Implementation Of Parallel Shell Sort Using MPI

CSE 633 – Parallel Algorithms (Spring 2017)

Prasad Salvi

Instructor: Dr. Russ Miller

University at Buffalo The State University of New York

Insertion Sort

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. In each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

$\boxed{6}\ \boxed{5}\ 3\ \ 1\ \ 8\ \ 7\ \ 2\ \ 4$

6 5 3 1 8 7 2 4

Insertion Sort Example

Shell Sort

The shell sort, improves on the insertion sort by breaking the original list into a number of smaller sublists, each of which is sorted using an insertion sort.

The unique way that these sublists are chosen is the key to the shell sort.

Instead of breaking the list into sublists of contiguous items, the shell sort uses an increment i, sometimes called the **interval**, to create a sublist by choosing all items that are i items apart.

Shell Sort Example

54	26	93	17	77	31	44	55	20

Let Interval=3, Make the virtual sublist of all values located at interval of 3 position.

54	26	93	17	77	31	44	55	20

We compare values in each sub-list and swap them (if necessary) in the original array.

17	26	20	44	55	31	54	77	93

Shell Sort Example (Cont.)

17	26	20	44	55	31	54	77	93

Decrement, Interval=2, Make the virtual sublist of all values located at interval of 2 position.

17	26	20	44	55	31	54	77	93

We compare values in each sub-list and swap them (if necessary) in the original array.

17	26	20	31	54	44	55	77	93

Shell Sort Example (Cont.)

17	26	20	31	54	44	55	77	93

Decrement, Interval=1, perform standard insertion sort

17	20	26	31	44	54	55	77	93

The number of swap operations performed for this example

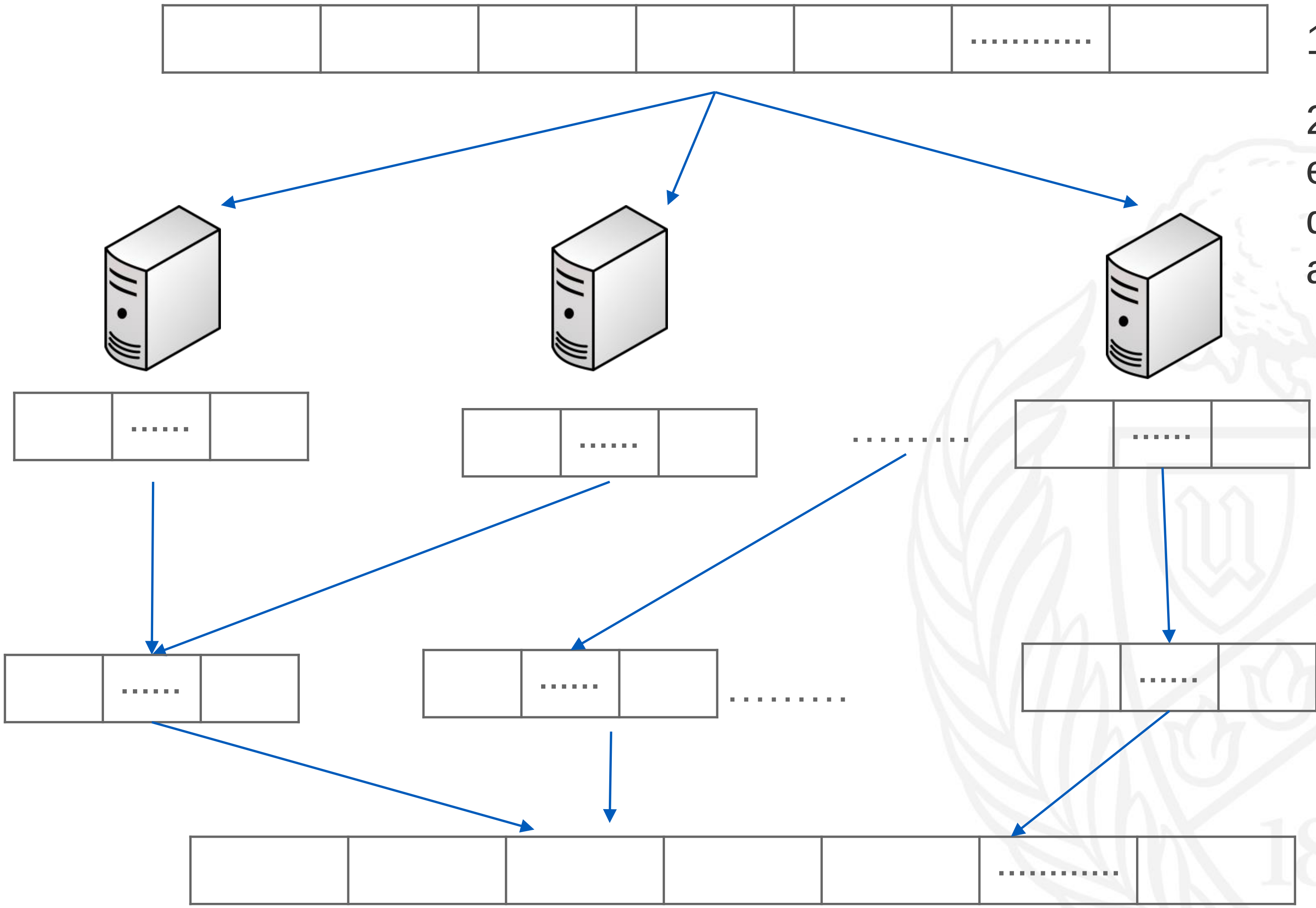
- Insertion Sort: 22
- Shell Sort: 10

Sequential Shell Sort

1. Initialize Data of size N elements and interval value h
2. Divide the Data into virtual sub-lists of elements h interval apart
3. Perform Insertion sort on these smaller sublists
4. Decrement the interval h, and Repeat Until h=1 and Data is sorted

```
// Sequential algorithm of Shell sort
ShellSort ( double A[], int n, int incr )
{
  while( incr > 0 )
  {
    for ( int i=incr+1; i<n; i++ )
    {
      j = i-incr;
     while ( j > 0 )
         if ( A[j] > A[j+incr] )
         {
           swap(A[j], A[j+incr]);
           j = j - incr;
         }
         else
           j = 0;
     }
     incr = incr-1;
  }
}
```

Parallel Shell Sort



1. N elements
2. N/P elements distributed across P PE
3. Perform shell sort in each PE in parallel
4. Compare & Merge elements of PE in parallel

Merge Step Approach

Merge Sort Routine

23	48	50	75

10	65	70	90

10	23	48	50	65	70	75	90

Shuffle and Shell sort

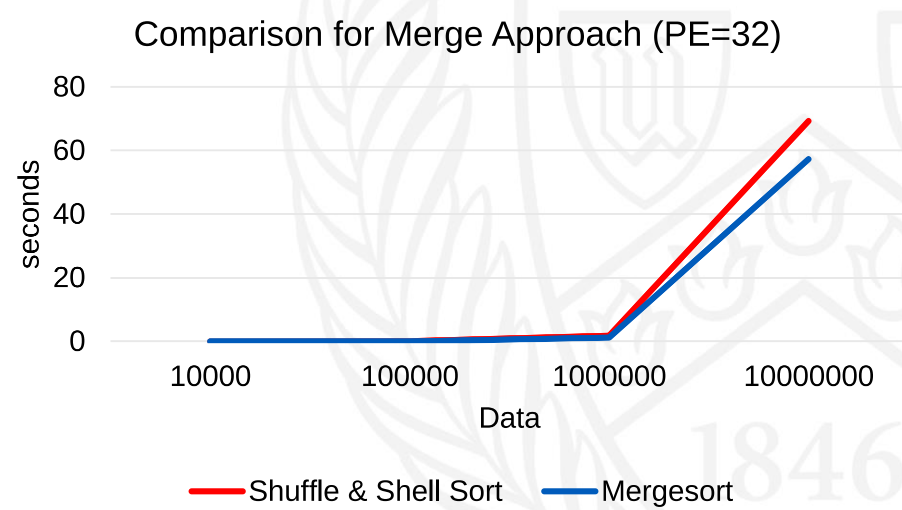
23	48	50	75

10	65	70	90

23	10	48	65	50	70	75	90

10	23	48	50	65	70	75	90

Data	**Shuffle & Shell Sort**	**Mergesort**
10000	0.001856	0.000781
100000	0.032809	0.021486
1000000	1.783714	1.174828
10000000	69.236249	57.307384



Parallel Shell Sort

Parallel_Shell_Sort(Data,N,h,P)

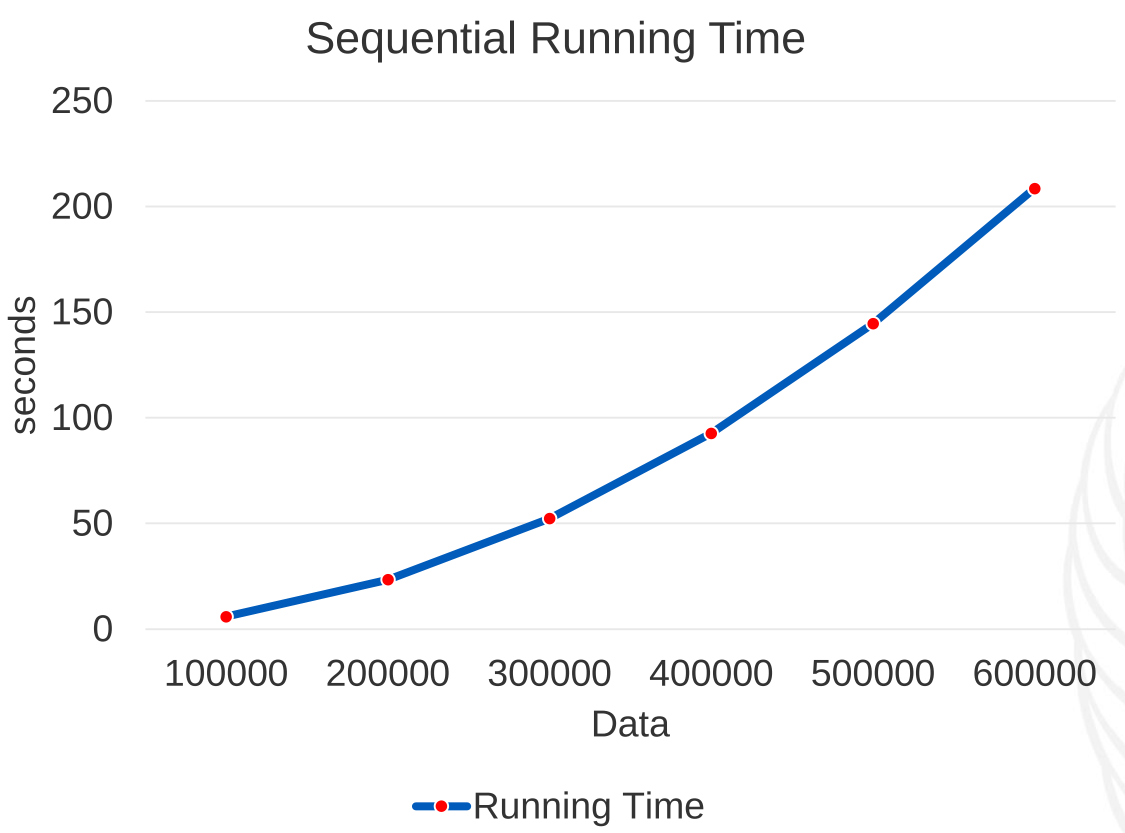
1. Initialize Data of size N elements and interval value h on P_0
2. Broadcast (MPI_Scatter) data elements across P Processors, each holding (N/P) data
3. In Parallel perform Shell sort on P processor
    1. Divide the Data into virtual sub-lists of elements h interval apart
    2. Perform Insertion sort on these smaller sublists
    3. Decrement the interval h, and Repeat Until h=1 and Data is sorted for each P
4. For 2^i where i=0,1,2,… till 2^i = P, Merge the data across P_i & P_{i+1} using merge sort operation in parallel till entire list is sorted

Project Execution

- **Parameters tested:**
  - **Sequential Algorithm**
    - Number of PE (Constant) vs Data (Variable)
  - **Parallel Algorithm**
    - Number of PE (Variable) vs Data (Constant ~ 10000,100000,1000000)
    - Number of PE (Constant) vs Data (Variable)

- **Project Milestones:**
  - MPI Learning & Algorithm Design: 07 Feb – 25 Feb
  - Implementing code & Debug: 20 Feb – 30 Mar
  - Test Runs & data collection : 20 Mar-25 Apr

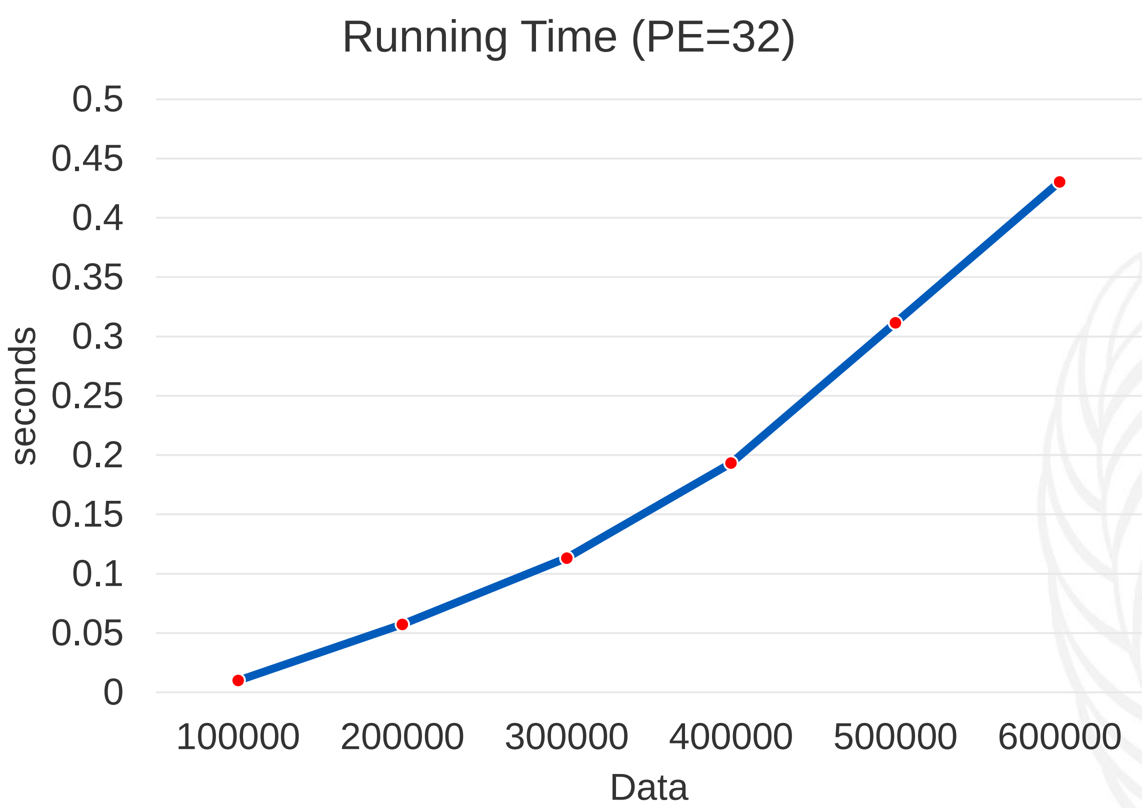
Sequential Execution



Data	Running Time
100000	5.843089
200000	23.379516
300000	52.322923
400000	92.687364
500000	144.582511
600000	208.581895

Parallel Execution

Variable Data and Constant PE

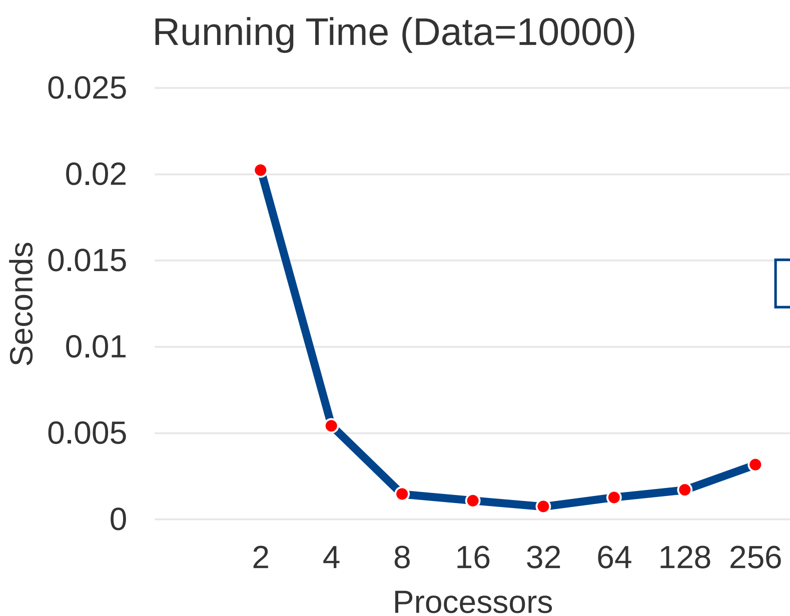


Data	Running Time
100000	0.009948
200000	0.057285
300000	0.113133
400000	0.193175
500000	0.311486
600000	0.430405

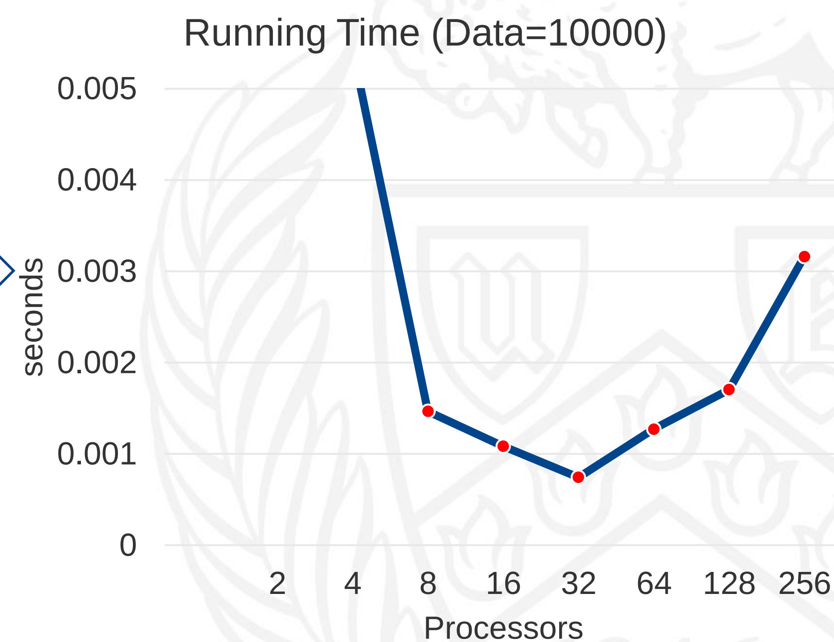
Parallel Execution

Constant Data and Variable PE

PE	Running Time
2	0.020242
4	0.005426
8	0.001465
16	0.001083
32	0.000743
64	0.001271
128	0.001705
256	0.003161



Scaled



Parallel Execution

Constant Data and Variable PE

Data	Processors								
AVG	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**
100	0.000044	0.000042	0.000028	0.000245	0.000305	0.000494	0.002169	0.004168	0.005229
1000	0.000749	0.000255	0.000149	0.000156	0.000216	0.000376	0.000788	0.001871	0.002482
10000	0.068646	0.020242	0.005426	0.001465	0.001083	0.000743	0.001271	0.001705	0.003161
100000	5.809234	1.485181	0.403242	0.138614	0.030226	0.011225	0.005582	0.004493	0.007636
1000000	574.6069	144.7589	36.44519	9.532319	2.632521	0.795049	0.246187	0.101531	0.047912
10000000	58505.77	14713.06	3626.309	908.2444	227.5709	57.07818	15.10796	4.070404	1.487145

References

- Dr. Russ Miller; Laurence Boxer : Algorithms Sequential and Parallel: A Unified Approach (Third Edition)
- Dr. D. L. Shell ; A high-speed sorting procedure". Communications of the ACM. 2 (7): 30–32
- Dr. Matt Jones (CCR) : Tutorial & Training on MPI & CCR Infrastructure
- http://www.cse.buffalo.edu/faculty/miller/Courses/CSE633/Narendran-Sankaran-Spring-2014-CSE633.pdf

Appendix

Data Collected

[Data Collected for Parallel Implementation of Shell Sort on CCR Servers]

Thank You