.uio.no/studier/emner/matnat/ifi/INF3380/v10/undervisningsmateriale/inf3380-week12.pdf
- https://webdocs.cs.ualberta.ca/~jonathan/publications/parrallel_computing_publications/psrs1.pdf
- https://ubccr.freshdesk.com/support/solutions/folders/13000001591