Stations

CMP 202 2023/24 Term 2

CPU Mini-Project

Annie Place

2301241

Introduction

Displaying accurate Train simulations are an entertaining and helpful way to show the use cases for mutex locks. With this simulation, the locks and their usefulness help with efficiency and application construction. The build would not be as useful without parallelisation and due to the mechanics of this programming methodology the stations run smoothly.

Application Overview

The application is based upon a program originally developed by Javad Zarrin. There exist five train objects moving across a track consisting of interchanging track segments and stations. These trains move at varying speeds in varying directions across the track. Their behaviour dictates that none of the trains are allowed to collide into one another and must not reside on the same track segment at the same time. When entering a station, the trains check to see if the next track segment contains a train. If this is the case, the train waits in the station until the train on the track exists that segment. The program ends when all the trains finish their journeys, journeys that end when they reach the end of the track based on their starting position.

Parallelisation Technique

The trains run in parallel from one another, locking when entering a station whilst seeing a train on the track in front of them. The standard database command unique_lock was utilized to lock the trains when needed. This, on top of creating a more efficient program, prevented trains from colliding into one another whilst on the track. The trains would effectively stand idle in the stations until their turn to go across the track came up. In the sequential version of this code, the trains cannot run in parallel of one another, so each train goes across the track one at a time. This results in a very inefficient track.

In order for the trains to run in parallel with one another, a set of mutex locks were used to lock certain trains in stations to prevent crashes. On top of this, condition variables were utilized to notify other trains when a track is free or occupied. The following lines provided the mechanism for the trains to communicate with one another:

```
std::lock_guard<std::mutex> lock(segments[segmentIndex]->mutex);

segments[segmentIndex]->occupied = false;

segments[segmentIndex]->cond.notify_one();
```

The sequential version of this program contains no such locks and thus its performance suffers greatly.

Performance Analysis

To conduct testing the CPU used was a 13th Gen Intel Core i7-1360P with 16 GB of RAM on a 64-bit operating system. The operating system used was Windows 11 Home.

On average, the parallelisation version of the program took 78051 milliseconds whilst the sequential program took three times that amount, 227484 ms. This immense difference can be found in the code for each. The sequential program, having each train run one after the other, will take roughly O(N) to finish whilst the parallelised methods will take an amortized O(N) due to its usage of parallelism. The parallel trains cut the time it takes by a third because for most of the runtime, the trains are running side by side together unless a track is occupied by another train. In a real-world scenario, this stark difference confirms that a parallelised version of the program is more effective, which holds with reality as having trains run on the same track is something regularly practiced on all railways.

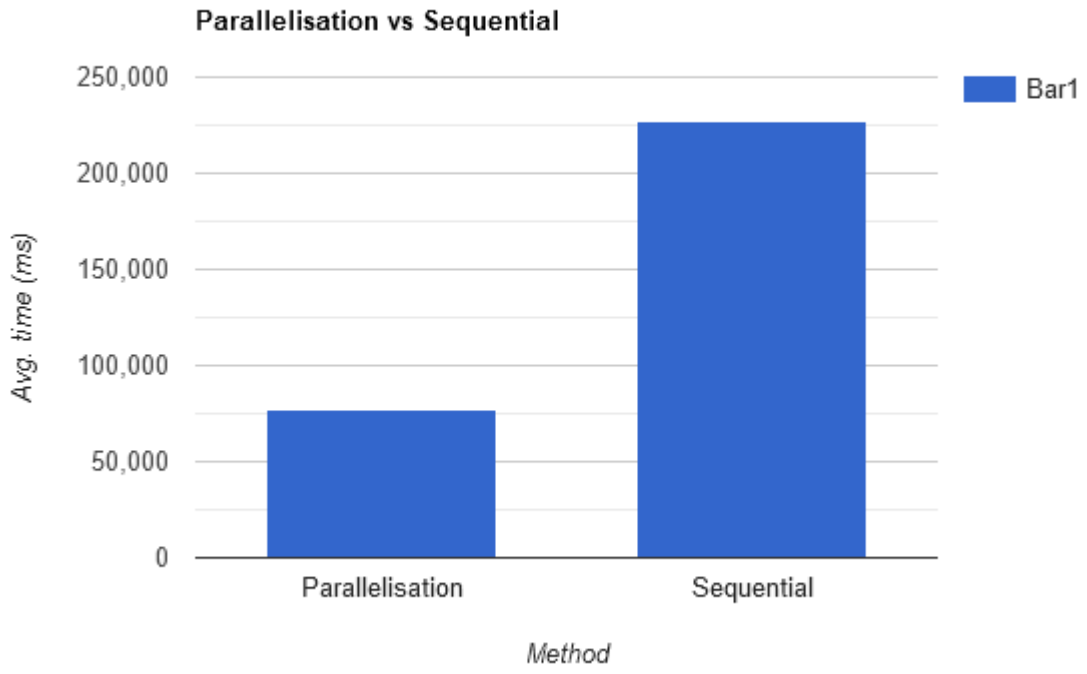

Fig 1. Bar graph showing the difference in runtimes between the two programs.

Additional Difficulties

During programming, there were several different errors that propped up. Oftentimes due to incorrect mutex usage the trains would lock on unoccupied tracks and create a deadlock in the program. Fixing the usage of condition variables with the mutex remedied this issue and the trains worked properly. Another issue in parallelisation occurred when the trains would collide with each other. Once the condition variables were properly implemented this issue also disappeared. Lastly, the construction of the sequential program used to have all the trains running in tandem, resulting in several collisions and improper track usage. Having the trains run one after another solved this issue.

Conclusion

Having trains utilize the same railway at the same time is obviously more efficient and these programs display this. If trains ran in real life as they do in the sequential program their inefficiency would cost significant amounts of time and dollar. The methods of parallelisation prove to be a practical design choice in several endeavours, and train stations are no exception.