Train Network Simulation using Parallel Programming (MPI)

Agus Sentosa Hermawan & Jerrell Ezralemuel

Full data can be viewed at this Sheets and testcases and code can be viewed at this Github

Assumptions

- The assumptions taken for using this method is the computer can handle the number of links to be processed.
- The train is larger than the number of links such that simulating every tick is worth the computation and sharing the train information with all of the links is worth the computation, hence this implementation will be efficient.

Implementation Details

In a nutshell the implementation of the train simulation is by simulating every links in the graph. where every links have their own copy of the previous station and taking control of it simultaneously.

Stats

Stats is a class storing the basic statistic of a station, `min_time` is storing the minimum delta between two loading. `max_time` is the maximum delta between two loading, `sum_time` is the sum of the delta and `count` is the number of loading happened in the simulation such that from `sum_time` and `count` we can infer the average delta

Train

Train is an class storing it's `color`, `destination` and `next_destination`. this object is free of pointers and addresses s.t. it can be sent to other processes easily.

Link

Link denotes the linkway between two stations lets say the link connects station A and station B, `current_station` will be A and `next_station` is B. doing the housekeeping of each `current_station` on its own. even though there might be more than 1 process doing the station this is worth it, as in having a "master link" that handle that station this create a heavy process as in the master link have heavier load as the process had to maintain the process and we will have another latency in communication as that process had to send that train to other links that is responsible of that train. link have a queue in which a train is in this queue if and only if the train have load the passanger but the train is still queueing for the link, this is a better representation. and if a train arrives at the end of the link this link will broadcast the train at the end of the road, to make this more efficient we use allGather. to reduce the communication cost.

every train random loading time is determined before sending because we have to ensure all of the links have the same loading time for that train as we have to distribute it to all links connected to the `next_station`

Station

Station is an object taking charge of the station queue loading and unloading and filtering which train is going through the specific link. and doing the station statistics.

Starvation & Deadlocks

This method is also won't introduce starvation in any way because everything is handled in a queue where every train have their own turn. The 2 allGather wont create any deadlocks as it is implemented the lower id station is resolved first like in dining philosophers problem and this is very safe.

Performance Difference

We can state the obvious in here the computational complexity of the MPI and OMP is different as in this is a problem viewed from 2 different perspective in MPI the complexity scales with the number of links and in OMP it scales with the number of trains. as in OMP implementation will be faster on a system with 10000 edges and 10 train, and the MPI implementation will be worse on that. but the Link implementation / MPI implementation will be better on a system of densely populated train system as there is only several edges compared to trains.

OMP

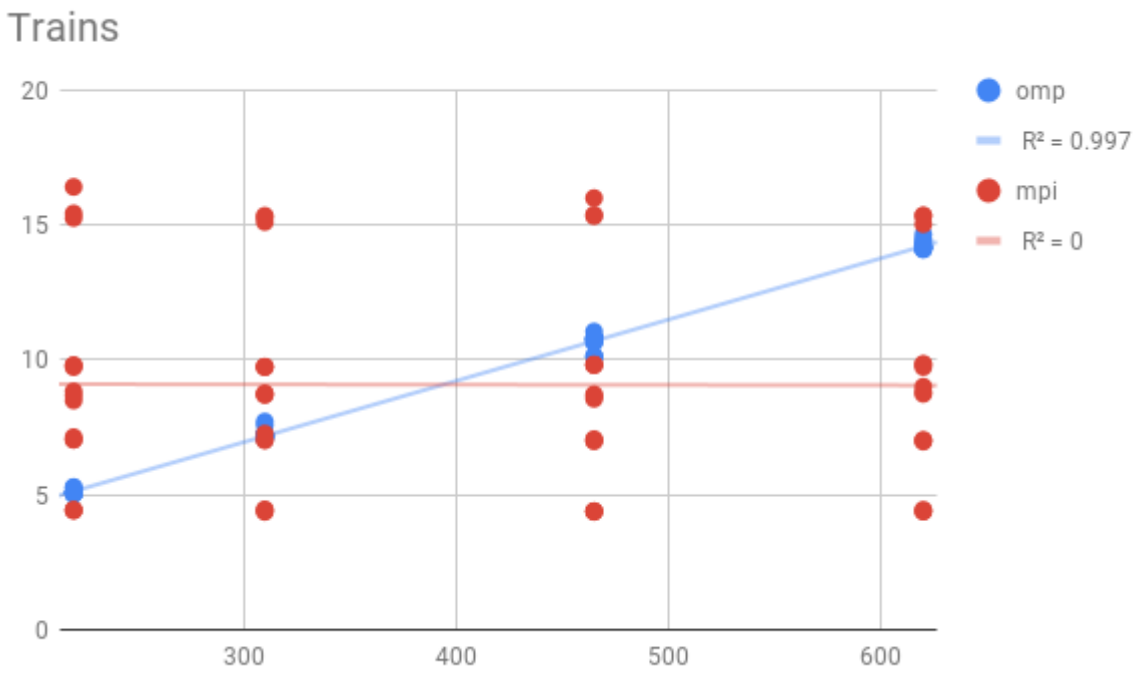
	220	310	465	620
14	5.039	7.719	10.642	14.098
24	5.058	7.153	10.773	14.133
34	5.045	7.133	10.693	14.115
42	5.068	7.126	10.131	14.162
52	5.070	7.074	10.790	14.175

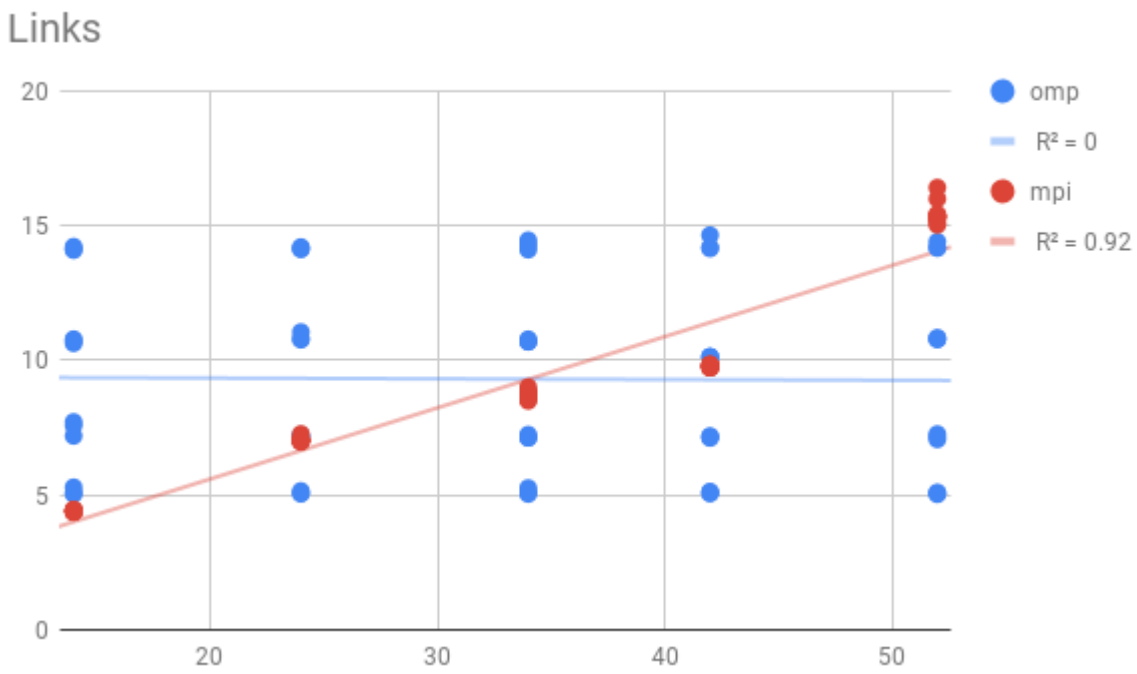
MPI

	220	310	465	620
14	4.424	4.371	4.372	4.392

	220	310	465	620
24	7.047	7.028	6.994	6.977
34	8.502	8.713	8.563	8.746
42	9.802	9.744	9.818	9.74
52	15.265	15.117	15.334	15.035

the data also shows the same thing about correlativity of number of trains in each column and number of links that is in the row.





Bonus

Our implementation wont introduce any starvation at any point in time as it is very safe. this implementation wont introduce any deadlocks and starvation. this is done as we have stated on the implementation details and why this method is very safe. The OMP implementation is the same as well as it uses some sort of dispatching system that resembles a queue. No deadlocks / starvation will happen.