Assignment-1 Parallel Sorting using Multi-threading

REPORT

ES19BTECH11003

Initially our aim is to find the sorted array from an unsorted array of elements using method-1 and method-2 that is described.

For the sort algorithm I used merge sort as I personally feel more comfortable in using it and it also had a very good time complexity(O(nlogn)) when compared with other sorts.

First we have to include few extra header files like math.h for pow function,pthread.h to do operations using threads

Later we first code the actual merge sort algorithm for an array of elements using the actual merge sort and after a correct execution of merge sort later we start introducing threads creation and their execution

We define an array globally in which a sequence of numbers will be stored and which needed to be sorted

We also declare two int variables globally as it helps to keep track of the start index and end index of the sub part of the array which needs to be sorted by threads

Merge sort algorithm is same for both(method-1 and method-2)

We will start the start time to calculate the time taken for execution here in the program

Later in main after extracting the input from the inp.txt file we have to create 2^p threads and assign them so that there will be 2^(n-p) elements assigned to in each thread.so by calling merge sort through each thread will sort those segments to which these threads are assigned.

After sorting done by initial set of 2^p threads these threads will be exited and after we get the original array modified into the segment wise sorted array.

From now method-1 and method-2 differs.

METHOD-1

Now in method 1 we gradually combine each segment starting from combining of (1 and 2 into one and later combining it with 3 and soon) calling the same merge sort functions by each thread and modifying the start and end values according to the threads respectively Using the for loop and altering the start and end values according to the segments that needed to be combined.

METHOD-2

Now in method2 using for loop we initially make a count of no of threads that are required to carry the complete sorting and initially we create an array of that size so that each element will be assigned as an id to the each thread created.

After completion of creating p threads now we start another loop which Is a combination of while and for in which we create remaining new threads and each thread is assigned to segments(like mix of (1,2) into seg-1,mix of (3,4) into seg-2 and soon)and again call the same merge function to follow merge sort.later with help of while loop we create

new threads and assign them to seg after combining new 1,2 segments(old 1,2,3,4 segments) and soon .

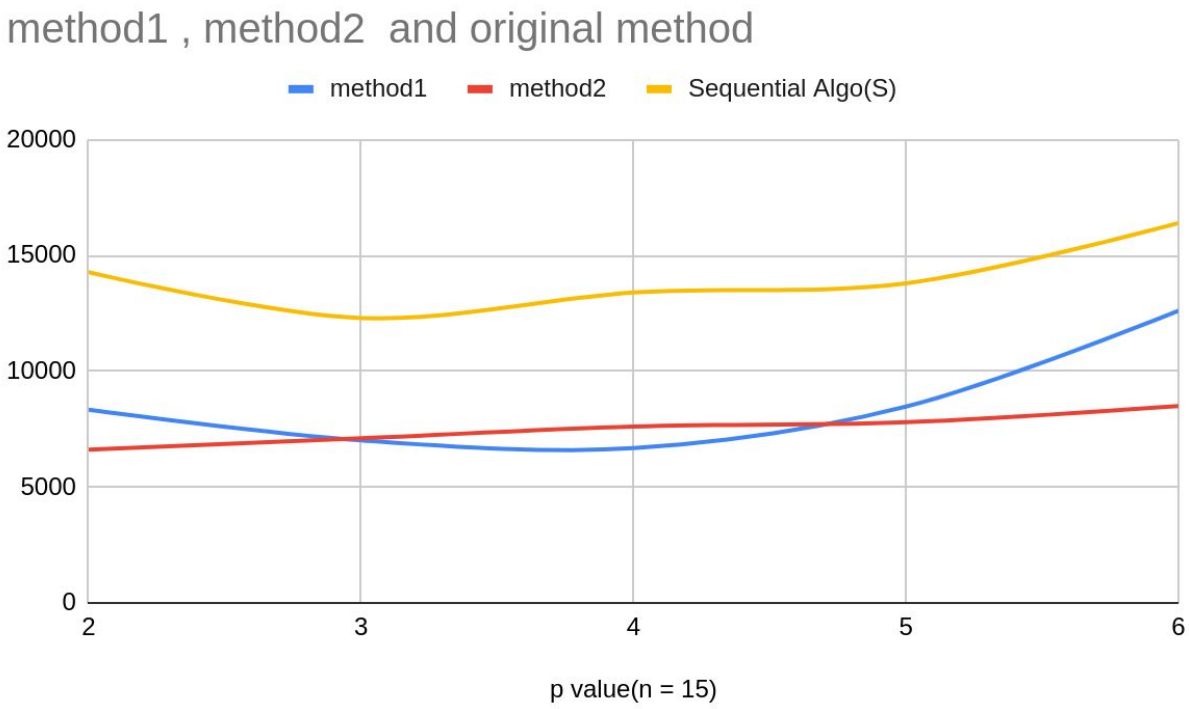
So finally we get one final thread which is assigned to add and join 1st half segment and 2nd half segment.After completion of thread activity we finally get the sorted array.

Analysis on output:
From the output of both the codes method-1 and 2 we could see that multithreading and the concept of parallelism saves a lot of time when compared to actual sequential algorithm execution time.

As we know that in comparison between method-1 and method-2 as we know method-2 includes parallelism so it saves a lot of time when compared to method-1

Analysis on Graphs:
Here the time mentioned below is in microseconds and it is the graph for which n is constant(n=15) and p value differs from(2 to 6) so time for each execution of method -1 ,method-2 were tabulated

Time	in	micro	seconds
p value(n = 15)	*method1*	method2	Sequential Algo(S)
2	*8338*	6611	14,278
3	*7016*	7105	12300
4	*6684*	7614	13400
5	*8474*	7800	13800
6	12613	8500	16400



From the above graph we can see that as p value increases time of execution slightly increases for method-1 and method-2 but for actual Sequential algo is constant as it is independent of p value.

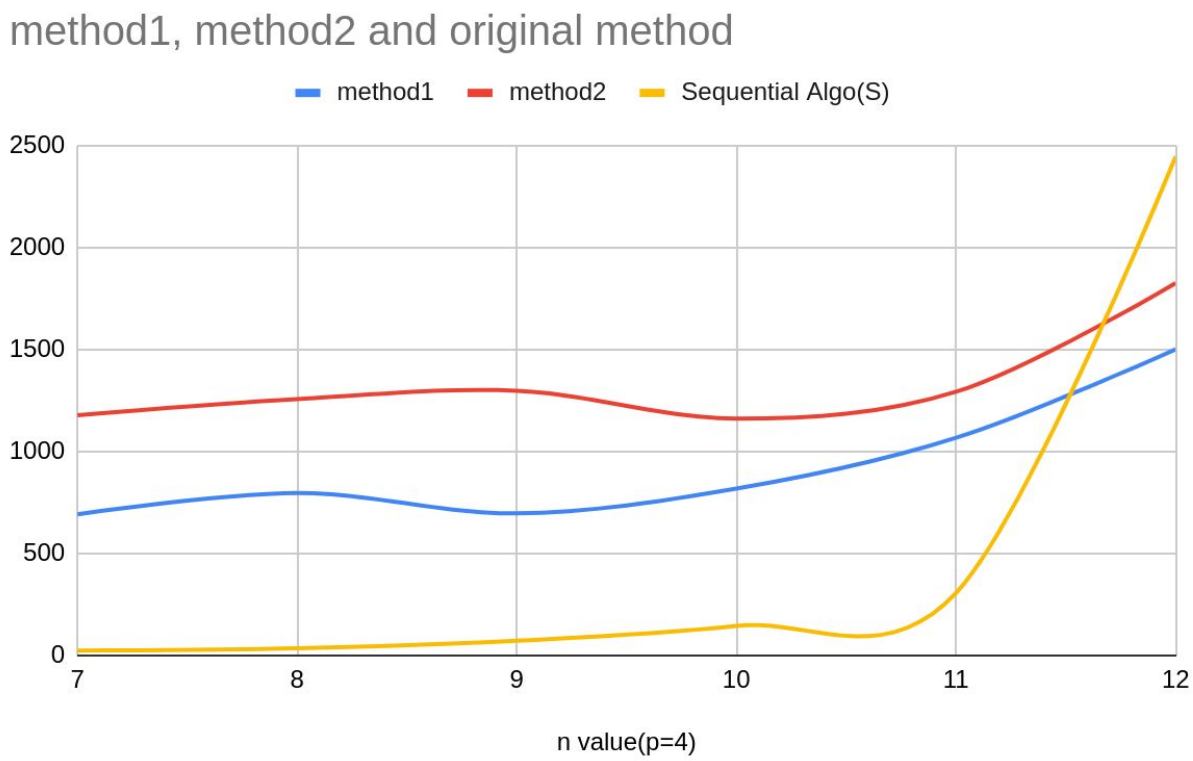
In comparison of M1 and M2 we can see that time of execution for M1 is greater than M2 in most of the cases due to parallelism that undergoes in M2 time of execution for actual sequential algo is greater than of M1 and M2

Hence order of execution is **M2<M1<sequential Algo**

Time	in	micro	seconds
n value for p=4	method1	method2	Sequential Algo(S)
7	690	1176	21
8	794	1255	32
9	694	1296	69
10	816	1159	142

11	1065	1291	303
12	1499	1824	2446

The above table contains the time of execution for constant p value(p=4) and different n values from 7-12 and as from the above table we can see that actual algo takes least time of execution as it is independent of p value.



From the above graph we can see that curves for for both method-1 and method -2 are parallel(similarly increasing) but time of execution for method-1 is smaller when compared with method-2 for smaller n values but as n value increases time of execution will be less for method-2 when compared with Method-1 as method -2 has an parallelism advantage.

At present comparison of time of execution for method-1 and method-2 and actual sequential algo is as follows

Sequential Algo<M1<M2

Conclusion:

As from the above 2 graphs and after observing the time of execution for method-1 ,method-2 and sequential method we can clearly know the advantage of parallelism to reduce the time of execution