Report

The following graphs illustrate how many times the program truly worked – counted the number of 1s in the array correctly – out of 100 trails using different number of threads.

Arrays are generated randomly, having numbers between 0 and 5. The following experiments are combinations of arrays, their lengths, and number of threads.

Note:

1. Experimenting 1 billion integers in an array and running it 100 for multiple experiment is out of the capacities of the laptop and its memory.
2. Time taken is the average time taken by each thread.

**Race Condition**

Analysis:

On a small data as 100 integers, all threads worked properly. It was expected that the time decreases upon increasing the number of threads, yet it was the case after increasing from thread 4 to 8.

This might be due to having a small data and that the waiting time for the threads becomes greater than the time saved in using threads.

Analysis:

As before, all threads worked properly, and it was expected that the time decreases upon increasing the number of threads. However, the time taken continues to increase upon increasing the number of threads. The reason might be as before.

Analysis:

Unlike the previous two, the time taken continued to decrease as the number of threads increases as expected. However, minor errors occurred when increasing the number of threads starting from 8. This is due to idea that increasing the number of threads increases the possibility of conflict and race between threads especially on relatively small data.

Analysis:

In this and following experiments, using only one thread will output accurate results, the normal running program scenario.

However, there was an unexpected increasing in using 2 threads and a sudden drop in taken time when having 8 threads.

On the other hand, the errors decreased only upon increasing the number of threads. For 100k integers, 128 or 256 might be the optimal number of threads that have less executing time and errors.

Analysis:

Starting from 100k, no multithreading program worked properly.

However, we could notice a pattern that looks like a normal distribution. The peak is gradually shifting to the right as we increase the number of integers. The peak in these experiments is shifted from 1 to 2, then from 2 to 4, and is shifting to 8 upon more increase in the number of integers.

**Mutex**

Analysis:

In all mutex using experiment, we could notice that using locks will prevent the programs from racing conditions and erroring, yet this would cost some time because the processors have to have for the mutex to unlock. Also, we conclude for 100 integers, 8 is the optimal thread number to be used and using more will be burden due to the increase in waiting that is greater than the time saved by multithreading.

Analysis:

Even though the time taken by each processor has been decreasing until using 32, there was an extra time taken when increasing the number of processors from 32 to 64. This is because 1000 is not a multiple of 64 and that the last thread will have the greater remaining chunk of subarray out of all the previous threads that took equal yet very small distribution. Running 100 times meaning 100 last threads are adding more time to the total and increasing the average time taken by each processor consequently.

Analysis:

It could be seen that there is no detected pattern. The reason could be that having a relatively big data and a random one means taking time that differs in each run. Also, even though the race condition has been resolved, the race to lock the problem existed.

Analysis:

Upon increasing the dataset, an expected decrease in the average time taken has been detected but the performance stoped increasing after 16 threads that might be considered the optimal thread number for 10k integers.

Analysis:

Starting from 100k, as the race condition experiment has previously found, there exits a normal distribution that is shifting to the right as the number of integer increases.

Analysis:

However, when having 100M as data set, we conclude that are two peaks: at 4 and at 64(small).

As a final conclusion, mutex fixes the race condition however takes in many cases a lot of time than not using multithreading in some cases in large data set due to the continuous locking and unlocking and increasing the number of unnecessary threads.

**Private Counts**

Analysis:

Similar analysis to previous categories.

Analysis:

Similar analysis to previous categories.

Analysis:

Similar pattern to the previous graph.

Analysis:

Similar analysis to previous categories except that in private counts: at 64 threads, the average time taken is beginning to increase unlike before that kept decreasing even at 64.

Analysis:

The optimal thread is 32 because the performance reached its greatest at 32 threads.

Analysis:

The time taken on average didn’t increase at 64 due to having a larger data set that require more necessary threads.

Analysis:

Similar to the previous graph.

However, the more we increase the data, the more we reach a theory: doubling the threads -> doubles the performance -> half the time.

In conclusion, the private counts approach was more efficient than mutex and was near doubling the performance in big datasets due to using different memory and no locks.

**Padding Caches**

Analysis:

Similar analysis to previous categories

Analysis:

Similar analysis to previous categories

Analysis:

Other than having performance locked at 8 threads, having only one thread was much faster than multithreading which is quite unexpected.

This might be due to not initializing multiple dummies that might take time.

Analysis:

Similar analysis to previous approaches as mutex and others.

Analysis:

Similar analysis to previous approach (private count)

Analysis:

Similar analysis to previous approach (private count) – doubling performance-

Analysis:

Similar analysis to previous approach (private count)

In conclusion, this approach was much faster than all before (in general) because it doesn’t use locks and the processor don’t have to fill other parts of the cache line.