Parallel and Distributed Systems Assignement 1

Sayak Chakraborti NetID-schakr11

February 2017

Compilation and Running

Compile using the make file.Type in make.If there are errors,then for explicit compilation use "g++ parcount.cpp -std=c++0x -pthread -w -o parcount" or "g++ parcount -std=c++11 -pthread -w -o parcount" whichever works on your compiler.
For running the code use "./parcount -t 5 -i 10000" or something similar with variable parameters.Extensive error checking has not been done for the code.

Introduction

The purpose of this assignment is to get familiar with c++11 threads and use them with different synchronous mechanism to increment a counter ,iteration number of times.The number of threads and the number of iterations are provided as an input .If not then the number of threads is taken as 4 and the number of iterations is taken as 10,000.
Once my program deciphers the information about the number of threads and the number of iterations,it declares an array of numthreads threads myThreads.For each experiment the program calls these threads with the specific function for that experiment.I have also introduced a global atomic Boolean variable start which initialized to false before the start of each experiment. So the process of launching the threads with appropriate functions was done using a loop.In order for the threads to start running at the same time they spin wait on the atomic Boolean variable start which is set to true once all the concerned threads are launched.I have also used the chrono::high_resolution_clock to get the time spent on performing the increment for each of the experiments.
So the experiments are as follows,for the first experiment we increment the counter without any kind of synchronisation. This cause no access control to the shared counter which may be updated by different threads at the same time with same previous value and thus result in overlapping updates of the shared counter giving an erroneous result.
The second experiment is implemented using mutex lock and unlock operations.The access to the shared counter is restricted by lock acquired on the

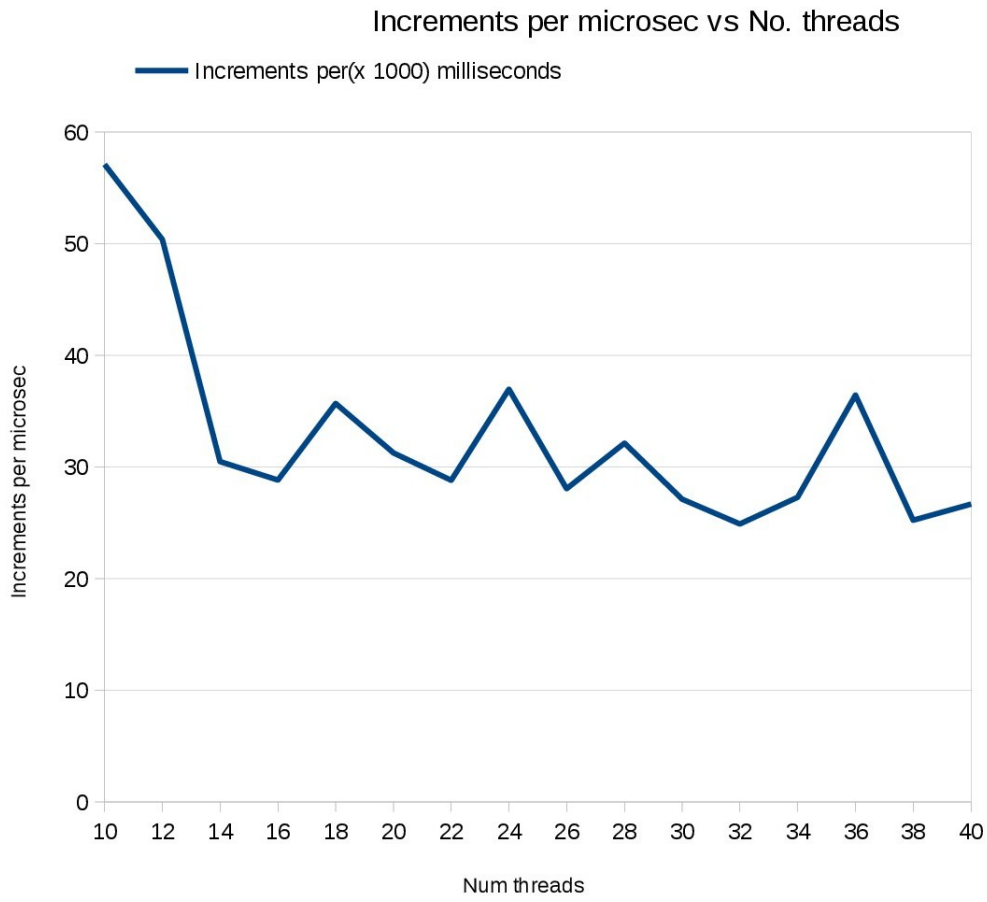
mutex variable.Thus each thread which wants to increment/update the counter has to acquire the lock on the mutex do the update and release the mutex.This assures that the updates are not overlapped and happen one at a time.
The third experiment uses lockguard for synchronization ,as far as I know lockguards behave the same way as locks ,similar to acquiring a lock on a mutex variable.When the scope of the lockguard is exited the lock on the mutex is automatically released and thus it gives us consistent value with the output as the lock unlock mechanism.
For the fourth experiment,I have declared an atomic counter and used the atomic fectch and add instruction to update the counter.Each update by the counter is atomic and thus assures synchrony ,also the variable in question is atomic and can be modified/accessed by a single thread at a time.We do get consistent result from this experiment.
For the fifth experiment I declared thread specific counter which is initialised to 0 when the threads starts to run.At the end of each thread's execution, the local counters are added together to form the overall value.
So for each experiment,the total increment should be t*i increments.

Experimental Results

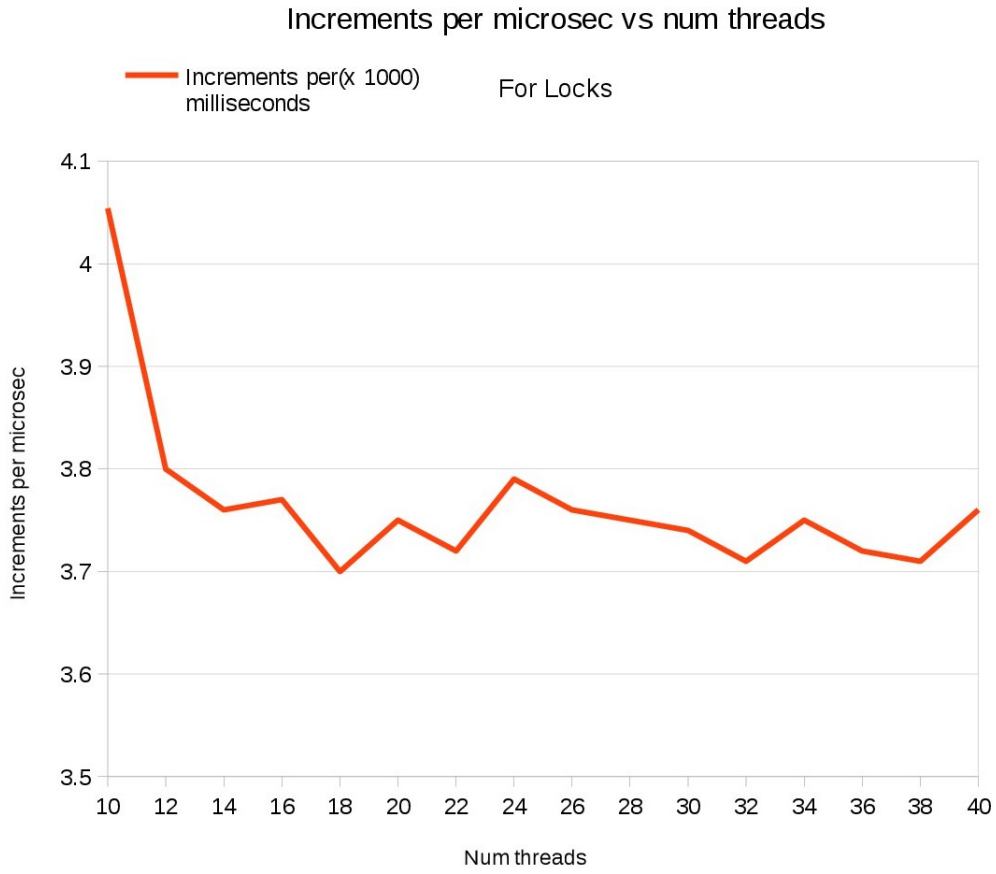
The first experiment was performed on the cycle 3 machine from 2 am in the morning to 8 am and the expected traffic was not more than 3 people using the server at that point in time.The specification of this server is that it has 24 nodes and works at 2.20 GHz clock speed.
To ascertain that the results produced are consistent,I have run at least 4 time with the same configuration(that is same number of threads and iterations).If in these four runs they have provided at least consistent values for the example time difference of maximum of 0.3 seconds is permissible.The following results are from running the experiments with different number of threads with the increment value being constant to 9,00,000.We see that the no synchronisation case gives us erroneous result but the others are okay.

height2*Number of threads	Counter value					Increments per 1000 milliseconds				
	No sync	Lock	Lockguard	Atomic	Local	No sync	Lock	Lockguard	Atomic	Local
10	57.08	3101070	9000000	9000000	9000000	9000000	4.05	3.71	38.6	264.7
12	2418143	10800000	10800000	10800000	10800000	50.37	3.8	3.6	34.8	245.45
14	1709275	12800000	12800000	12800000	12800000	30.48	3.76	3.5	38.76	297.87
16	2190894	14400000	14400000	14400000	14400000	28.82	3.77	3.53	37.78	320
18	2855253	16200000	16200000	16200000	16200000	35.69	3.7	3.52	37.6	225
20	2982265	18000000	18000000	18000000	18000000	31.26	3.75	3.42	35.29	276.92
22	3002534	19800000	19800000	19800000	19800000	28.81	3.72	3.48	39.42	208.42
24	4286811	21600000	21600000	21600000	21600000	36.95	3.79	3.4	40	213.86
26	3365558	23400000	23400000	23400000	23400000	28.04	3.76	3.45	38.36	316.21
28	4175807	25200000	25200000	25200000	25200000	32.12	3.75	3.46	38.18	301.79
30	4067832	27000000	27000000	27000000	27000000	27.11	3.74	3.43	38.57	182.43
32	4040200	28800000	28800000	28800000	28800000	24.89	3.71	3.49	37.89	176.68
34	4731597	30600000	30600000	30600000	30600000	27.28	3.75	3.46	39.74	204
36	6195604	32400000	32400000	32400000	32400000	36.44	3.72	3.51	32.4	221.91
38	4943216	34200000	34200000	34200000	34200000	25.22	3.71	3.42	32.57	340.16

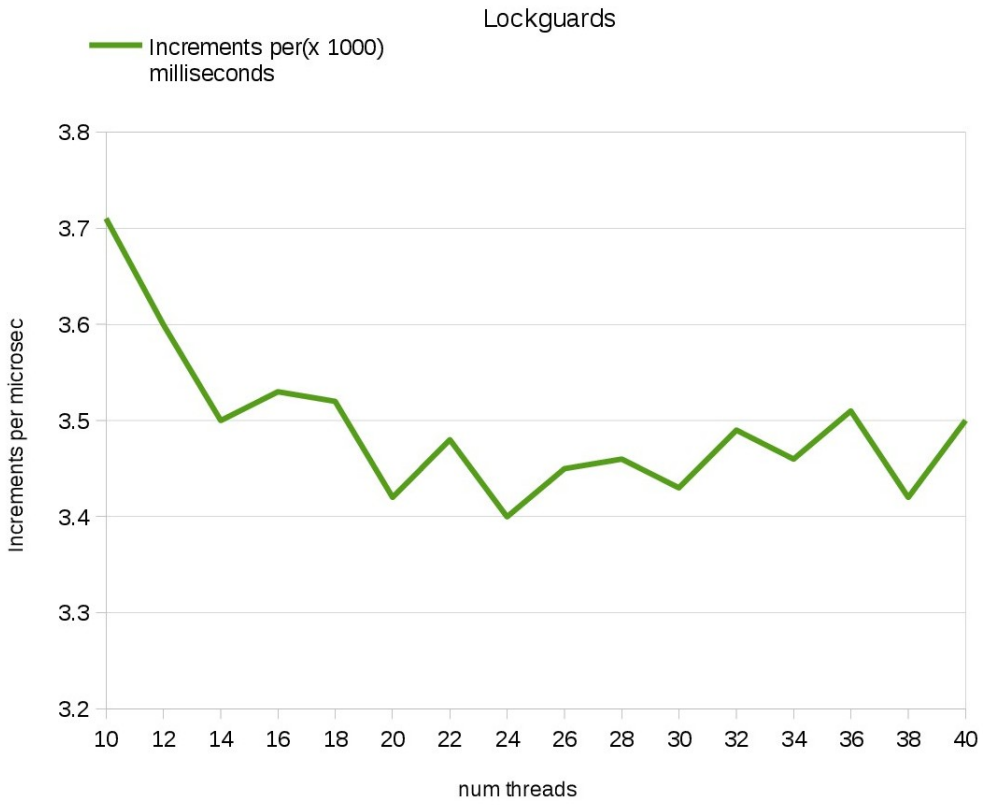
Now we have a graph for the increments per 1000 milliseconds for the no synchronisation case,we observe that this value on average between the 30-40 x 10^3 range maybe mean of 35x10^3 increments per millisecond and the values are not very consistent.
Case:No synchronisation



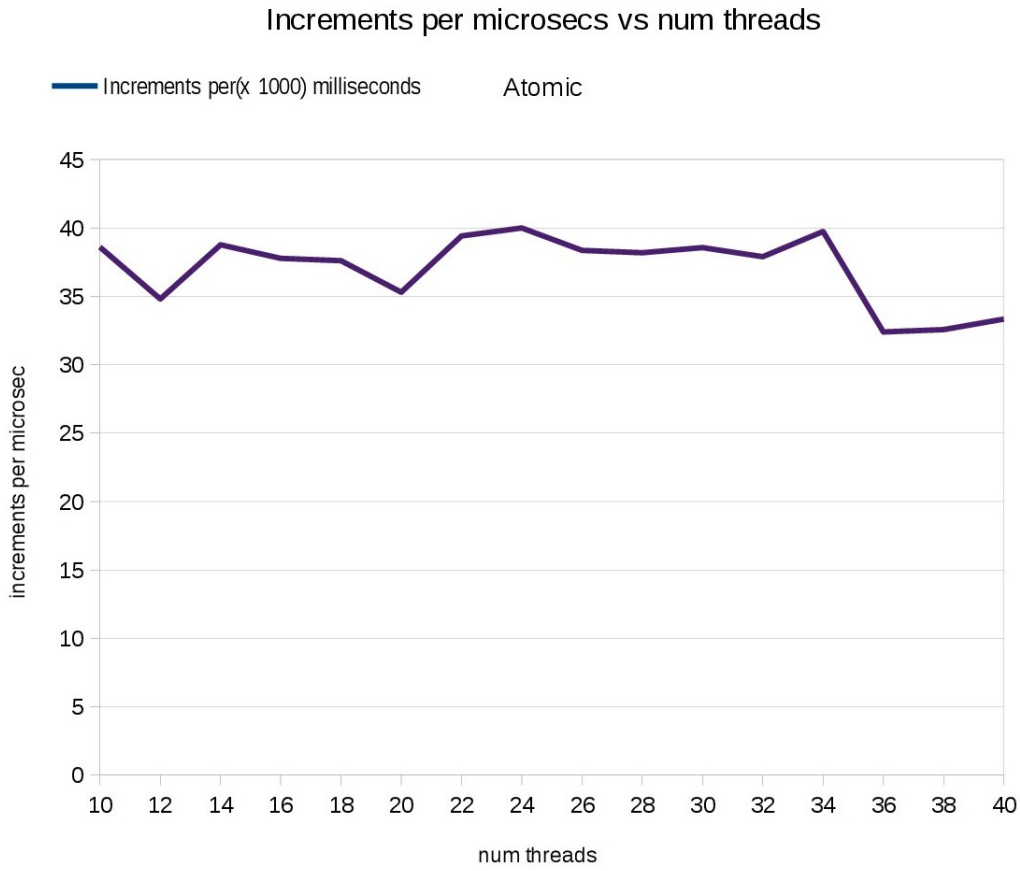
The next graph is of the case where we use the lock and unlock on a mutex.We observe that the value lies between 3.7-3.8x10^3 increments per millisecond and the value is fairly consistent and linear.
Case:Lock



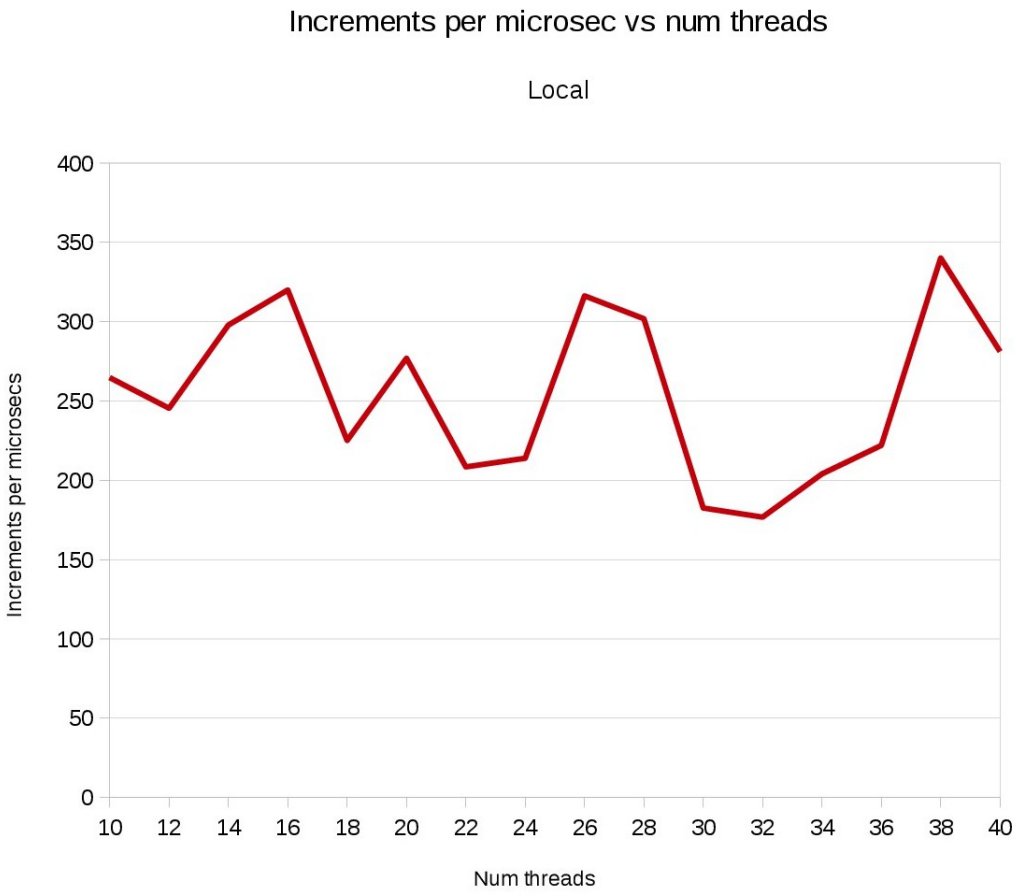
Similar case to the locks, the lockguard has a fairly consistent value between 3.4-3.5x10^3 increments per millisecond and the value is fairly consistent and linear.
Case:Lockguard



For atomic case the updates are much faster than the lock cases and the value lies between 35-40x10^3 increments per millisecond and the value is fairly consistent and linear.
Case:Atomic



For the local case the value fluctuates from 180-350x10^3 increments per millisecond and the value is not constant and neither linear.But is much faster updates than any of the other cases
Case:Local



The second experiment was performed on the node2x18a from 2am to 6am with least traffic.The machine boost of having 72 nodes and has cache of 45MB.The iteration value was 9,000,000 this time.The results are as follows:-

height2*Number of threads	Counter value					Increments per 1000 milliseconds				
	No sync	Lock	Lockguard	Atomic	Local	No sync	Lock	Lockguard	Atomic	Local
26	10635494	234000000	234000000	234000000	234000000	10.21	4.5	4.48	47.46	320.54
28	16350872	252000000	252000000	252000000	252000000	16.31	4.52	4.3	47.63	356.94
30	13729512	270000000	270000000	270000000	270000000	11.33	4.78	4.58	48.91	419.38
32	14246735	288000000	288000000	288000000	288000000	11.39	4.59	4.32	46	450
34	18953688	306000000	306000000	306000000	306000000	13.76	4.53	4.24	47.29	318.75
36	13957387	324000000	324000000	324000000	324000000	9.43	4.38	4.18	46.61	276.92
38	14466658	342000000	342000000	342000000	342000000	8.98	4.22	4.29	47.5	475
40	19881717	360000000	360000000	360000000	360000000	12.42	4.3	4	48.6	349.51
42	20286727	378000000	378000000	378000000	378000000	12.01	4.34	4.02	48.09	295.31
46	28561867	414000000	414000000	414000000	414000000	16.7	4.22	4.07	46.99	311.27
50	30733708	450000000	450000000	450000000	450000000	15.21	4.05	3.7	47.36	416.67

Thus we see that increasing the number of threads sometimes causes the increment per millisecond to go down.But this result seems much more consistent than the previous one,maybe because of the low traffic issue.If the output files are required I will be happy to provide them.Please let me know.