Assignment 1 Report

Julius Putra Tanu Setiaji (A0149787E), Chen Shaowei (A0110560Y)

1 October 2018

1 Program Design

This train simulation is implemented in OpenMP. Primary design considerations are:

- Each train is simulated by one thread as required in the specifications.
- This is achieved by calling `omp_set_num_threads(num_trains);`
- Each train stores its next state, and when it will enter that state.
- Each train will generate the time at which it will open its doors when it starts waiting at a station.

1.1 Assumptions

- Only one train can open its doors at each at any one time, regardless of direction.
- Train stations have infinite capacity for waiting trains.
- Time units are discrete and can have no subdivisions
  - **Implication**: It is sufficient to store all time units as integers instead of floating point numbers
- Trains must open their doors for at least 1 unit of time.
  - **Implication**: We round every randomly generated door open time up to the nearest integer

2 Points to Note / Implementation Details

- Current simulation time needs to be shared across all threads. In addition, time can only be advanced after all threads have completed the actions to be done in the current tick.
- Each train keeps track of its next action time, and actions to be completed within the current tick are performed in a `while` loop.
- There is a `#pragma omp barrier` to ensure that all threads exit the `while` loop before time is advanced.

- The advancement of time is done in a `#pragma omp single` block to ensure that it is only performed by one thread. In addition, `#pragma omp single` has an implicit barrier at the end of the block so this prevents other threads from entering the next iteration of the `while` loop until the advancement of time is complete.

- Certain resources i.e. train tracks, door-opening rights are limited, and only one train may access them at any time.

    1. One way to implement this is to have threads block when waiting to access such resources. However, this would interfere with the way we have chosen to implement time within this simulation. Blocked threads would prevent code after a `#pragma omp barrier` statement from being executed. We therefore decided **not** to go with this implementation.
    2. An alternative way to implement is to have a queue of trains requesting access to such resources. At each tick, each train would check whether it is at the head of the queue, and if so, it will be able to access the resource. We realised that it will also be necessary to store the time at which a train would be able to gain access to the resource, since our simulation time advances in integer increments, but door open time may have a fractional value. Upon further consideration, we realised that this design could be simplified.
    3. The key insight we arrived at is that any train waiting for access to a resource only needs to know the next time said resource will be available. Since this system does not permit a train to give up waiting for a resource, this can be implemented simply with a thread-safe timekeeper object. When a train requests access to a resource, it tells the timekeeper how much time it will occupy the resource for. The timekeeper will then inform the train of the time when the train can access the resource, and update its internal next available time. This is the implementation we decided to go with.

- The next consideration is ensuring that system statistics are reported correctly. Since we assume that trains cannot open its doors for 0s, only one train can open its doors at each station per tick. There is therefore no potential race condition in the update of statistics.

- Since print statements are not atomic, we wrapped them in a `#pragma omp critical` block to ensure that only one print operation executes at any one time.

3 Execution Time and Analysis

Note: To facilitate more accurate execution time analysis, we disabled per-tick status output for the following tests.

4 Discussion

5 Bonus