Srinivas Prasad Prabhu

UT EID - sp55629

CS380P – Lab 3 – GO Tree Comparison

Platform Details

CPU - AMD EPYC 7313P 32-Core Processor
RAM - 128 GB RAM
OS - Ubuntu 22.04.3 LTS

go version

-go version go1.18.1 linux/amd64

Hashing

Hashing Questions

What kind of speedup do you expect for the two different approaches in this part? Compare and analyze them with respect to each other and to the sequential as a baseline. Explain your observations with respect to your expectations.

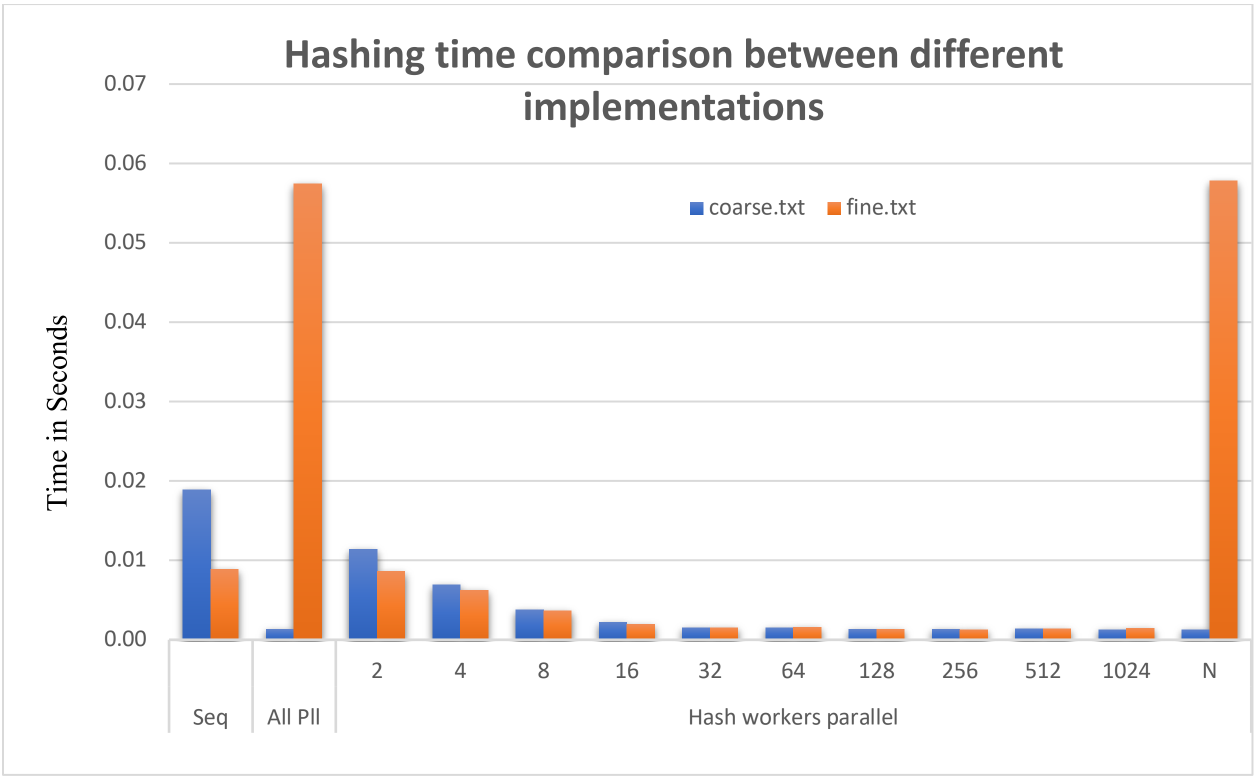
Ans -

For calculating Hashes, I have used 3 different approaches –

1) Sequential
2) Parallel –
    a. spawn a separate go routine for each tree – all parallel
    b. spawn hash-workers number of go routines and share the load to calculate the hashes of the trees.

coarse.txt		Time in Sec
Sequential		0.018910
All Parallel		0.001359
Hash workers Parallel	2	0.011432
	4	0.006986
	8	0.003836
	16	0.002268
	32	0.001579
	64	0.001566
	128	0.001357
	256	0.001411
	512	0.001438
	1024	0.001352
	N	0.001349

fine.txt		Time in Sec
Sequential		0.008892
All Parallel		0.057464
Hash workers Parallel	2	0.008647
	4	0.006294
	8	0.003694
	16	0.001963
	32	0.001567
	64	0.001633
	128	0.001342
	256	0.001353
	512	0.001423
	1024	0.001511
	N	0.057817



Observations are –

For coarse.txt input

1) Sequential is the slowest. All parallel - having a separate go routine for each tree, is the fastest.
2) As we increase the number of hash-workers we can see parallel scaling and times improving.

These observations are in line with expectations.

- Sequential is the slowest as a single go routine must traverse each tree and calculate hash sequentially.
- All parallel is the fastest as we are able to parallelize the hash calculation to as many go routines as the number of trees. However, the number of go routines is still small enough that the penalty of scheduling/switching them is still small compared to the time gain of parallel execution.
- As we increase the number of hash workers, we are able to achieve parallel scaling.

For fine.txt input

1) Sequential is slow. All parallel - having a separate go routine for each tree, is the slowest.
2) As we increase the number of hash-workers we can see parallel scaling and times improving. However, I am reaching the limits of parallel scaling around 512 hash workers. Any increase in the number of hash workers beyond this is counterproductive and starts increasing the times.

These observations are in line with expectations. There are 100K trees in fine.txt input.

- Sequential must calculate each hash sequentially. Hence consumes a lot of time.
- In all parallel - we create a 100K go routines. A lot of time is spent in context switch of these go routines. Hence all parallel ends up being the slowest of the implementations for fine.txt input.
- As we increase the number of hash workers, I can see time improvement up to 512 hash workers. Beyond this, times start increasing, indicating that the penalty for parallelizing far exceeds the gain achieved.

This shows that achieving performance improvement by parallelizing work isn't always a simple process. It is very dependent on the data, parallelizing infrastructure, algorithm used etc.

Hash Grouping

Hash Grouping Questions

What kind of speedup do you expect for the two different approaches in this part? Compare and analyze them with respect to each other and to the sequential as a baseline. Explain your observations with respect to your expectations.

Ans -

For calculating hash groups, I have used 3 different approaches –

1) Sequential – hash-workers = 1 and data-workers = 1
2) Parallel –
    a. Hash-workers > 1 and data-workers = 1
    b. Hash-workers > 1, data-workers > 1 and hash-workers=data-workers

Observations from the graphs below –

For coarse.txt

1) Sequential takes the highest amount of time.

HW > 1 and DW = 1

2) As hash-workers increase, we are seeing parallel scaling and time decreases compared to sequential. The delay here is mainly because of the contention time to process all the hashes received on a single input channel in the manager go routine.

HW > 1 and DW > 1 and HW=DW

3) With slight increase in hash workers (2), the time increases as there are multiple waits on channels and the global map lock. As the workers increase, times start to become better, and we get very good speed up. The improvements plateau around 128 workers and there is no significant improvement beyond this.

For fine.txt

1) Sequential is the fastest as all the calculations are done in a single thread and there is no locking.

HW > 1 and DW = 1

2) As hash-workers increase, we are seeing time increases compared to sequential. The delay here is mainly because of the locking contention time

to process all the hashes received on a single input channel in the manager go routine.

HW > 1 and DW > 1 and HW=DW

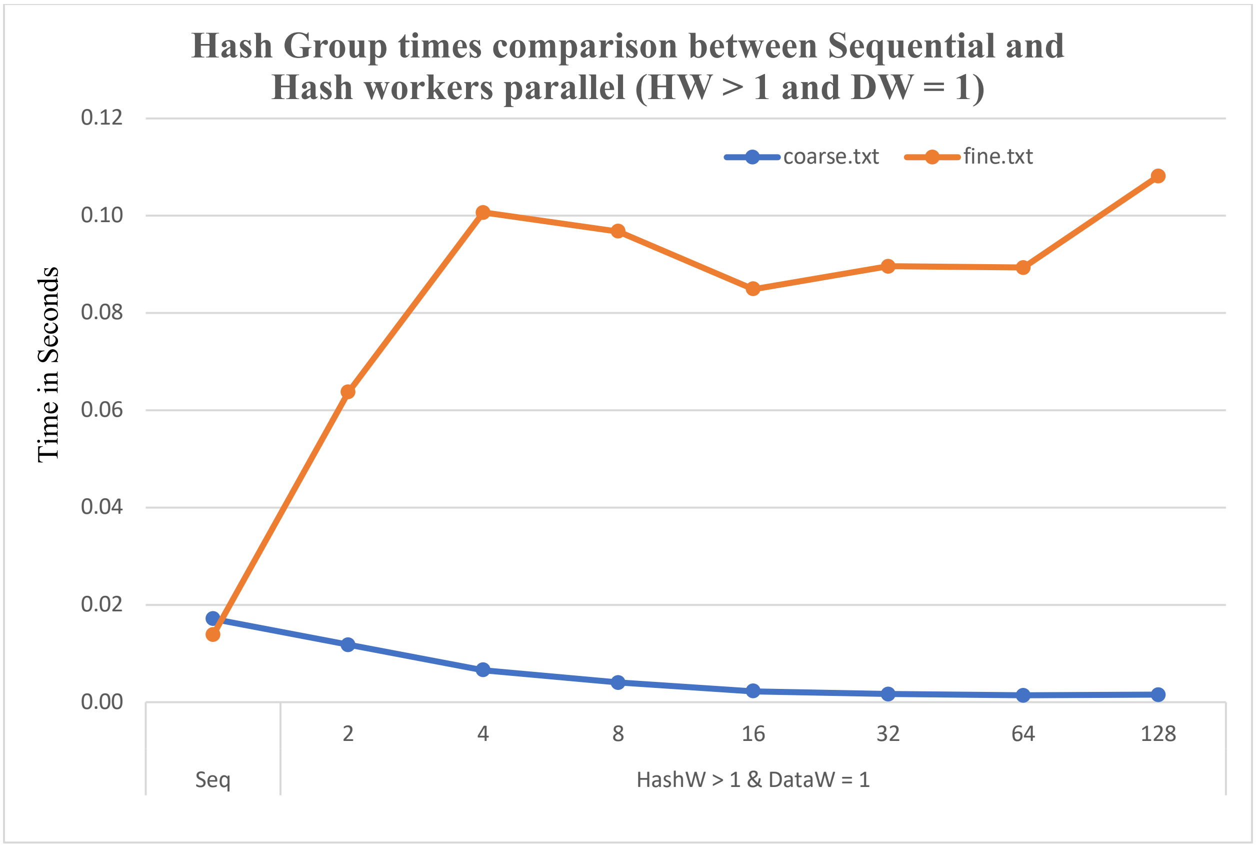
3) We see some scaling as we increase the hash workers. However, it quickly reduces, and more parallelism starts weighing down heavily on times. This is because fine.txt input has 100K trees.
   - With lower hash workers, since we are using unbuffered channels, each go routine is blocking for its hash result to be received before proceeding for the new tree.
   - With higher hash workers, along with the wait described above, most of the time is spent in switching/scheduling the go routines.
   - Lot of time is spent in locking the map in each of the data worker threads.
4) However, when comparing between case 2 and case 3 above for fine.txt input, we can clearly see times being much better in case 3. The improvement is very marginal.

These things show that parallelism doesn't always help in improving times. It also depends on the size of the input, the contention handling mechanisms used etc.

Hash Group times comparison between Sequential and Hash workers parallel (HW > 1 and DW = 1)

coarse.txt		Time in Sec
Sequential		0.017105
HashW > 1 and DataW = 1	2	0.011803
	4	0.006566
	8	0.003997
	16	0.002280
	32	0.001624
	64	0.001418
	128	0.001543

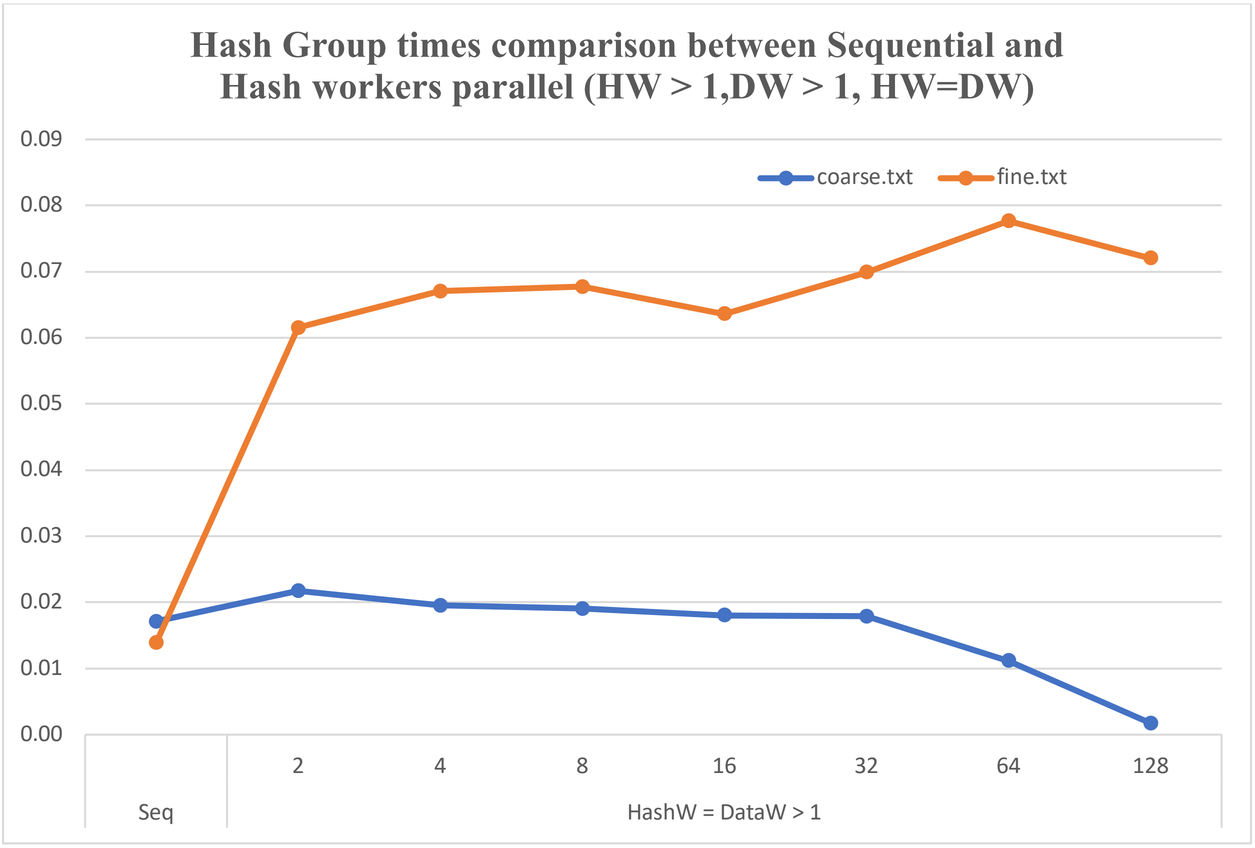
fine.txt		Time in Sec
Sequential		0.013861
HashW > 1 and DataW = 1	2	0.063743
	4	0.100625
	8	0.096750
	16	0.084927
	32	0.089591
	64	0.089341
	128	0.108133

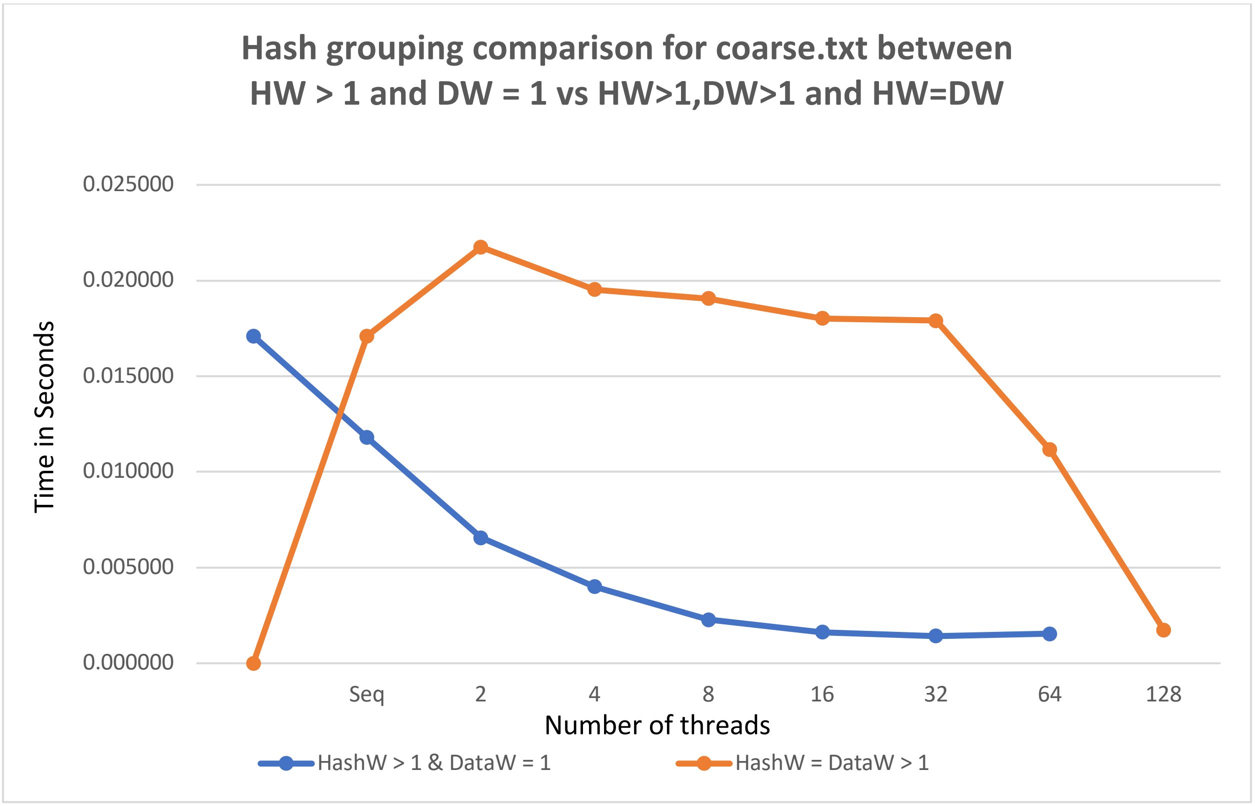


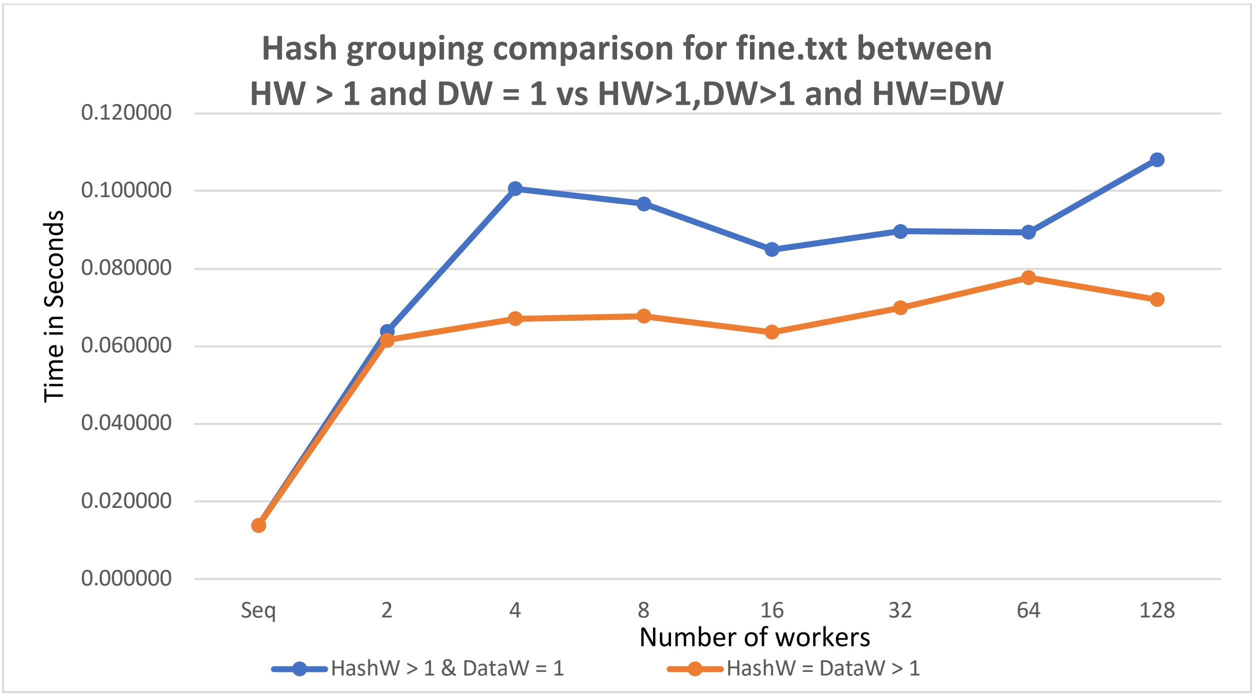
Hash Group times comparison between Sequential and Hash workers parallel (HW > 1, DW > 1, HW=DW)

coarse.txt		Time in Sec
Sequential		0.017105
HashW > 1, DataW > 1 and HashW= DataW	2	0.021745
	4	0.019520
	8	0.019054
	16	0.018018
	32	0.017900
	64	0.011163
	128	0.001735

fine.txt		Time in Sec
Sequential		0.013861
HashW > 1, DataW > 1 and HashW= DataW	2	0.061537
	4	0.067065
	8	0.067740
	16	0.063598
	32	0.069922
	64	0.077660
	128	0.072025




Hash grouping comparison for coarse.txt between
HW > 1 and DW = 1 vs HW>1,DW>1 and HW=DW
Time in Seconds
0.025000
0.020000
0.015000
0.010000
0.005000
0.000000
Seq
2
4
8
16
32
64
128
Number of threads
HashW > 1 & DataW = 1
HashW = DataW > 1


Hash grouping comparison for fine.txt between
HW > 1 and DW = 1 vs HW>1,DW>1 and HW=DW
Time in Seconds
0.120000
0.100000
0.080000
0.060000
0.040000
0.020000
0.000000
Seq
2
4
8
16
32
64
128
Number of workers
HashW > 1 & DataW = 1
HashW = DataW > 1

Compare Trees

Compare Trees Questions

What kind of speedup do you expect for the two different approaches in this part? Compare and analyze them with respect to each other and to the sequential as a baseline. Explain your observations with respect to your expectations.

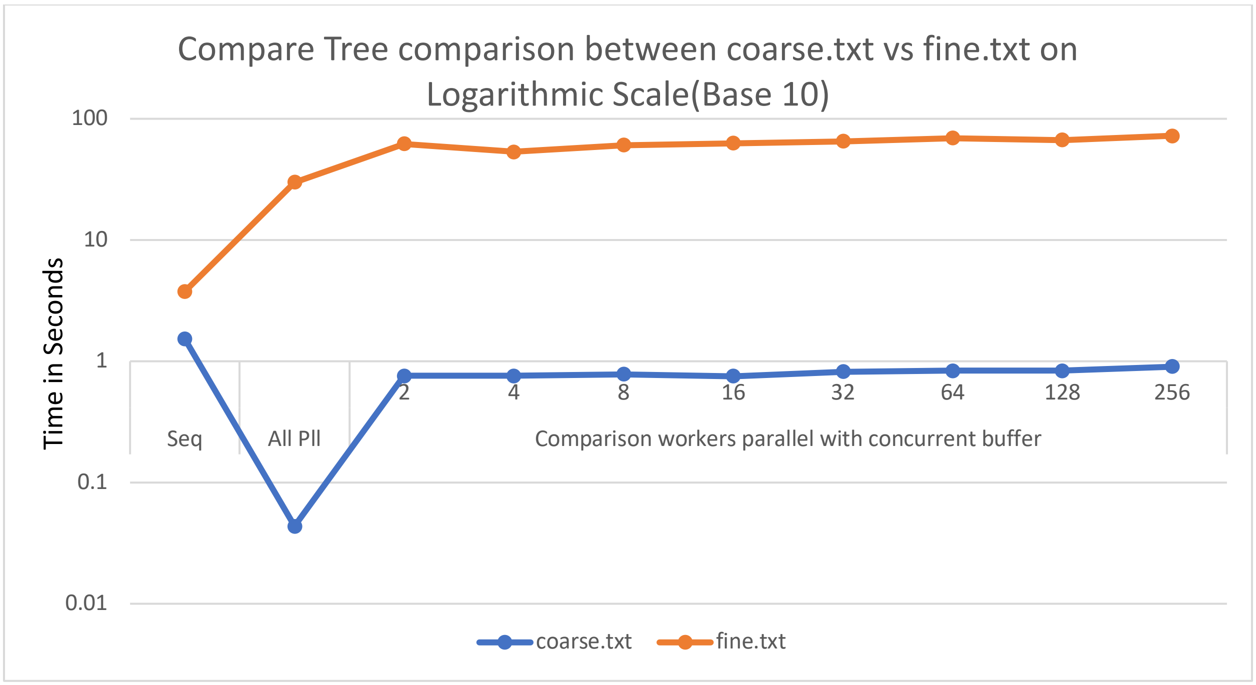
Ans -

For comparing trees, I have used 3 different approaches –

1) Sequential – comp-workers = 1
2) Parallel –
    a. comp-workers > 1, comp-workers-flag = 0 – all paralle
    b. comp-workers > 1, comp-workers-flag = 1

coarse.txt		Time in Sec
Sequential		1.5241872
All Parallel		0.0436838
CompW parallel with concurrent buffer	2	0.7598415
	4	0.7415462
	8	0.7791839
	16	0.7543274
	32	0.8199015
	64	0.8376350
	128	0.8371422
	256	0.9067645

fine.txt		Time in Sec
Sequential		3.76896
All Parallel		30.0259326
CompW parallel with concurrent buffer	2	62.033983
	4	53.443383
	8	60.725216
	16	62.835759
	32	65.356997
	64	69.113531
	128	66.753628
	256	72.474362



Observations from the graph above –

For coarse.txt

All parallel is the fastest. As the number of comparison workers in the pool increases, we can see comparison times increasing slightly. However, it is still better than the sequential time. As the number of workers increase, there is lot of overhead with respect to scheduling the various workers, signaling between the producer and workers to ensure there are jobs in the queue for workers.

For fine.txt

Sequential is the fastest as all the trees are compared in a single thread. With all parallel and multiple comparator workers in the pool, the overhead of scheduling them and the signaling between the producer and workers weighs down heaving on parallel scaling. This can be seen in the increased times with parallel go routines compared to sequential time.

Again, this indicates, that parallelism is not a one stop solution to speed up any algorithm on any input. It is very dependent on a combination of factors including input, algorithm etc.