Sample Sort Using MPI

Experimentation data for Serial computation

For serial computation we have chosen the serial code with 12 buckets and obtained the runtimes with size = 120000, 1200000, 12000000, 120000000. These results will be used as the baseline to compare with the parallel models.

(N)	Runtime (sec)
120000	0.058558
1200000	0.407464
12000000	3.881106
120000000	44.601656

MPI Implementation

Overview of the Algorithm and suggested optimization

1. Rank 0 creates the data set and scatters it to the P processes.
2. Each process does local sorting and select p-1 splitters.
3. All the p(p-1) splitters from p processes are gathered at rank 0.
4. Rank 0 selects a set of p-1 global splitters after sorting the sample of p(p-1) splitters and broadcasted that to P processes.
5. Each process put its local data set into p local buckets based on the p-1 global splitters.
6. Each process rearranges its buckets using MPI_Alltoall so that P0 has all the data < splitter[0], P1 has all the data < splitter[1] and soon.
7. Each process finally sorts their local data.

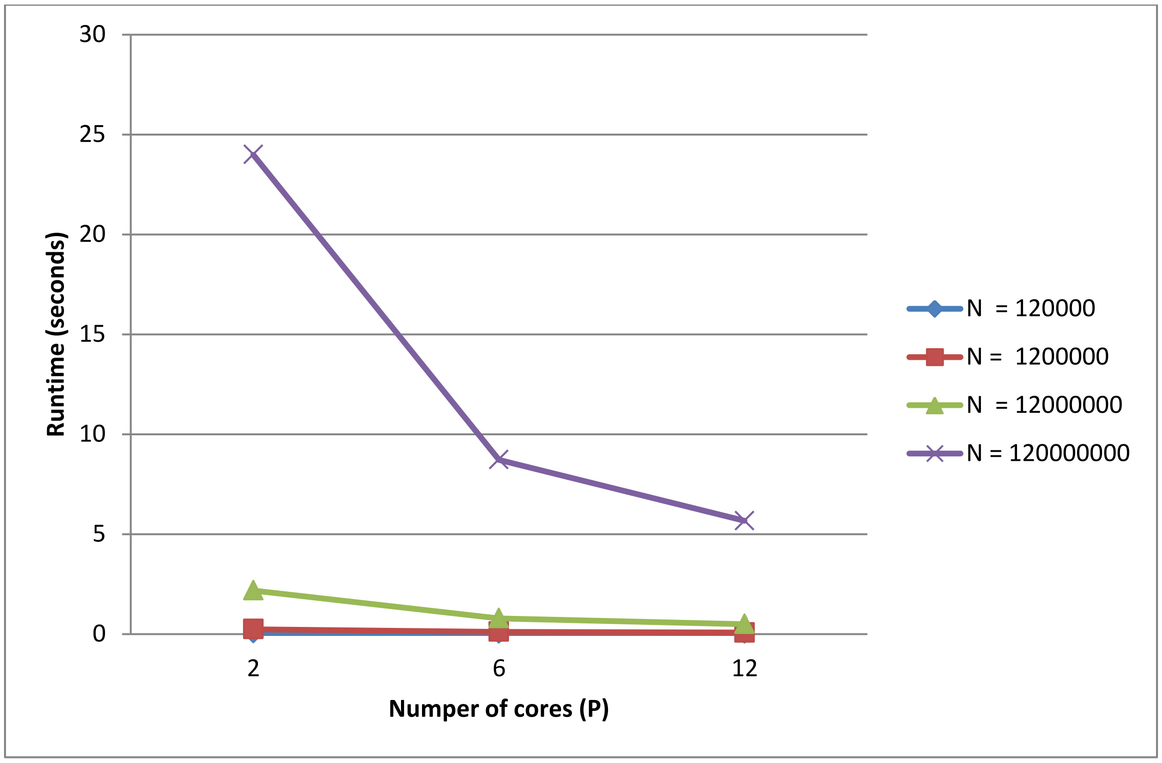
Experimentation data

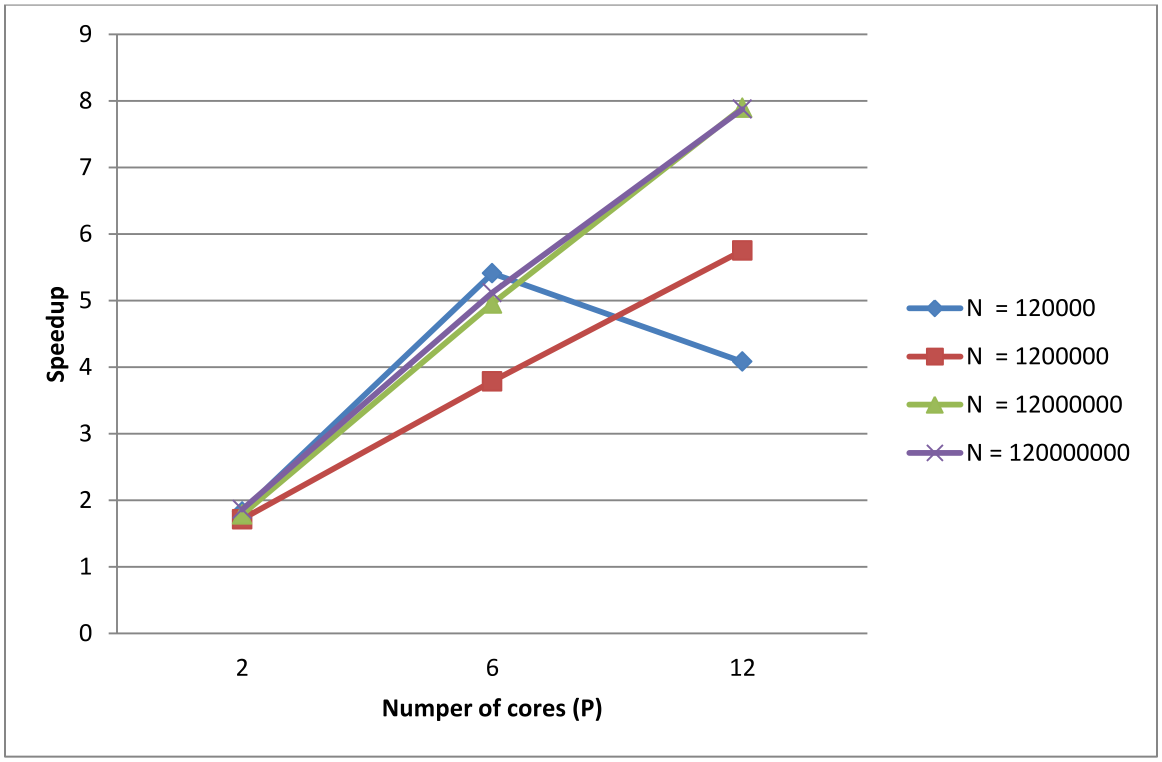
Tabular Representation

Note: The speedup is calculated using (serial runtime calculated above / runtime for P).

P	Runtime (seconds)	Speedup
N = 120000		
2	.032027	1.828349
6	.010830	5.407017
12	.014347	4.081379
N = 1200000		
2	0.237942	1.71245
6	0.107647	3.785186
12	0.070884	5.748321
N = 12000000		
2	2.17792	1.782024
6	0.78367	4.952474
12	0.491693	7.893352
N = 120000000		
2	23.98413	1.859632
6	8.715355	5.117594
12	5.66478	7.873501

Graphical Representation





Analysis

Q. Discuss the results you obtained. Were you able to gain a linear or near-linear speedup? Why or why not?
Answer.

Discussion on results

1. For N = 120000, the speedup drops down after P = 6.
    - Here the communication cost involved when P > 6 is much greater than the computation gain by sharing the workload among processors and as a result the performance degrades after P = 6.

2. For a given N, the speedup increases with value of P.
    - With more number of P, the workload per process (N/ P) decreases leading to better overall performance (or lesser runtime).

3. With very large values of P, the performance curve breaks the linear behavior.
    - Let take the example of N=1200000. Had the curve been linear, then the expected value of speedup at P = 12 is 7.57, but we got 5.748. Similarly, for N = 12000000, the obtained value of speedup is 7.893352 at P = 12 which degrades from the expected value of 9.904 (which should be if the curve is linear). The same is true for N = 120000000.
      In the examples mentioned above, the communication cost involved when P = 12 is much greater than the computation gain by sharing the workload among processors and as a result the performance degrades at P = 12.

Linear or near linear speedup

- The speedup that we obtain **is near linear.**
    - With initial incremental values of P, the speedup is increasing proportionately as the workload is distributed proportionately among the processes. But with larger values of P, the communication cost outweighs the gain with workload distribution and as a result the speedup curve started degrading at values where the computation/ communication ratios are low. (This is substantiated in Analysis 3 above.)

Q. A bucket sort is similar to a sample sort, except the endpoints of the buckets are predetermined as evenly spaced along the interval from the minimum to the maximum value. Therefore, no sample is taken. Instead, the algorithm immediately splits the data into the buckets and then sorts the buckets. When could a bucket sort be better than a sample sort? When would it be worse?

Answer.
Bucket sort will be better than sample sort when the array elements are uniformly distributed over the interval [min, max], where min, max are the minimum and maximum elements of the array.

Reason
Bucket sort is used for sorting an array of size n whose values are uniformly distributed over an interval [a,b]. In this algorithm the interval is divided in p equal sized buckets and each element is placed in the appropriate bucket. Now if the elements of the array are uniformly distributed over the interval [a, b], then each bucket will have n/p elements and the time complexity comes out to be O(n*log(n/p)).

On the other hand, for sample sort, the complexity comes out of the following factors (assume n as the array size and p be the number of processes):

- Local sort: O((n/p) log (n/p))
- Selection of p-1 sample splitters: O(p)
- Sorting the sample of p(p-1) slitters: O(p^2log(p))
- Selection of p-1 global splitters: O(p)
- Partitioning the block into p sub-block based on the p-1 splitters: O(n *log(n/p))
- + Communication cost

Adding all the above factors, this time complexity comes out to be poor as compared to the bucket sort.

Bucket sort will give worst performance when the elements are not uniformly distributed over the interval [min, max].
Reason
In the worst case, we may have n elements in a single bucket resulting in the time complexity of n*log (n). With n >> p, this will give the worse performance as compared to sample sort.

Histogram Sort Using MPI

Experimentation data for Serial computation

For serial computation we have chosen the serial code with 12 buckets and obtained the runtimes with size = 120000, 1200000, 12000000, 120000000. These results will be used as the baseline to compare with the parallel models.

(N)	Runtime (sec)
120000	0.060661
1200000	0.449810
12000000	4.006512
120000000	45.562630

MPI Implementation

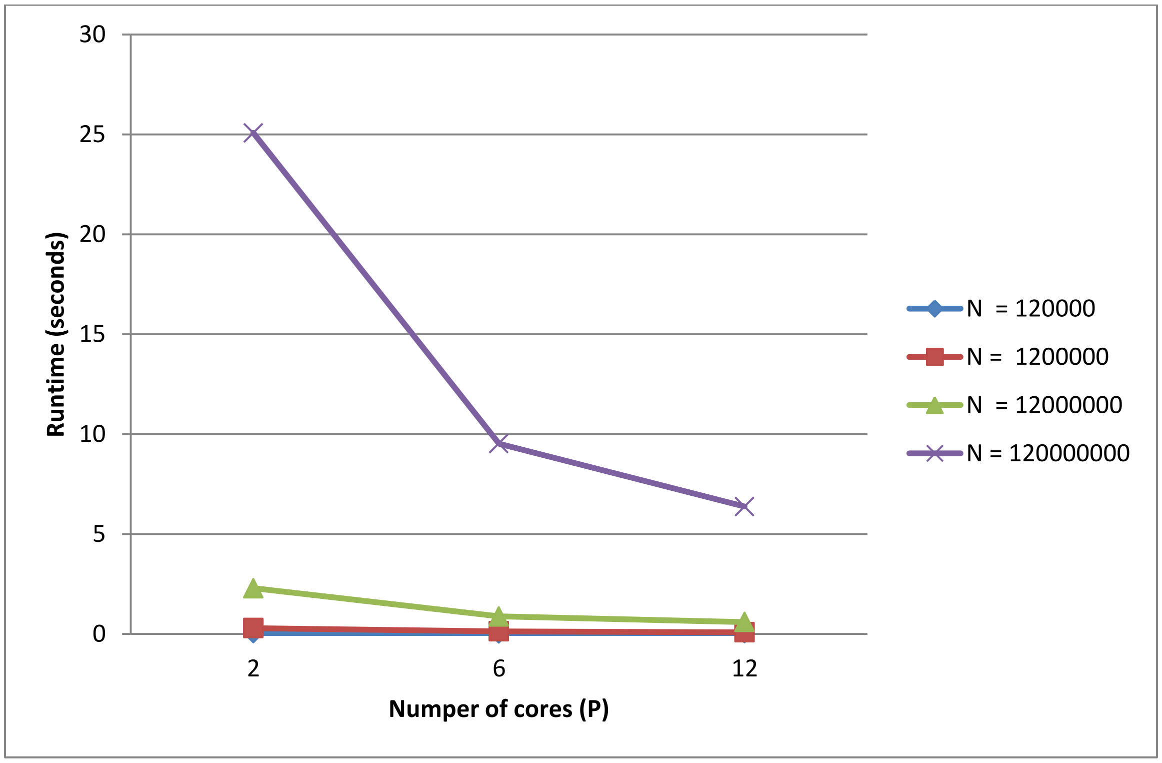
Experimentation data

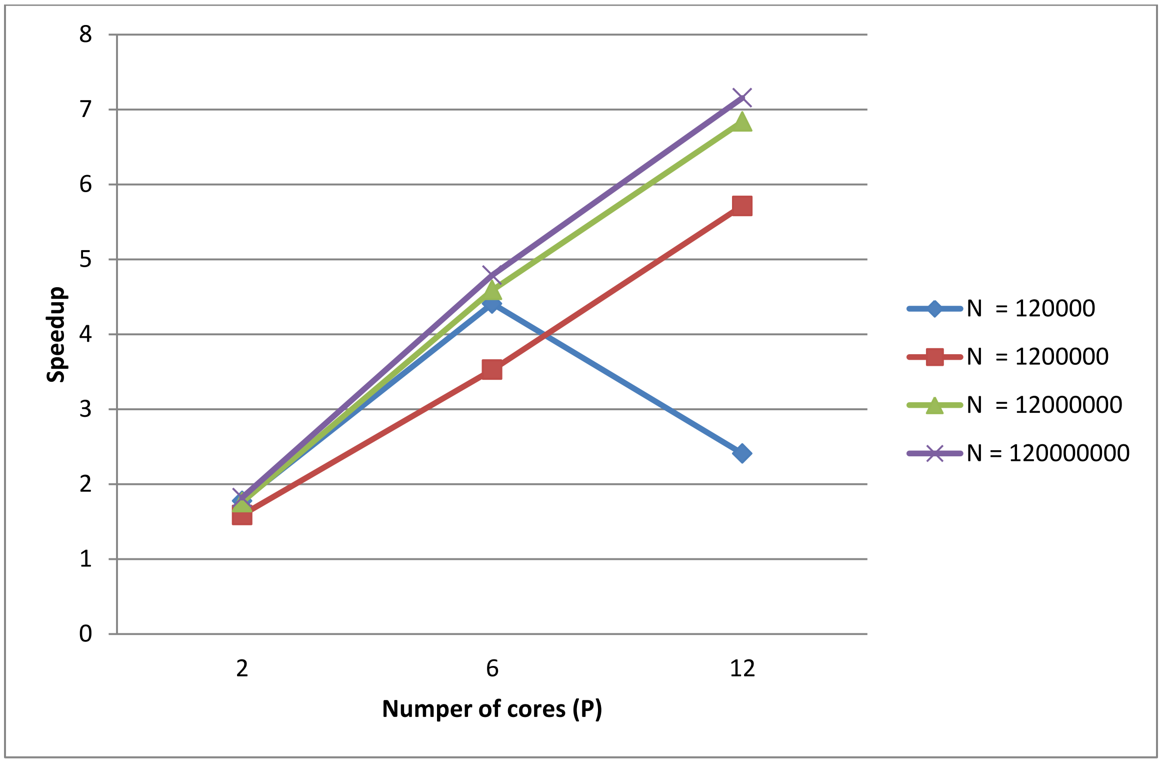
Tabular Representation

Note: The speedup is calculated using (serial runtime calculated above / runtime for P).

P	Runtime (seconds)	Speedup
N = 120000		
2	.034180	1.774720
6	.013763	4.407349
12	0.025200	2.407182
N = 1200000		
2	.283378	1.587312
6	.127599	3.525173
12	.078755	5.711481
N = 12000000		
2	2.280247	1.757051
6	.873000	4.589360
12	.5860612	6.836337
N = 120000000		
2	25.065012	1.817778
6	9.515479	4.788264
12	6.369941	7.152755

Graphical Representation





Analysis

Q. Discuss the results you obtained. Were you able to gain a linear or near-linear speedup? Why or why not?
Answer.

Discussion on results

1. For N = 120000, the speedup drops down after P = 6.
   - Here the communication cost involved when P > 6 is much greater than the computation gain by sharing the workload among processors and as a result the performance degrades after P = 6.

2. For a given N, the speedup increases with value of P.
   - With more number of P, the workload per process (N/ P) decreases leading to better overall performance (or lesser runtime).

3. With very large values of P, the performance curve breaks the linear behavior.
   - Let take the example of N=1200000. Had the curve been linear, then the expected value of speedup at P = 12 is 7.05, but we got 5.711481. Similarly, for N = 12000000, the obtained value of speedup is 6.83 at P = 12 which degrades from the expected value of 9.17 (which should be if the curve is linear). The same is true for N = 120000000.
     In the examples mentioned above, the communication cost involved when P = 12 is much greater than the computation gain by sharing the workload among processors and as a result the performance degrades at P = 12.

Linear or near linear speedup

- The speedup that we obtain **is near linear.**
    - With initial incremental values of P, the speedup is increasing proportionately as the workload is distributed proportionately among the processes. But with larger values of P, the communication cost outweighs the gain with workload distribution and as a result the speedup curve started degrading at values where the computation/communication ratios are low. (This is substantiated in Analysis 3 above.)