NYU Tandon School of Engineering
Computer Science

Nov 22, 2021
CS-GY 6233, Fall 2021

Homework 4

Tyler Kobil (tak7229) and Max Christ (mmc639) worked together on this homework.

To answer the questions asked in this assignment, we ran each program (parallel_hashtable, parallel_mutex, and parallel_spin) 6 times using varying numbers of threads (1, 5, 8, and 12), and recorded the run times in the following tables. Each column is a measurement of the time it takes to complete the insertion phase ("Inserted") or the time it takes to complete the retrieval phase ("Retr") for a given number of threads ("1 Thrd", "5 Thrd", etc).

Results of running ./parallel_hashtable (all measurements in seconds):

Inserted, 1 Thrd	Retr, 1 Thrd	Inserted, 5 Thrd	Retr, 5 Thrd	Inserted, 8 Thrd	Retr, 8 Thrd	Inserted, 12 Thrd	Retr, 12 Thrd
0.009845	11.78068	0.009401	6.878818	0.013307	6.39799	0.011442	6.886452
0.008162	12.066315	0.018134	6.537905	0.012765	6.634285	0.011706	7.027803
0.007827	12.009887	0.010847	6.630116	0.011524	6.65934	0.010958	7.271324
0.009014	12.549524	0.013002	6.593429	0.013355	6.722465	0.011554	7.333203
0.007662	12.062564	0.010661	6.6124	0.010214	7.000784	0.012307	7.05991
0.011095	11.919766	0.009002	6.309651	0.011597	6.616444	0.011184	6.922029

Results of running ./parallel_mutex (all measurements in seconds):

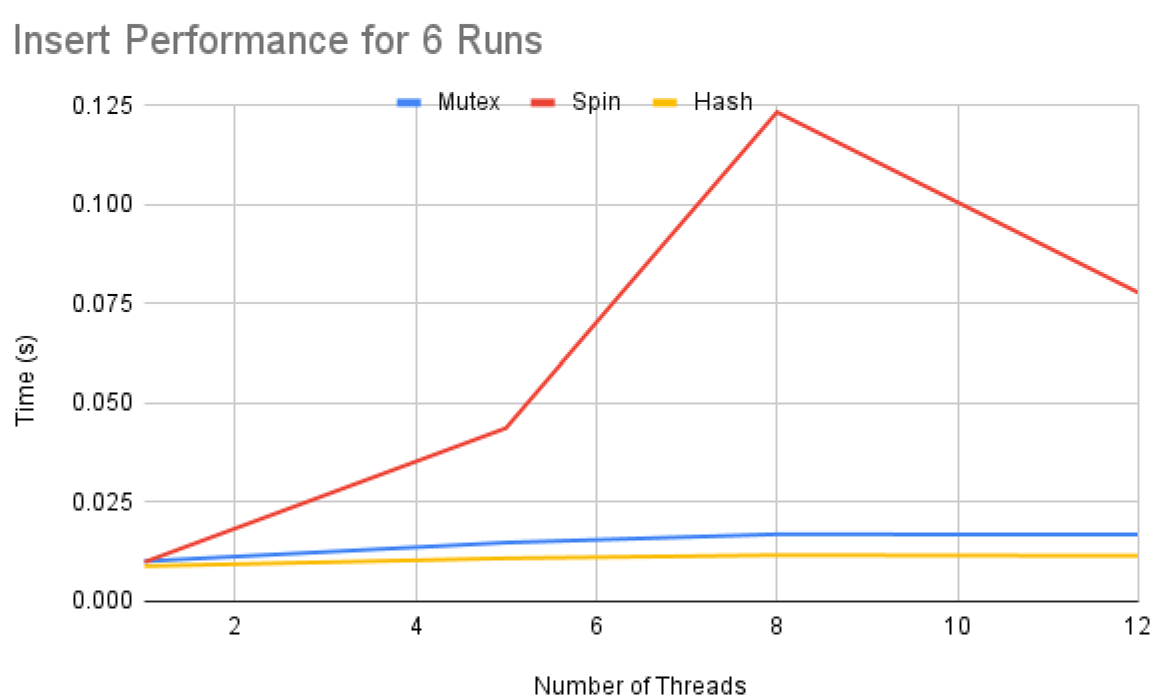
Inserted, 1 Thrd	Retr, 1 Thrd	Inserted, 5 Thrd	Retr, 5 Thrd	Inserted, 8 Thrd	Retr, 8 Thrd	Inserted, 12 Thrd	Retr, 12 Thrd
0.009445	11.833044	0.014189	5.743001	0.015548	5.911221	0.031753	6.149958
0.009303	11.441753	0.013932	5.734635	0.015269	6.116738	0.01574	6.327749
0.009931	11.938756	0.013651	5.831939	0.015621	6.532999	0.016009	6.13926
0.00937	11.499608	0.017083	5.854591	0.016421	5.790719	0.017709	6.222485
0.009858	11.561796	0.015087	5.947623	0.015716	5.961647	0.017316	6.105914
0.011339	11.700696	0.013535	5.871121	0.020043	5.932088	0.016431	6.233931

Continued on the next page...

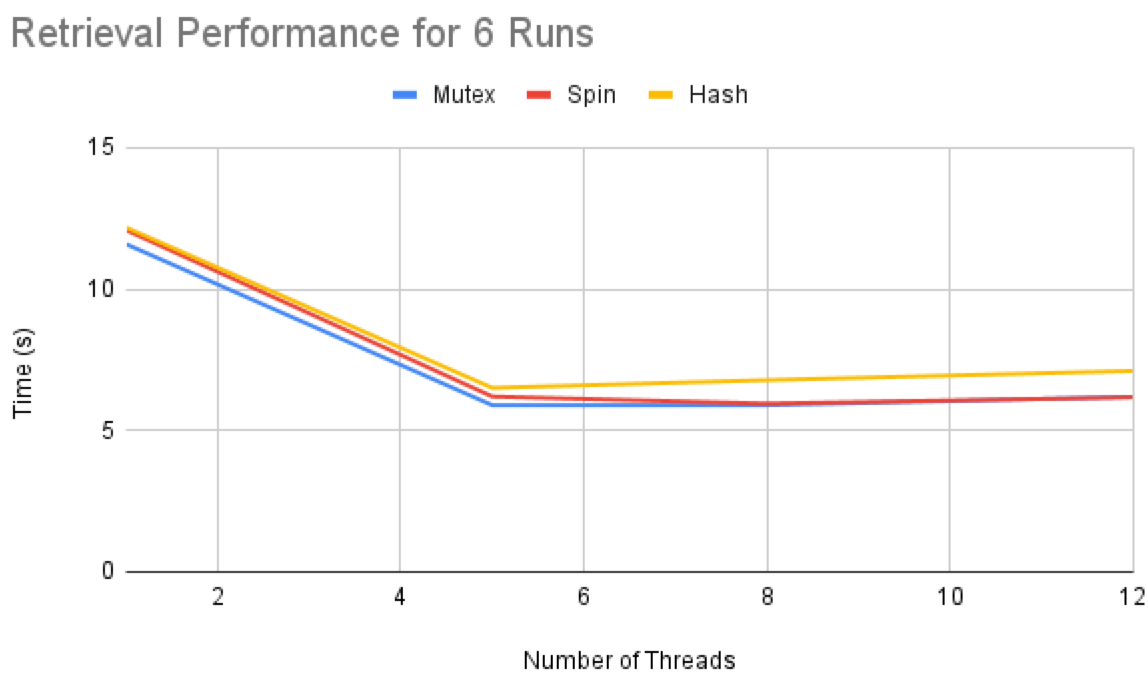
Results of running ./parallel_spin (all measurements in seconds):

Inserted, 1 Thrd	Retr, 1 Thrd	Inserted, 5 Thrd	Retr, 5 Thrd	Inserted, 8 Thrd	Retr, 8 Thrd	Inserted, 12 Thrd	Retr, 12 Thrd
0.008197	11.781194	0.026405	6.421035	0.05222	5.94198	0.214829	6.370782
0.008712	11.876041	0.028758	5.992981	0.04772	6.023796	0.193427	6.536658
0.008898	12.332464	0.055853	6.627934	0.1975	5.969094	0.03748	6.533358
0.00823	12.057163	0.038149	6.279146	0.040384	5.99887	0.038217	6.14428
0.011167	12.155888	0.03678	5.956589	0.057448	5.900442	0.166468	6.309715
0.011413	12.03578	0.044148	6.359538	0.198268	5.947785	0.069211	6.062762

Next, the columns were averaged and plotted to better understand the data. The labels in the legends correspond to the three different programs (Hash == parallel_hashtable, Mutex == parallel_mutex, and Spin == parallel_spin).



Continued on the next page....



Analysis, notes, and thoughts on timing differences between using mutexes and using a spinlock:

As explained above, to examine any differences in timing, we ran each program 6 times for a number of different thread counts (1, 5, 8, and 12). We found that there is a much greater degree of variability in the time it takes for the spinlock program to run, as opposed to the mutex program (which is very consistent). Overall, the average time of the spinlock program was much higher than the mutex program. We did not expect this, because we assumed that for quick operations like the ones we are performing, the overhead associated with the system calls for locking/unlocking the mutex would be greater than just spinning. However, we theorize that the greater variability and higher average time is due to the fact that when a thread is spinning, it is not put to sleep like when using a mutex. This means that the thread can be preempted due to hitting it's quanta, and this can happen after a lock has been acquired. This would lead to increased wait times for other threads that are waiting for that lock, because the thread was put to sleep before it could release the lock. This should not happen in the mutex method, because when a thread wakes up to get a mutex, it will either acquire it, perform the needed actions, and release it right away, or it will go back to sleep if it cannot acquire it. Going to sleep will free up the CPU for other threads to use, which will also help to improve overall performance.