CS 5473 **Homework 4** **May 5, 2017**

Due: May 5, 2017

1 Count Sort

A serial code for count sort algorithm was given. To sort a given array, the algorithm considers an empty array, then pick the first element from unsorted array and go along the array to find the position or place of the element in an ordered array. It repeats the procedure with every single element array up to the last elements.

1.1 Parallelization

To parallelize the code with OpenMP, count, i, and j should be defined as private and tmp, a, and n are shares variable. The loop for i would be:

```
#       pragma omp parallel num_threads (thread_count) \
        default (none) private (count, i, j) shared (n, a, tmp)

        for (i=0; i<n; i++){
           count = 0;
           for (j = 0; j < n; j++)
                        if (a[j] < a[i])
                                count++;
                        else if
                                (a[j] == a[i] && j < i)
                                        count++;
            tmp[count] = a[i];
        }
```

1.2 OpenMP Code

Parallel program was implemented by OpenMP is available as attachment.

1.3 Analysis

The written code ran on Slurm machine and following results have been achieved by different size of arrays and different threads:

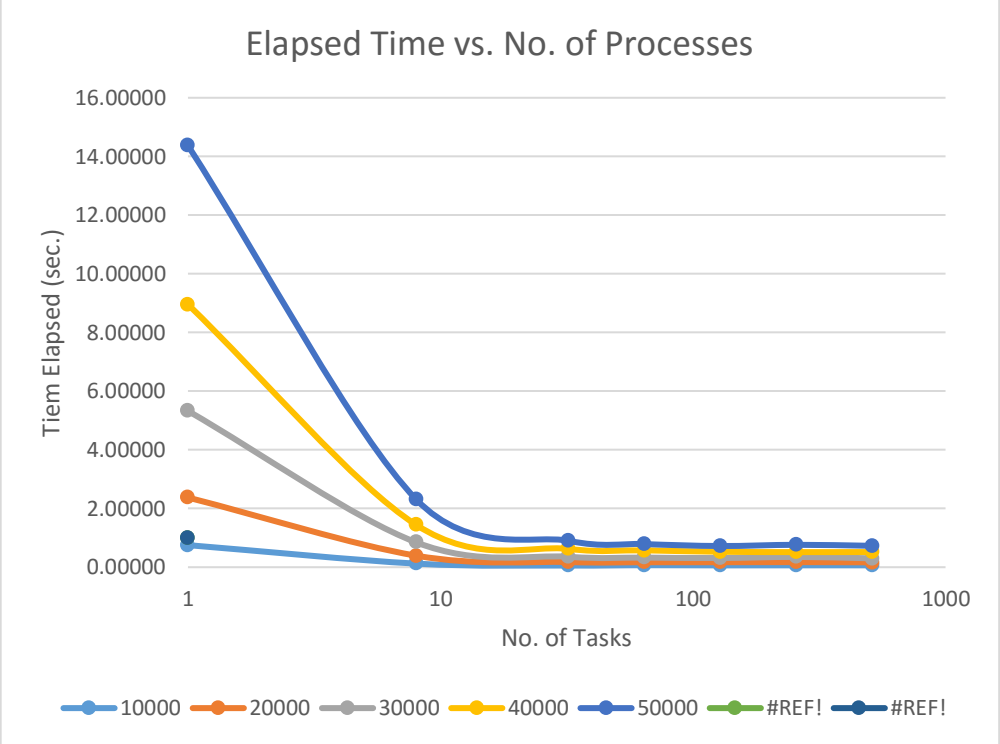
		P						
		1	8	32	64	128	256	512
N	10000	0.75230	0.12251	0.05812	0.06921	0.06354	0.06541	0.06471
	20000	2.37381	0.39273	0.17571	0.16942	0.16935	0.17342	0.16521
	30000	5.32698	0.84895	0.35622	0.32742	0.31212	0.35047	0.30156
	40000	8.95615	1.44599	0.62460	0.57454	0.52581	0.51420	0.51698
	50000	14.36951	2.31754	0.90217	0.78719	0.72206	0.76314	0.72845

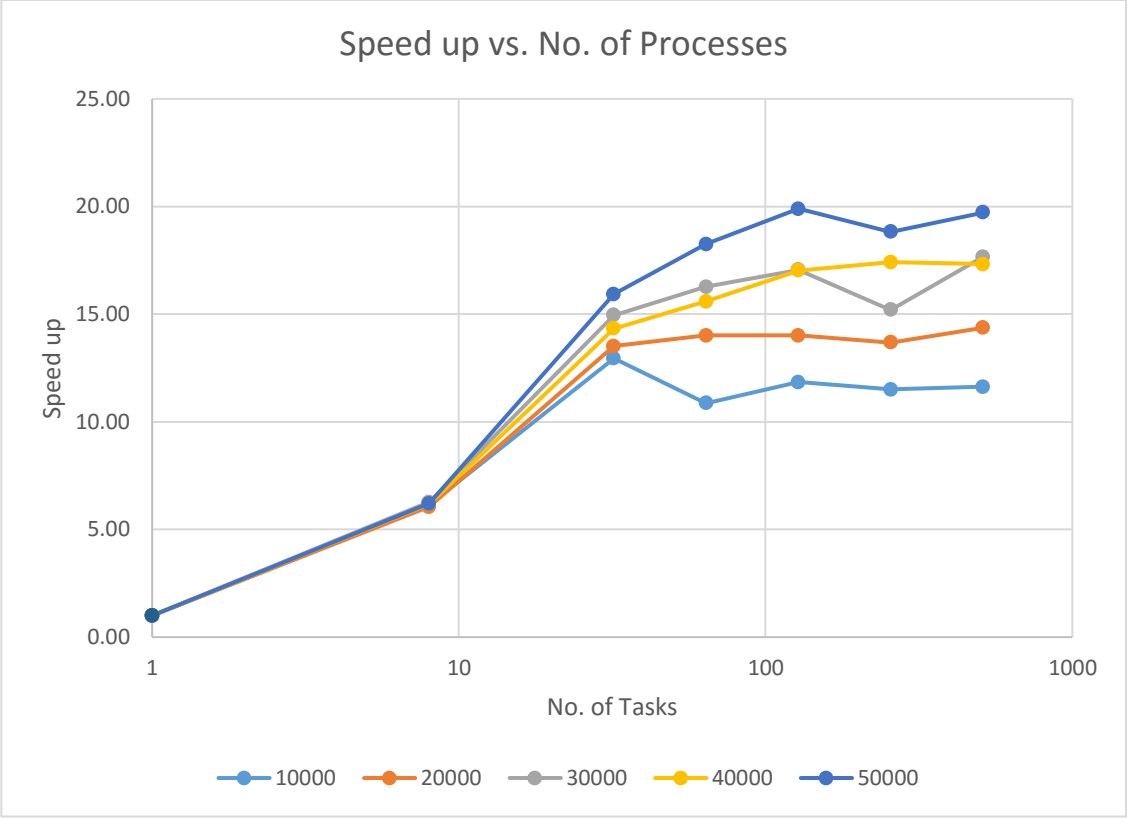
As previous home works the speed up and efficiency were also calculated and summarized in the following tables:

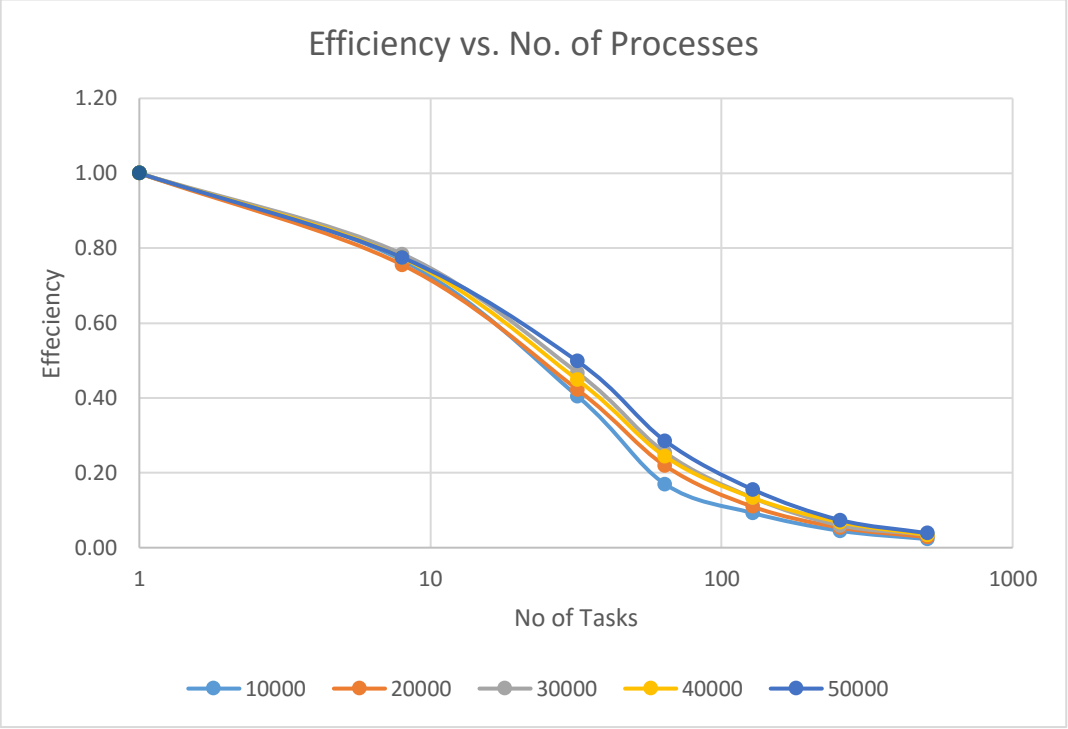
		P						
		1	8	32	64	128	256	512
Sp	10000	1.00	6.14	12.94	10.87	11.84	11.50	11.63
	20000	1.00	6.04	13.51	14.01	14.02	13.69	14.37
	30000	1.00	6.27	14.95	16.27	17.07	15.20	17.66
	40000	1.00	6.19	14.34	15.59	17.03	17.42	17.32
	50000	1.00	6.20	15.93	18.25	19.90	18.83	19.73

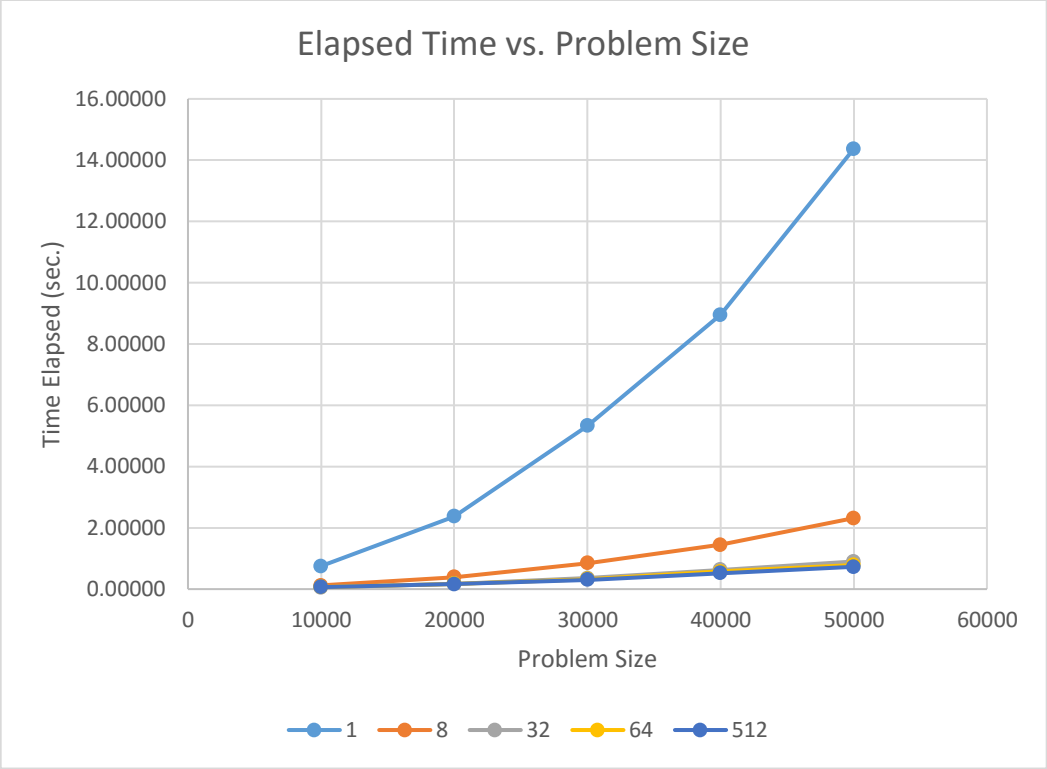
		P						
		1	8	32	64	128	256	512
Ep	10000	1.00	0.77	0.40	0.17	0.09	0.04	0.02
	20000	1.00	0.76	0.42	0.22	0.11	0.05	0.03
	30000	1.00	0.78	0.47	0.25	0.13	0.06	0.03
	40000	1.00	0.77	0.45	0.24	0.13	0.07	0.03
	50000	1.00	0.78	0.50	0.29	0.16	0.07	0.04

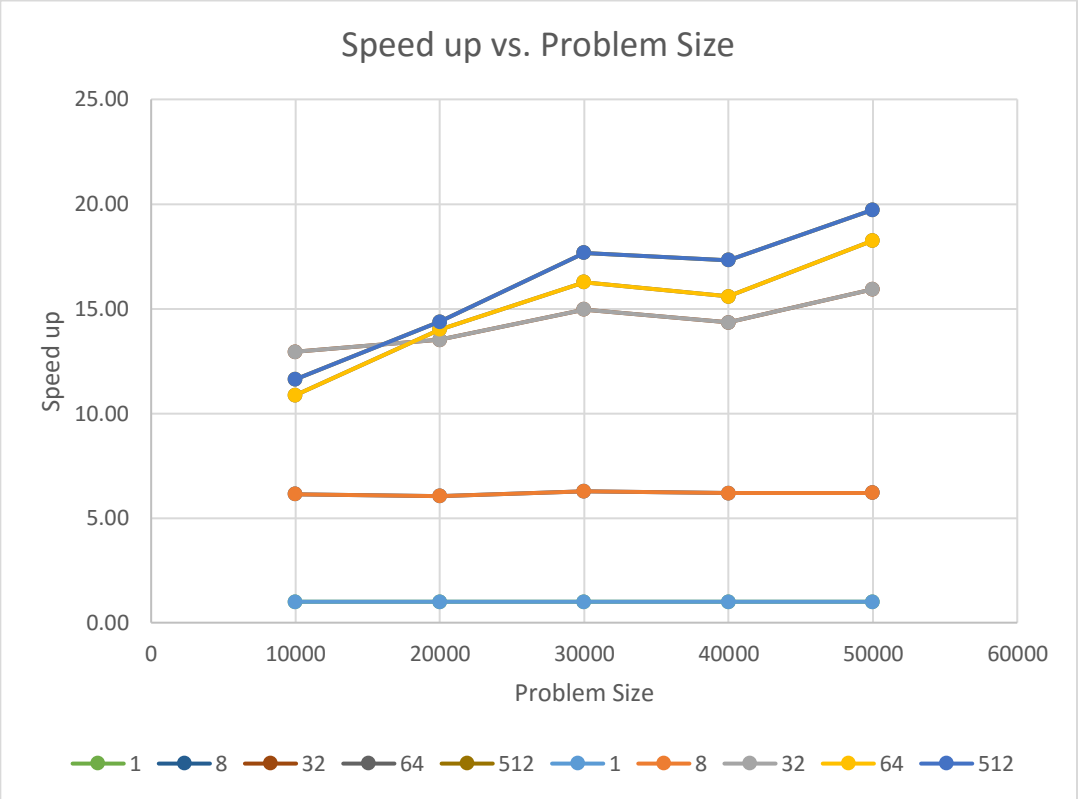
The following graphs show the performance of the program with different size and different number of threads (processes).

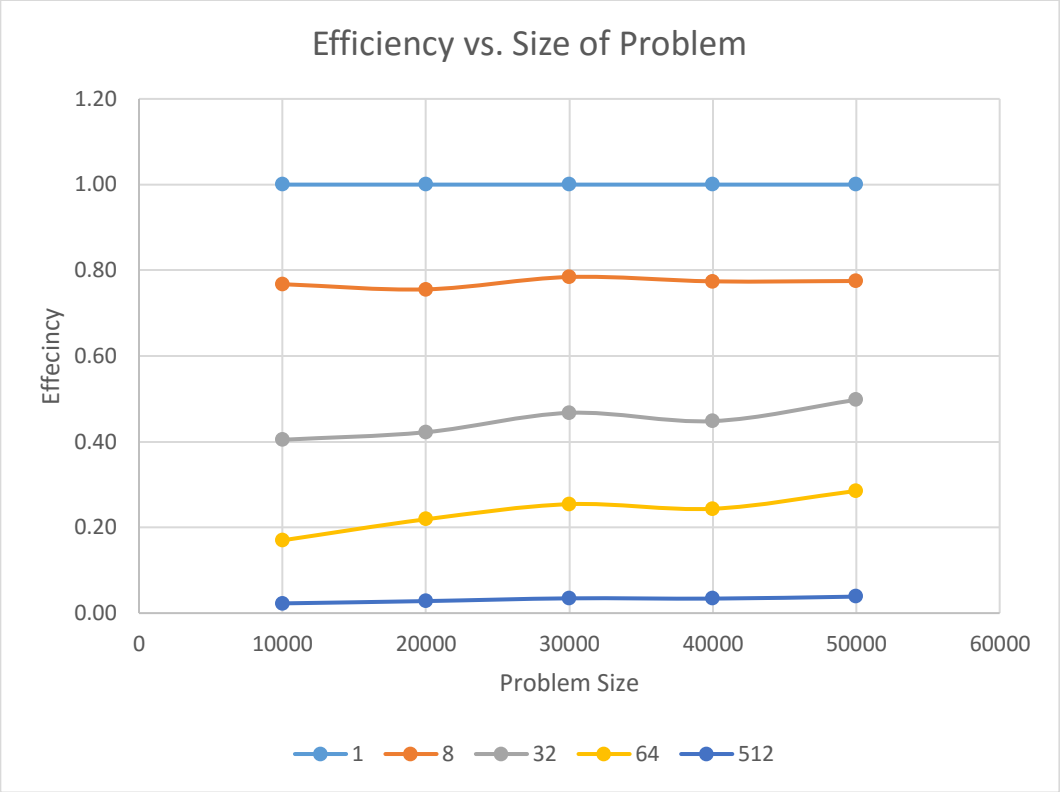



Speed up vs. No. of Processes
25.00
20.00
15.00
10.00
5.00
0.00
Speed up
1
10
100
1000
No. of Tasks
10000
20000
30000
40000
50000


Efficiency vs. No. of Processes
1.20
1.00
0.80
0.60
0.40
0.20
0.00
Effeciency
1
10
100
1000
No of Tasks
10000
20000
30000
40000
50000


Elapsed Time vs. Problem Size
16.00000
14.00000
12.00000
10.00000
8.00000
6.00000
4.00000
2.00000
0.00000
Time Elapsed (sec.)
0
10000
20000
30000
40000
50000
60000
Problem Size
1
8
32
64
512


Speed up vs. Problem Size
25.00
20.00
15.00
10.00
5.00
0.00
Speed up
0
10000
20000
30000
40000
50000
60000
Problem Size
1
8
32
64
512
1
8
32
64
512


Efficiency vs. Size of Problem
1.20
1.00
0.80
0.60
0.40
0.20
0.00
Effecincy
0
10000
20000
30000
40000
50000
60000
Problem Size
1
8
32
64
512