Project report for

Project 2: MULTI-THREADED COLLATZ STOPPING TIME GENERATOR

Course: COP 5990

CS Foundations: OS & Networks

Student: Probal Chandra Dhar

**Introduction:** In this project, we have created some threads and calculate the collatz sequence of the numbers given from commandline argument. The number of threads to create also given from commandline argument of the program. The program should use mutex\_lock to update the global COUNTER variable. Though a –nolock argument is optional to address the race condition when two threads access the COUNTER variable in the same time. These things are done in the problem given in the .zip file.

**Race condition:** Race condition happens when two threads are access the same memory or try to change the same location of a memory in the same time. In mt-collatz program this problem can be seen if anyone run the program with –nolock optional argument in the commandline. Using –nolock the program don’t use the mutex\_lock when increasing the global COUNTER variable and also when writing the stopping time to the histogram array. What mutex\_lock does is that it’s do the operation within the lock mutually exclusively so that not more than one thread can do the process that is within the mutex\_lock.

**Result:** In this report, I showed some results of time required for some values of N & T and if I used –nolock option or not in the table below. I calculated the required time by running the program with the same values of N & T 15 times and then took the average value of those time. The time required to complete the task is dependent on the other programs running the machine. I tested my program in the UWF CS servers measured the time from there. The time measurement was done with file writing for both stdout and stderr.

A graph of the collatz-sequence stopping time is given below for the value of 200000 as the maximum number and with 25 threads.

The time measurement graph is also given below. In that graph the number of N is fixed and the number of T varies in the x-axis. In this graph, we can see that with the increasing number of threads the time required is getting higher for some problem. So, the performance if decreasing. I addressed the problem in the README.docx file. That file is also attached with the project. But in general, the time required to complete the task using more threads should be less than using less threads to complete the task and the performance should increase.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | T | | Time required  (average) s | | | -nolock | | comment | | Output |
| 100 | 8 | | .005073 | | | No | |  | | stdout |
| 1000 | 10 | | .006586 | | | No | |  | | Stdout |
| 1000 | 2 | | .006843333 | | | No | |  | | Stdout |
| 1000 | 2 | | .003164667 | | | No | |  | | File using > |
| 1000 | 2 | | .005628333 | | | Yes | |  | | stdout |
| 10000 | 12 | | 0.026789 | | | no | |  | | stdout |
| 10000 | 12 | | 0.010398 | | | yes | |  | | stdout |
| 200000 | 5 | | 0.194082 | | | no | |  | | stdout |
| 200000 | 2 | | 0.1628688 | | | No | |  | | File using > |
| 200000 | | 3 | | 0.171777867 | No | |  | | File using > | |
| 200000 | 4 | | 0.1682306 | | | no | |  | | File using > |
| 200000 | 5 | | 0.1782224 | | | no | |  | | File using > |
| 200000 | 6 | | 0.180440733 | | | No | |  | | File using > |
| 200000 | 7 | | 0.178814867 | | | No | |  | | File using > |
| 200000 | 8 | | 0.180485133 | | | No | |  | | File using > |
| 200000 | 9 | | 0.1761264 | | | No | |  | | File using > |
| 200000 | 10 | | 0.176296533 | | | No | |  | | File using > |
| 200000 | 11 | | 0.1654732 | | | No | |  | | File using > |
| 200000 | 12 | | 0.179530933 | | | No | |  | | File using > |
| 200000 | 13 | | 0.179328133 | | | No | |  | | File using > |
| 200000 | 14 | | 0.178116333 | | | No | |  | | File using > |
| 200000 | 15 | | 0.175129667 | | | No | |  | | File using > |
| 200000 | 16 | | 0.180814133 | | | No | |  | | File using > |
| 200000 | 17 | | 0.178189533 | | | No | |  | | File using > |
| 200000 | 18 | | 0.1791328 | | | No | |  | | File using > |
| 200000 | 19 | | 0.1802426 | | | No | |  | | File using > |
| 200000 | 20 | | 0.178290067 | | | No | |  | | File using > |
| 200000 | 21 | | 0.181943467 | | | No | |  | | File using > |
| 200000 | 22 | | 0.177300267 | | | No | |  | | File using > |
| 200000 | 23 | | 0.17319333 | | | No | |  | | File using > |
| 200000 | 24 | | 0.1776404 | | | No | |  | | File using > |
| 200000 | 25 | | 0.1749142 | | | No | |  | | File using > |
| 200000 | 26 | | 0.174191867 | | | No | |  | | File using > |
| 200000 | 27 | | 0.1770456 | | | No | |  | | File using > |
| 200000 | 28 | | 0.1794522 | | | No | |  | | File using > |
| 200000 | 29 | | 0.180705933 | | | No | |  | | File using > |
| 200000 | 30 | | 0.178621533 | | | No | |  | | File using > |

Time measurement graph. This graph is calculated with the value of N 200000.

Now the table below shows some of the time required to calculate the collatz sequence using –nolock so if we use thread the calculation should create some race condition.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | T | | Time required  (average) s | | | -nolock | | comment | | Output |
| 200000 | 2 | | 0.2806815 | | | Yes | |  | | File using > |
| 200000 | | 3 | | 0.269641667 | Yes | |  | | File using > | |
| 200000 | 4 | | 0.301185667 | | | Yes | |  | | File using > |
| 200000 | 5 | | 0.301539067 | | | Yes | |  | | File using > |
| 200000 | 6 | | 0.3025112 | | | Yes | |  | | File using > |
| 200000 | 7 | | 0.304119067 | | | Yes | |  | | File using > |
| 200000 | 8 | | 0.297914867 | | | Yes | |  | | File using > |
| 200000 | 9 | | 0.298218533 | | | Yes | |  | | File using > |
| 200000 | 10 | | 0.300838667 | | | Yes | |  | | File using > |
| 200000 | 11 | | 0.2972212 | | | Yes | |  | | File using > |
| 200000 | 12 | | 0.294405333 | | | Yes | |  | | File using > |
| 200000 | 13 | | 0.3015412 | | | Yes | |  | | File using > |
| 200000 | 14 | | 0.2988758 | | | Yes | |  | | File using > |
| 200000 | 15 | | 0.306145267 | | | Yes | |  | | File using > |
| 200000 | 16 | | 0.301616733 | | | Yes | |  | | File using > |
| 200000 | 17 | | 0.3089822667 | | | Yes | |  | | File using > |
| 200000 | 18 | | 0.312657533 | | | Yes | |  | | File using > |
| 200000 | 19 | | 0.315805467 | | | Yes | |  | | File using > |
| 200000 | 20 | | 0.308564133 | | | Yes | |  | | File using > |
| 200000 | 21 | | 0.320376867 | | | Yes | |  | | File using > |

This is the graph of collatz sequence. I used 200000 as maximum number and I used 25 threads to calculate the stopping time of the numbers for this graph

**Conclusion:** calculating collatz-sequence for a number is no big deal for modern computers. That’s why in this project the sequence is calculated for a big range of number and in this range, all the sequence for all the number has been calculated. This is done to get the machine bust for some certain program. To calculate the collatz-sequence for a range of large number like 100000. In this program the collatz-sequence of all number from 2 to a given number for example 100000 has been calculated. Multi-threading has been used to do this job so that the time measurement can be seen with the increase number of threads. Also, a graph of the stopping time for a collatz-sequence can be seen in above graph. When using multi-threading race condition also can be seen when lock is not using during access shared memory. The output of race condition should differ from one to another. This can be seen if output of the same program ( with the same number of N & T ) can be written in different files and check the difference between them.