HOMEWORK #5 – Nth FIBONACCI NUMBER GENERATION

TABLE A : 1 COMPUTER

Communication Latency Optimization	Workers	Avg. time as seen by the client (Tc milliseconds)	Parallel Efficiency : T1/C.Tc
On	Multiple	7292 (T1)	1
On	Single	8022 (T1)	1
Off	Multiple	13398 (T1)	1
Off	Single	11028 (T1)	1

TABLE B : 16 COMPUTERS

Communication Latency Optimization	Workers	Avg. time as seen by the client (Tc milliseconds)	Parallel Efficiency : T1/C.Tc
On	Multiple	6048	0.0753
On	Single	7450	0.0672
Off	Multiple	35393	0.0237
Off	Single	26881	0.0256

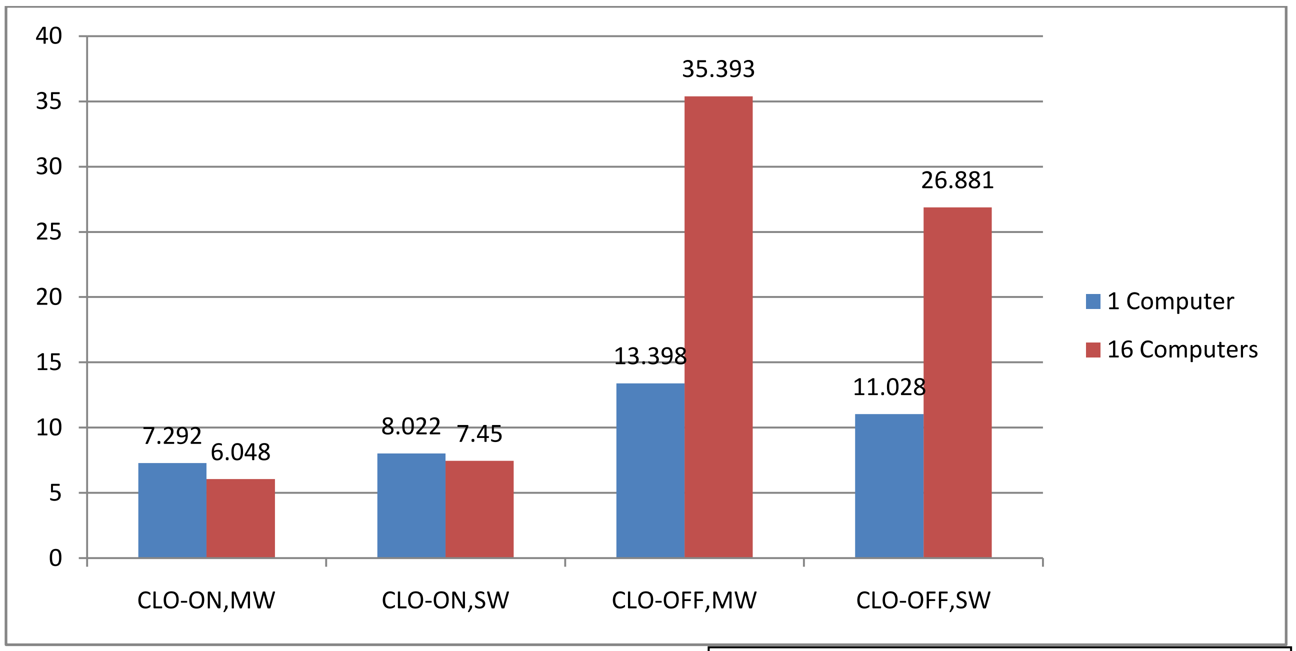
TABLE C : AVERAGE EXECUTION BASED ON LATENCY OPTIMIZATION

Communication Latency Optimization	Computers	Avg. execution time across different worker types viz.multiple/single (milliseconds)
On	1	7657
Off	1	12213
On	16	6749
Off	16	18854.5

Diff. = 4.5 s

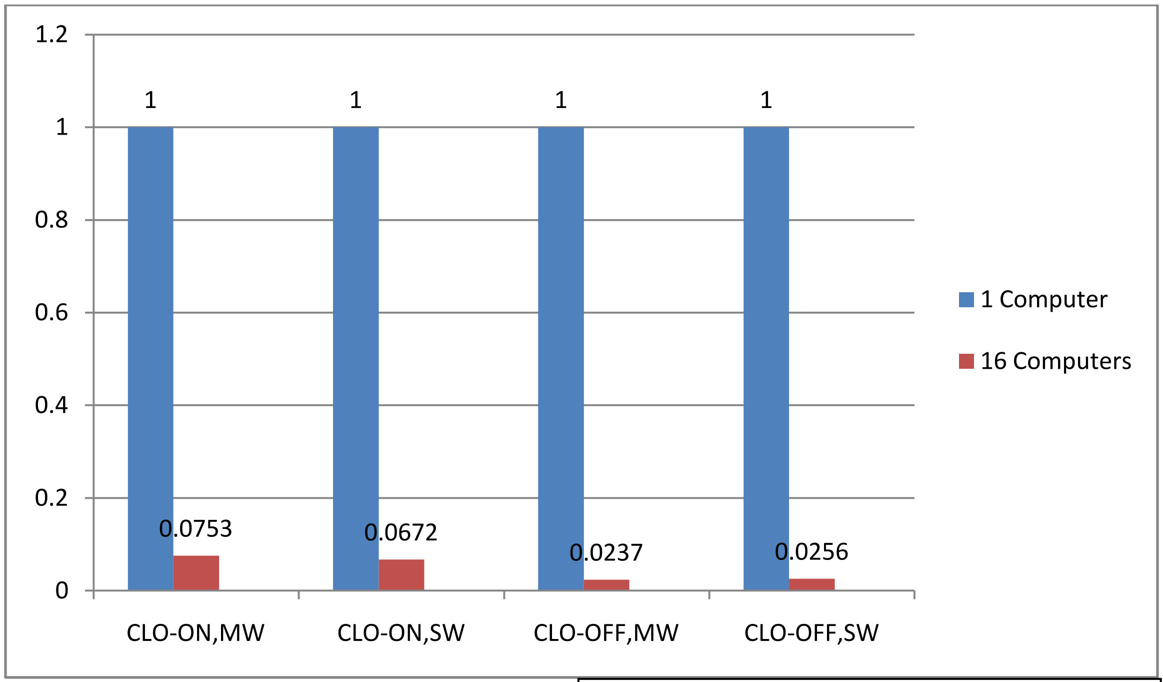
Diff. = 12.1 s

GRAPH A - EXECUTION TIMES



CLO – Communication Latency Optimization
MW – Multiple Workers
SW – Single Worker

GRAPH B - EFFICIENCY PLOT



CLO – Communication Latency Optimization
MW – Multiple Workers
SW – Single Worker

In the case : C=16, it can be seen that the parallel efficiency decreases with increase in number of parallel processors. This suggests that the degree of parallelization keeps decreasing and approaches zero as more parallel computational power is added to the infrastructure. This was an expected result. Not all tasks can achieve perfect parallelization, and this also holds good for the Fibonacci Number Generation task which we experimented with.

A noticeable abnormality in the data is the drastically increased execution times when latency optimization is switched off in C=16. This is because the massive number of subtasks generated in the recursion tree of F(18) have increased the time spent in communicating with the compute space and also idle time of the computers by a large extent.

However, a strange occurrence is the increase in parallel efficiency and decrease in execution time for the case C=16 with single worker no latency optimization. This means that the system seems to be inefficient when multiple workers operate without latency optimization. This is because the worker threads across 16 machines are not always busy executing tasks. They spend more time on synchronized calls to poll a local queue containing only a negligible number of tasks at any point of time.