PARALLEL DATA PROCESSING USING MAP-REDUCE – ASSIGNMENT -1 (CS6240)
Prof. MIREK Riedewald
Name: VIJET BADIGANNAVAR

B.

No of worker threads = 8 (for all the experiments)

a. SEQ version (Avg. over 10 iterations)

AVG TIME	2786.5 ms
MIN TIME	2123 ms
MAX TIME	6206 ms
SPEED-UP	-

b. Multithread – NO LOCK (Avg. over 10 iterations)

AVG TIME	717.6 ms
MIN TIME	684 ms
MAX TIME	764 ms
SPEED-UP	3.88

c. Multithreaded – COARSE LOCK (Avg. over 10 iterations)

AVG TIME	827.3 ms
MIN TIME	813 ms
MAX TIME	854 ms
SPEED-UP	3.36

d. Multithreaded – FINE LOCK (Avg. over 10 iterations)

AVG TIME	748 ms
MIN TIME	727 ms
MAX TIME	783 ms
SPEED-UP	3.72

e. Multithreaded – SEPARATE LOCK PER THREAD (Avg. over 10 iterations)

AVG TIME	762.6 ms
MIN TIME	745 ms
MAX TIME	804 ms
SPEED-UP	3.65

C.

No of worker threads = 8 + Fibonacci (17) delay - When temperature is added to running sum. (for all the experiments)

a. **SEQ version (Avg. over 10 iterations)**

AVG TIME	2886.6 ms
MIN TIME	2191 ms
MAX TIME	6533 ms
SPEED-UP	-

b. Multithread – NO LOCK (Avg. over 10 iterations)

AVG TIME	765.4 ms
MIN TIME	730 ms
MAX TIME	818 ms
SPEED-UP	3.77

c. Multithreaded – COARSE LOCK (Avg. over 10 iterations)

AVG TIME	887.8 ms
MIN TIME	855 ms
MAX TIME	930 ms
SPEED-UP	3.25

d. **Multithreaded – FINE LOCK (Avg. over 10 iterations)**

AVG TIME	794.2 ms
MIN TIME	704 ms
MAX TIME	859 ms
SPEED-UP	3.63

e. **Multithreaded – SEPARATE LOCK PER THREAD (Avg. over 10 iterations)**

AVG TIME	814.3 ms
MIN TIME	774 ms
MAX TIME	870 ms
SPEED-UP	3.54

QUESTIONS:

a. **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.**

   NO-LOCK version should be fastest. As the input is spread across multiple threads and there is no locking as such, each of the thread process the input separately thus increasing the efficiency of the program (though the results are not consistent). In SEQ version there is single thread thus its execution time will be greater. In COARSE-LOCK, FINE-LOCK there is some sought of locks, thus is some kind of delay in terms of execution. NO-SHARING, there is some time spent while collaborating the results from the individual threads thus this version might eat up some time. Thus, NO-LOCK version is expected to finish fast.

   My experiments do confirm that **NO-LOCK version** has the lowest average time and finishes fast when compared to other program versions.

b. **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.**

   SEQ version of the program should be the finish slowest as there is no parallelism and all the records needs to be processed one at a time. As the number of records are more, using threads will definitely boost the running time.

   My experiments do confirm that **SEQ version** has the slowest finishing time when compared to other program versions.

c. **Compare the temperature averages returned by each program version. Report if any of them is incorrect.**

   The avg. temperatures of all the versions matches with the SEQ version, except the NO-LOCK version.
   Some of the StationIds that mismatch in the NO-LOCK version are:
   USC00030936 USW00014828 USC00188405 USC00048218 USC00256135
   USC00476838 USW00014845 USC00048210 USC00049540 USC00362108

USC00109658 USW00014840 USC00388714 USC00176430 USC00417678 CA001163779 USC00036248 USC00458190

d. **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.)**

Running time of COARSE-LOCK version is better than SEQ version. This is expected because, in SEQ version, the instructions are executed one at a time. For one operation say fetching a record from the list, 3-4 operation would be happening in the memory like splitting the string by " " (1),then checking the if it's a TEMP type(2) and then loading into the register (3), adding the count(4), saving it back to memory(5). All these operations happen sequentially.

With the COARSE-LOCK approach, there are 8 threads that are running in parallel. Thus, 8 records can be fetched at a time and (1) and (2) operations can be performed within the thread itself. But (3), (4), (5) needs to be done within the shared memory using locks to maintain consistency.

e. **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.**

In case of Course lock, the lock is associated on the whole data structure thus, if one thread has a lock on it, then all the other threads have to wait. Moreover, if the thread increments the count, then it adds the additional delay(Fibonacci) as well that will affect all the waiting threads.
In Fine-lock version, the lock is on a particular object associated with the stationId, but not on the data structure as a whole. Thus, thread t1 wants to update stn1 and thread t2 wants to update stn 2 they can do it simultaneously. But, if the thread t3 wants to obtain the lock on stn2 then it has to wait until t2 release it. In the former case all the threads need to be waited but in this case only t3 will be waiting until it receives the lock. Thus the delay is applicable to this waiting thread not for other threads. Thus, Fine-lock version of the program has better running time when compared to that of the coarse lock.