

Laboratory Practice 1 :
High Performance Computing Mini Project
Parallel Bubble Sort

Project ID: 13

Project created by :
Prathamesh Chaudhari (50)
Sanket Sutar(49)
Rupesh Deshmukh(52)
Piyush Patil(51)

Project Guide: Prof. J. R. Mankar

December 24, 2020

Contents

1	Problem Statement	1
2	Objectives	2
3	Bubble Sort	3
4	Parallel Bubble sort	4
5	OpenMP	5
6	Screenshot of Output	6
7	Outcomes	8
8	Conclusion	9

List of Figures

3.1	Pseudocode of BubbleSort	3
4.1	Pseudocode of Parallel BubbleSort	4
6.1	Number of elements sorted vs Time	6
6.2	Output	7

Chapter 1

Problem Statement

To implement Parallel bubble Sort using OpenMP API.

.

Chapter 2

Objectives

- To implement parallel bubble sort by using OpenMp.
- To implement sequential bubble sort.
- To compare parallel bubble sort with sequential bubble sort.

Chapter 3

Bubble Sort

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of (n^2) where n is the number of items.

```
procedure bubbleSort( list : array of items )  
  
    loop = list.count;  
  
    for i = 0 to loop-1 do:  
        swapped = false  
  
        for j = 0 to loop-1 do:  
  
            /* compare the adjacent elements */  
            if list[j] > list[j+1] then  
                /* swap them */  
                swap( list[j], list[j+1] )  
                swapped = true  
            end if  
  
        end for  
  
        /*if no number was swapped that means  
        array is sorted now, break the loop.*/  
  
        if(not swapped) then  
            break  
        end if  
  
    end for  
  
end procedure return list
```

Figure 3.1: Pseudocode of BubbleSort

Chapter 4

Parallel Bubble sort

Implemented as a pipeline.

```
void *Parallel_bubble_sort(void *arg) |
{
    int id, i;
    Get_id(id);
    int lsize = size/no_threads;
    if (size % no_threads != 0)
        lsize++;
    int my_start = id*lsize;
    int my_end = min(size-1, (id+1)*lsize);
    for (i=-id; i<size+no_threads-id; i++)
        if (i >= 0 && i<size)
            Local_loop(my_start, my_end);
    Barrier(no_threads);
}
if (id == no_threads-1)
    Output_array();
}
```

Figure 4.1: Pseudocode of Parallel BubbleSort

Chapter 5

OpenMP

OpenMP is a widely adopted shared memory parallel programming interface providing high level programming constructs that enable the user to easily expose an application's task and loop level parallelism in an incremental fashion. The range of OpenMP applicability was significantly extended recently by the addition of explicit tasking features. The OpenMP is the dominant programming model for heterogeneous systems and adopted by Intel, Clear Speed, PGI and CAPS SA. The idea behind OpenMP is that the user specifies the parallelization strategy for a program at a high level by providing the program code.

Chapter 6

Screenshot of Output

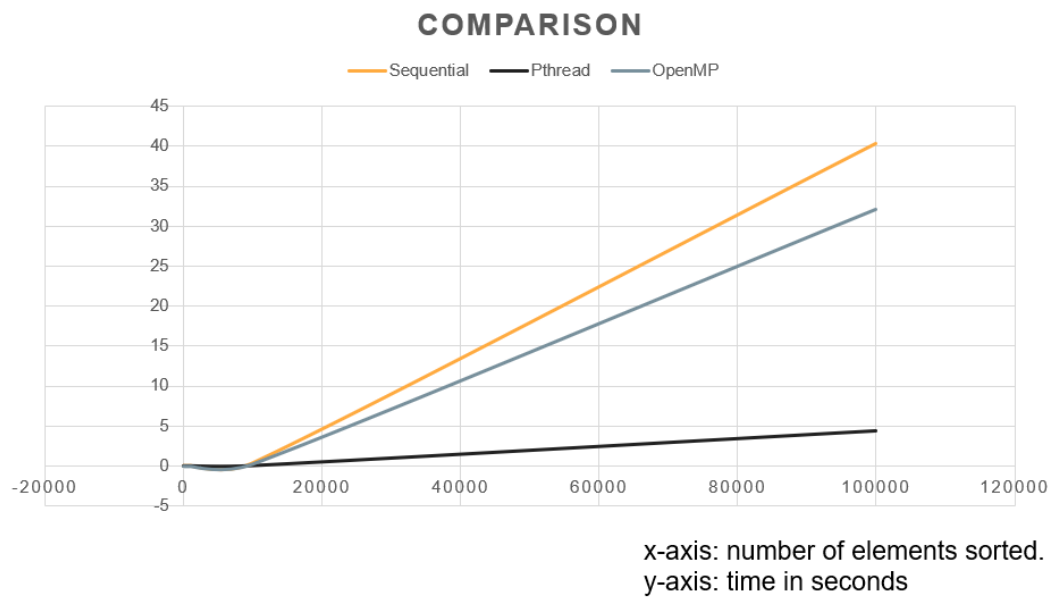


Figure 6.1: Number of elements sorted vs Time

```

-----Before Sorting-----

-----OUR INTEGER ARRAY-----
354      96      944      745      254      142      116      624      886      89      477      819      772      674
      292      754      611      821      974      839      400      970      626      89      605      832      747
      341      406      552      125      186      899      347      202      183      552      302      775      727
      864      15      384      151      679      364      125      478      931      136      625      626      648
      608      179      649      421      149      745      592      527      658      810      764      110      200
      818      861      581      131      237      721      692      652      789      85      35      609      814
      143      449      20      482      310      711      103      148      547      311      762      651      358
      268

-----INTEGER ARRAY PRINTED-----

-----After Sorting-----

-----OUR INTEGER ARRAY-----
15      20      35      85      89      89      96      103      110      116      125      125      131      136
      143      148      149      151      179      183      186      200      202      237      254      268      280
      302      310      311      341      347      354      358      364      384      400      406      421      449
      478      482      514      527      547      552      552      578      581      592      605      608      609
      618      624      625      626      626      639      648      649      651      652      658      674      679
      711      721      727      745      745      747      754      762      764      772      775      783      789
      814      818      819      821      832      839      861      864      886      899      931      944      969
      974

-----INTEGER ARRAY PRINTED-----
Bubble Sort Open MP took 0.044000 seconds to sort the integer array.

-----
Process exited after 0.1972 seconds with return value 0
Press any key to continue . . .

```

Figure 6.2: Output

Chapter 7

Outcomes

- The sequential bubble sort is inefficient sorting method for common usage.
- For large arrays parallel algorithms perform far better than sequential algorithms.
- parallel algorithms has better CPU utilization for large arrays.

Chapter 8

Conclusion

Hence we have successfully implemented parallel bubble Sort by applying parallelism using OpenMP constructs.