Report
Assignment-2

Vivek Kumar(B18092)
Bhumanyu Goyal(B18012)
Harish Jaglan(B18115)

Q2 Round Robin scheduling simulation

Results –

With Number of processes = 5, assuming all processes arrive at same time and burst time = [85, 30, 35, 20, 50]

Comparison of ATT, AWT and ART for different values of time quantum -

Time Quantum	ATT	AWT	ART
1	159	115.2	2
2	158	114.8	4
5	156	112	10
10	154	110	20
20	154	110	40
40	147	103	68
80	171	127	100

With Number of processes = 10, assuming all processes arrive at same time and burst time = [85, 30, 35, 20, 50, 60, 70, 90, 80, 100]

Comparison of ATT, AWT and ART for different values of time quantum -

Time Quantum	ATT	AWT	ART
1	467.5	405.5	4.5
2	464.9	402.9	9
5	453.5	391.5	22.5
10	440.5	378.5	45
20	426.5	364.5	90
40	397	335	156.5
80	357	295	226.5

Basically, as time quantum increases RR starts behaving more like FIFO. Since FIFO isn't optimal for performance, neither is RR, so we see that some times ATT and AWT initially decreases as TQ (time quantum) increases (and thus scheduling starts behaving more like FIFO), since interleaving decreases and processes can complete relatively sooner, but in between suddenly starts increasing, since there is some long process in beginning with more priority which is taking a lot of time to execute and hence overall ATT and AWT, like in table 1 above.
Since ART takes into account just the first run of the process, ART simply decreases as TQ decreases (i.e., as scheduling starts become more "RR-istic").

Q3 Multithreaded In-place merge sort

Idea

Convert the threads to the nearest no. of threads suitable to form a perfect tree, then use the normal function of merge sort with following changes:-

a) If the level in perfect tree(mergeSort tree) is greater than or equal to last level of perfect tree use mergeSort function without any threads
b) Else form 2 threads, for [left, mid] and [mid+1, hi] mergesort
c) Use *pthread_join* to wait for both subarrays to get sorted
d) Merge the sorted subarrays in the same thread and return

Note:-

a) For measuring time of execution of mergeSort with threading use Wall clock time not CPU clock time.
b) Benefit of multithreading is observed for arrays of very large size, most likely because of overhead in thread creation

Graphs

X-axis → #threads, Y-axis → time

Fig1. Array size=1e2, thread_step= +1

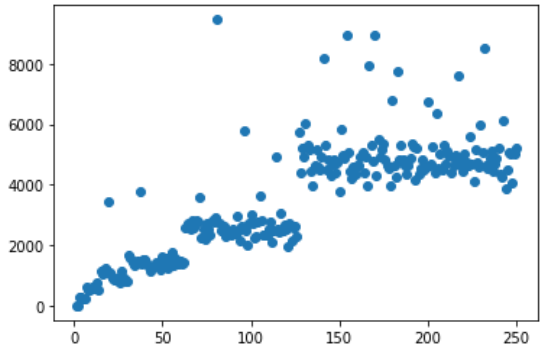


Fig2. Array size=1e4, thread_step= +1

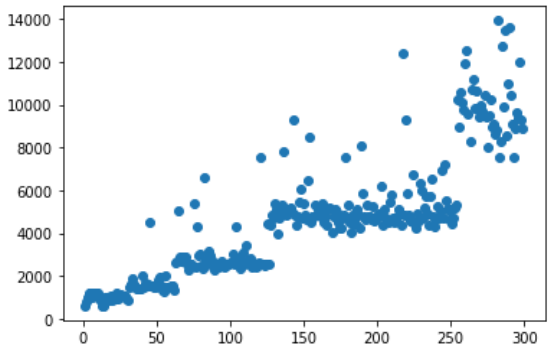


Fig3. Array size=1e7, thread_step= +1

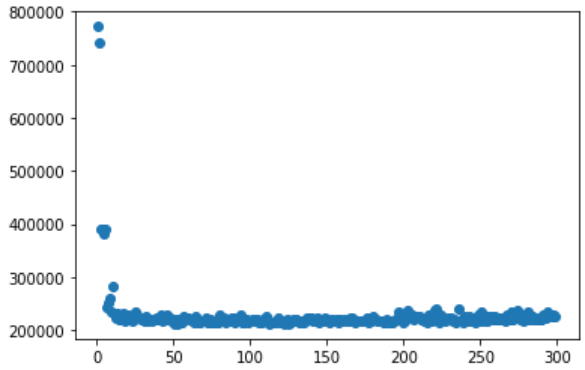
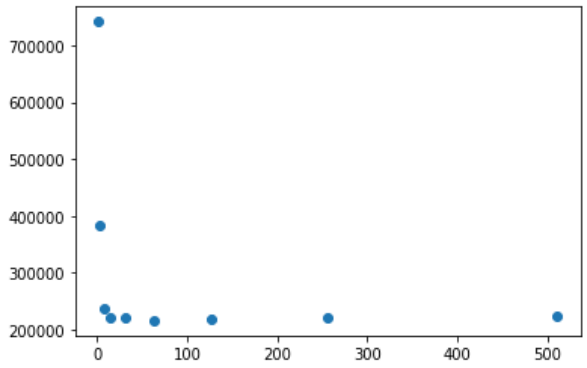


Fig4. Array size=1e7, thread_step= x2



Conclusion

As observed from graphs, multithreading decreases execution time for arrays of size 1e7(or large array) while for others multithreading has more time than 1 thread execution.