University of South Bohemia České Budějovice

Name	**Tushar Rewatkar**
Personal Number	**B21462**
Course	**UFY/505 – Parallel Programming and Computing**
Assignment	**Parallelization of Sorting Serial Code**

For the 100k.txt file

Number of threads	Max Wall time for Sorting	Total Job time	Load Imbalance
1 – serial	47.020954	47.370172	N.A.
1 – parallel	21.642019	21.901536	0.000000
2	3.894892	4.159084	0.001639
4	0.797169	1.121985	0.011382
8	0.184496	0.546546	0.018189
12	0.083729	0.536161	0.033189
24	0.016019	0.642201	0.060606

For the 10k.txt file

Number of threads	Max wall time for Sorting	Total Job time	Load Imbalance
1 – serial	0.292193	0.334216	N.A.
1 – parallel	0.130955	0.181261	0.000000
2	0.035179	0.117357	0.004789
4	0.008786	0.113422	0.029237
8	0.002213	0.141985	0.046912
12	0.001004	0.199355	0.089552
24	0.005618	0.393424	0.144124

Speed of parallelization

From the above table in 100k, wall time for sorting for 24 threads is 0.016019 and for 1 thread is 21.42 sec. Also when you compare with the serial version of the code which is taking 47.020954 it is much higher than all number of threads. This is because as you increase the thread number, each thread will get a subarray with a smaller length, that is why the sorting time for a higher number of threads will be lesser, i.e. with 24 threads (array divided into 24 subparts), each thread is getting almost 4k numbers for sorting while for 2 number of threads is getting 50k numbers. A hence higher number of threads takes much lesser time than a smaller number of threads and serial code. Hence, a higher number of threads is faster than a lower number of threads and serial code.

The same follows for 10k as well, time for Sorting will decrease as you increase the number of threads.

From the above Observation, we see that increasing the number of threads in parallelization reduces the wall time used for sorting. It is evident that as the number of threads increases, the speed also increases (it takes lower time for sorting hence high speed).

Load Imbalance

From the above table of 10k and 100k, you can see the load imbalance for 2 threads is much lesser than another number of threads which means the array divided into threads are much balanced (i.e. one array divided into 2 subarrays into 2 threads with each thread getting almost 50k or 5k numbers). As you increase your threads, the array will also get divided into that many parts, So as you increase the number of threads the distribution on each thread of the array will occur unevenly, hence the load imbalance will increase as you increase the number of threads.