COURSE NUMBER: CS6240
NAME: RUSHABH SHAH
Homework 1

PERFORMANCE METRICS:

1. SEQUENTIAL:

Metric	Time in seconds
Maximum Execution Time	8.42
Minimum Execution Time	4.105
Average Execution Time	4.9883

2. SEQUENTIAL WITH FIBONACCI:

Metric	Time in seconds
Maximum Execution Time	27.078
Minimum Execution Time	18.544
Average Execution Time	20.7429000000002

3. NO LOCK:

Metric	Time in seconds
Maximum Execution Time	5.599
Minimum Execution Time	1.902
Average Execution Time	2.73379999999996

4. NO LOCK WITH FIBONACCI:

Metric	Time in seconds
Maximum Execution Time	24.364
Minimum Execution Time	17.674
Average Execution Time	18.8758

5. COARSE LOCK:

Metric	Time in seconds
Maximum Execution Time	7.028
Minimum Execution Time	2.696
Average Execution Time	3.8493000000004

6. COARSE LOCK WITH FIBONACCI:

Metric	Time in seconds
Maximum Execution Time	40.058
Minimum Execution Time	30.06
Average Execution Time	34.7484000000004

7. FINE LOCK:

Metric	Time in seconds
Maximum Execution Time	7.648
Minimum Execution Time	1.965
Average Execution Time	2.999100000000003

8. FINE LOCK WITH FIBONACCI:

Metric	Time in seconds
Maximum Execution Time	23.668
Minimum Execution Time	17.669
Average Execution Time	20.321

9. NO SHARING:

Metric	Time in seconds
Maximum Execution Time	6.914
Minimum Execution Time	2.064
Average Execution Time	3.3114

10. NO SHARING WITH FIBONACCI:

Metric	Time in seconds
Maximum Execution Time	24.264
Minimum Execution Time	18.187
Average Execution Time	20.379199999997

SPEED-UPS:

Number of worker threads: 2

1. NO LOCK:
   PROGRAM B: 1.824
   PROGRAM C: 1.0989

2. COARSE LOCK:
   PROGRAM B: 1.295
   PROGRAM C: 0.597

3. FINE LOCK:
   PROGRAM B: 1.663
   PROGRAM C: 1.020

4. NO SHARING:
   PROGRAM B: 1.506
   PROGRAM C: 1.017

1. Which program version (SEQ, NO-LOCK, COARSE-LOCK, FINE-LOCK, NO-SHARING) would you normally expect to finish fastest and why? Do the experiments confirm your expectation? If not, try to explain the reasons.

Solution:

NO-LOCK runs the **fastest** as there are no locks on the threads and they can run in parallel. Although, NO-LOCK is not accurate because of the inconsistent results the program provides. The experiments confirm that NO-LOCK version runs the fastest by looking at the average time of all the programs.

2. Which program version (SEQ, NO-LOCK, COARSE-LOCK, FINE-LOCK, NO-SHARING) would you normally expect to finish slowest and why? Do the experiments confirm your expectation? If not, try to explain the reasons.

Solution:

Sequential is expected to finish the **slowest**. Other versions make use of parallelism, which helps to compute results faster. Yes, the experiments confirm the result.

3. Compare the temperature averages returned by each program version. Report if any of them is incorrect or if any of the programs crashed because of concurrent accesses.

Solution:

No-**LOCK** gives incorrect results, which maybe caused when thread tries to modify or write an outdated value in the HashMap and the program crashes too.

This maybe caused because one thread tries to read a value, which does not exist in the data structure which results in **null pointer exception**.

4. Compare the running times of SEQ and COARSE-LOCK. Try to explain why one is slower than the other. (Make sure to consider the results of both B and C—this might support or refute a possible hypothesis.)

Solution:

Sequential is **slower** than COARSE-LOCK considering without Fibonacci delay – it might be because the transfer of locks between threads might take less time.

Considering the versions with Fibonacci delay, the Coarse lock version is slower than the Sequential version as the Coarse program's threads might have to wait longer to access the data structure to write the value as the other thread is using the same data structure.

5. How does the higher computation cost in part C (additional Fibonacci computation) affect the difference between COARSE-LOCK and FINE-LOCK? Try to explain the reason.

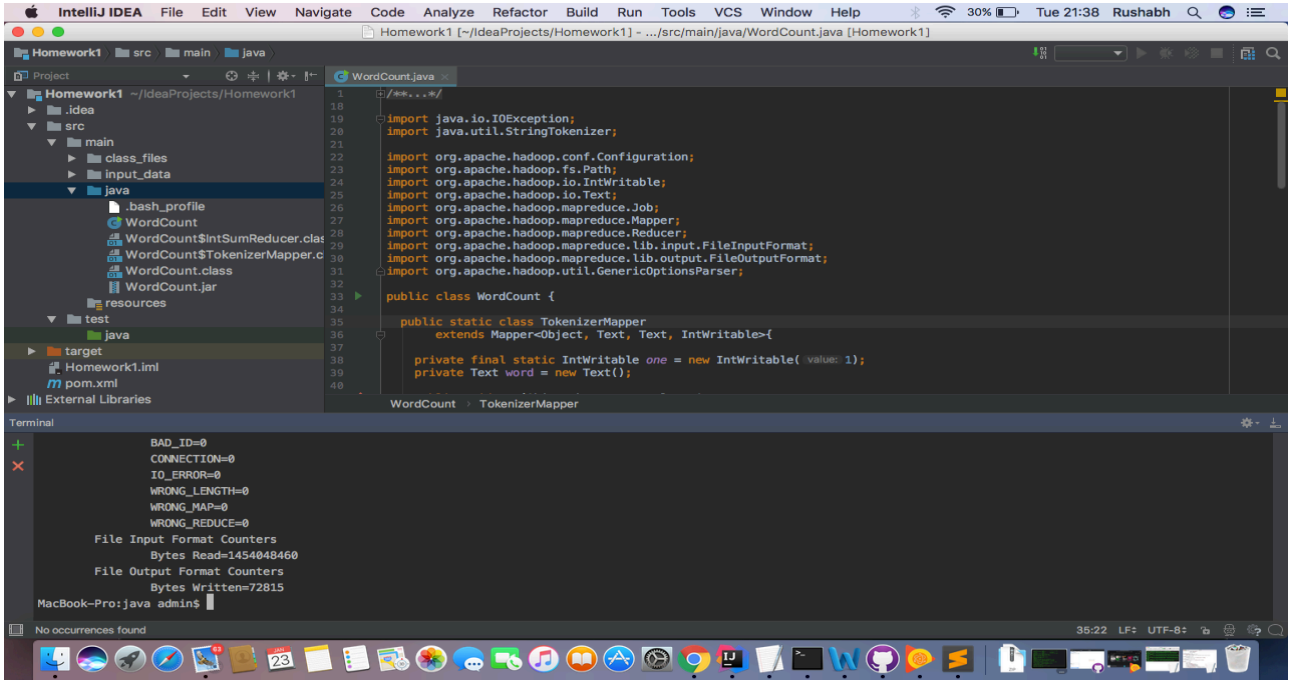
Solution

In Coarse lock, we lock the entire data structure, which increases the waiting time for processing and accessing the data structure between two threads. This is as good as Fibonacci running in series and causing a sequential delay.

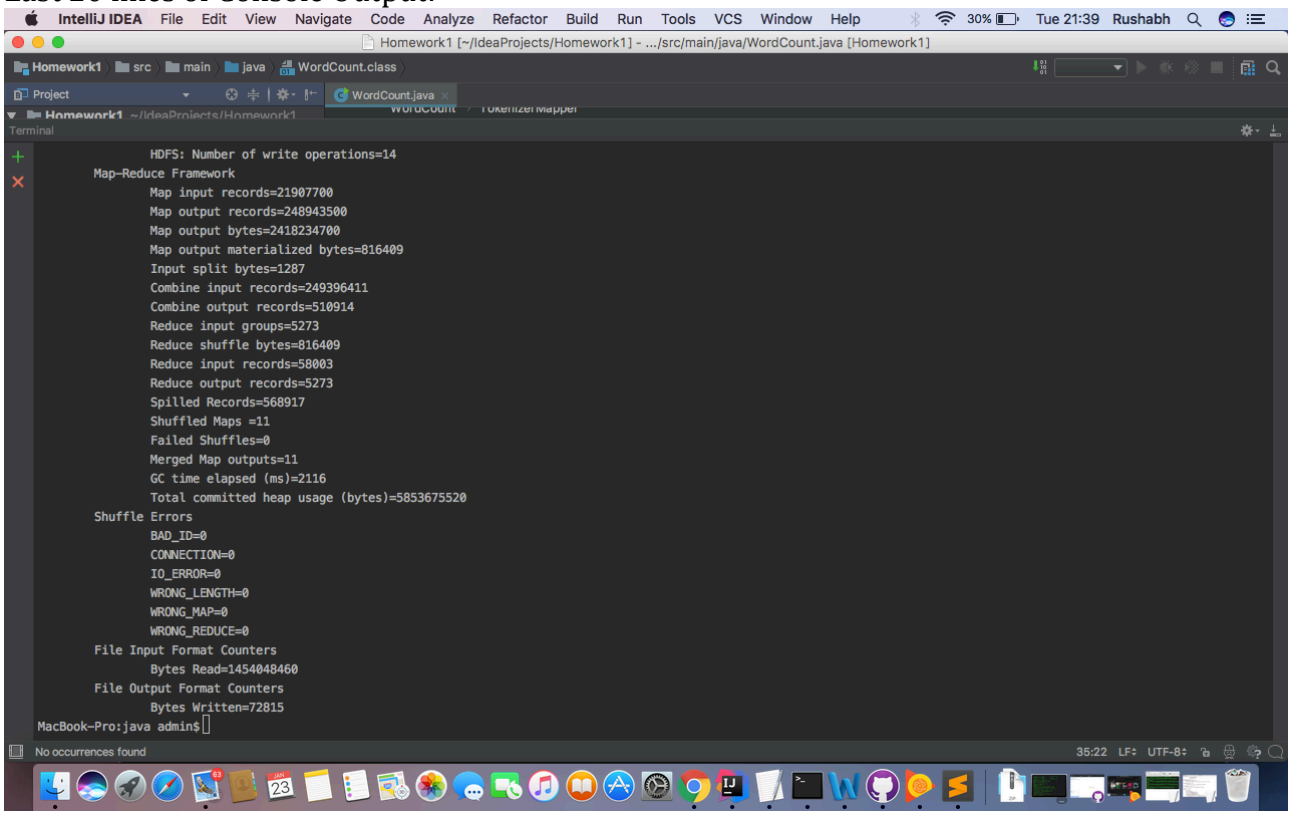
In Fine lock, the lock is applied on the attribute/variable(key) which allows certain allows Fibonacci being run in parallel.

LOCAL EXECUTION OF WORD COUNT:

Directory Structure for Local Execution of Word Count:



Last 20 lines of Console Output:



AWS EXECUTION OF WORD COUNT: