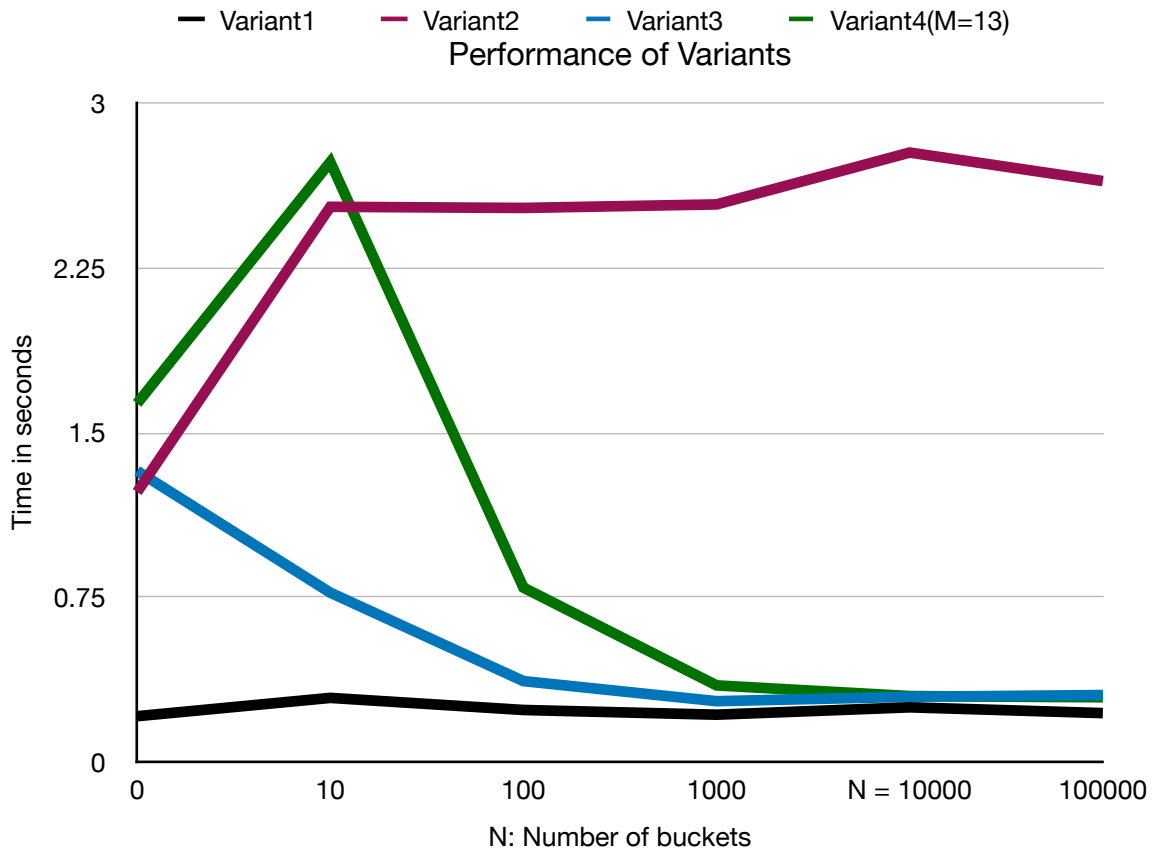
Parallel Histogram
Parallel Computing
Assignment 1
Multithreading in C++
Report

A parallel histogram using Multithreading in C++ is implemented. The C++ program checks the number of concurrent threads that the hardware can support and then creates those number of threads using a loop where each thread takes up a statically allocated set of tasks, which in this program is creating a number of random numbers and entering them into their correct histogram bucket in parallel. In order to avoid the race condition and to ensure synchronisation in adding elements to the histogram buckets in parallel, the following methods are implemented:

1. Histogram buckets as atomics
2. Single Mutex of the Histogram
3. Mutex for each Histogram bucket
4. Mutex per M buckets of Histogram.

All of these variants are implemented in switch cases inside the add function of the histogram class. Upon execution the user inputs which variant which they would like to run and that switch case is executed inside the add method of the histogram class.

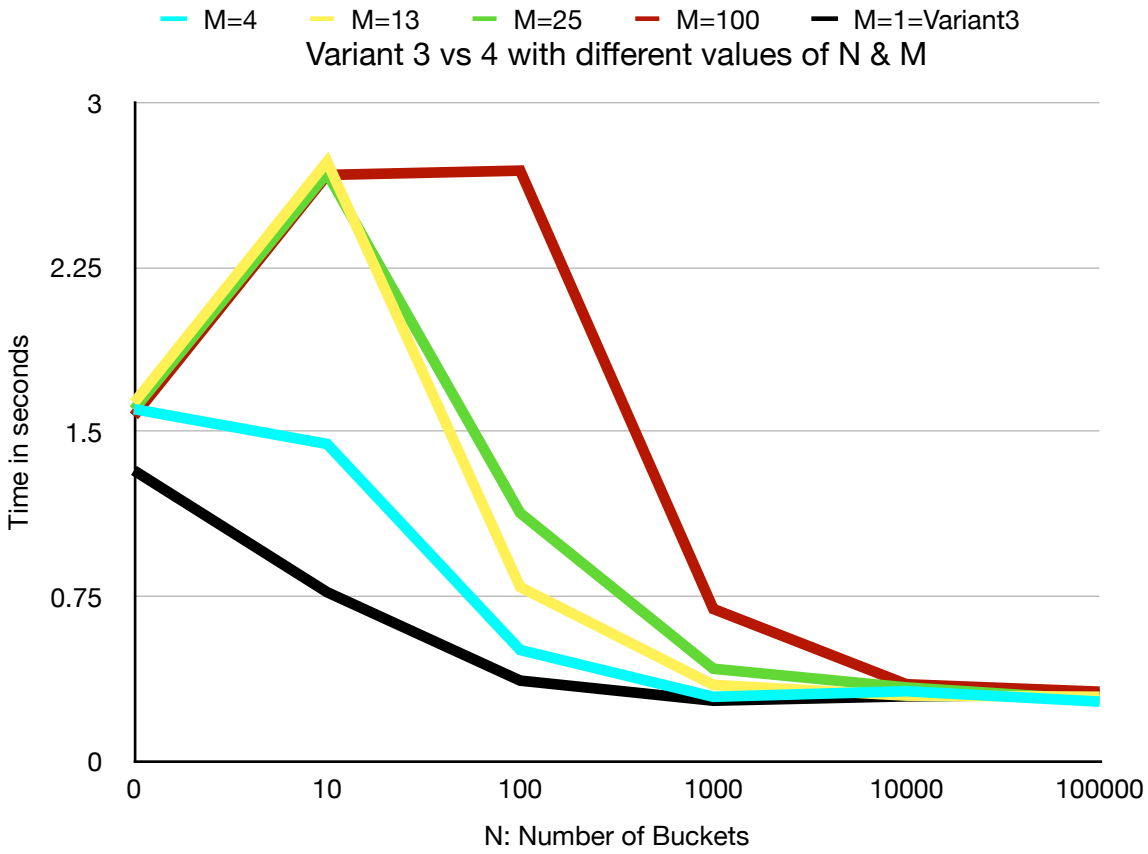
The performance of all the variants can be seen through the plots below and also in the Raw_Data file where the results of the program are recorded in detail.



The above displayed graph shows the performance of all the variants where the sample size of the data is kept constant high enough to get the runtime close to a second. The Number of buckets or the N value of the histogram is varied to see the differences in execution times.
Based on the obtained results we can observe:

1. atomics have the fastest execution time.
2. Variant 2 where there is just 1 Mutex for the histogram is the slowest.
3. As the number of Mutexes come close to the N value the execution becomes faster and then stagnates at roughly the same time which is quite fast.
4. Atomics is almost 10 times faster than the single mutex.
5. As the N value of the histogram becomes larger, the difference in performance of mutex for each bucket and the mutex per M buckets becomes smaller.

Below is the graph comparing the performance of Variant 3 and 4 for different values of M in variant4.



The colourful lines represent the time in seconds for execution of critical section in variants 3 and variants 4. Variant 3 is a type of Variant 4 where M = 1 or in other words there is a mutex for every histogram bucket.

We can observe that Variant 3 is certainly faster than Variant 4 unless M=1 where they become the same. However, as the N value rises the performance difference between Variant 3 and 4 becomes smaller. In Variant 4 as the value of M becomes higher the Mutexes are slower for the smaller values of N.

The individual performance graphs of all the variants along with their output results can be found in the Raw_Data file submitted along with this report. In addition, a working .cpp file implementing the program is attached with this report.