CS3210 Assignment 03

Student: Nguyen Truc Nhu Binh - A0293856L

I. Introduction

1. Algorithm

The simulation implements a parallel-by-station approach, where each process is responsible for handling a designated group of stations. Within each station, platforms serve as focal points for train-related computations.

For each platform within a station, the following tasks are performed:

- Updating unloading/loading wait times and train status
- Assigning incoming trains to the queue based on their arrival time and id.
- Controlling the status of the connected line: i.e. a platform at clementi station, leading to harbourfront, manages the line directly connecting the two stations. Additionally, the platform updates train information, facilitating efficient routing and decision-making.

At the station level, tasks include:

- Managing inter-station train transfers: get train information from other stations to assign trains to the appropriate platform.
- Spawning new trains each tick based on predefined requirements.

To ensure policy compliance and synchronization, the master process (process rank 0) pre-assigns all train details, including their unique identifiers and respective lines. These details are then scattered to the respective processes before the simulation begins, ensuring consistency across all operations.

2. MPI Construct

MPI_Scatterv

This construct is utilized to distribute the centrally initialized train data to the corresponding station processes. Pre-defining train attributes such as IDs, spawning times, and spawning

stations in the master process simplifies the simulation by reducing the computational complexity at each station, but still adheres to simulation policies.

MPI_Alltoallv

The `MPI_Alltoallv` construct facilitates dynamic communication of train data between stations. Since it is challenging to predict the timing and destination of train movements during the simulation, this construct provides an automatic mechanism for exchanging train-related information. Each simulation tick, stations can both send and receive data as needed.

MPI_Gatherv

After completing the simulation each tick, the `MPI_Gatherv` construct collects all output data back to the master process. This allows for centralized processing and formatting of results, ensuring the outputs are presented in a coherent and appropriate manner.

3. Synchronization

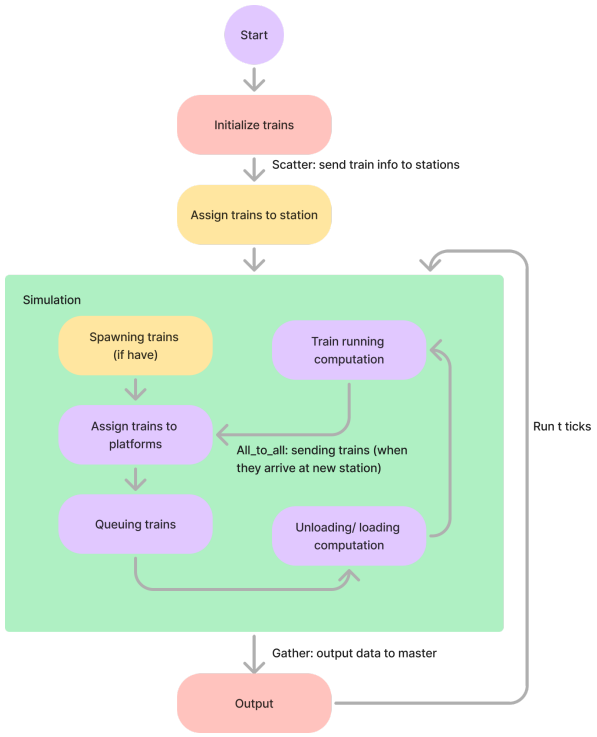
Effective synchronization is critical to avoid deadlocks and race conditions in the simulation. The following measures were implemented to ensure reliable and coordinated operations across all processes:

- **Thread-Safe MPI Constructs**
  All MPI constructs used in this project are inherently thread-safe, ensuring that concurrent operations within the simulation do not interfere with one another. This eliminates potential race conditions during inter-process communication.
- **MPI_Barrier**
  The `MPI_Barrier(MPI_COMM_WORLD)` construct is employed to synchronize all processes at key stages of the simulation. By enforcing a barrier, processes ensure that computations are completed before data exchange begins. This guarantees that no process sends or receives incomplete or outdated data, thereby preventing inconsistencies and potential deadlocks.
- **Unloading/Loading Computation Timing**
  To further ensure synchronization, the computation of unloading/loading times is delayed until all other tasks are complete. This is particularly important as trains from other stations may arrive, influencing the queue order and the overall loading/unloading process. By waiting until all tasks are finished, the simulation ensures that train

information is fully updated and consistent across all stations before any unloading/loading computations occur.

Diagram the total simulation process:

- Red: master (process 0) task
- Yellow: station task
- Pink: platform task



II. Analyzing

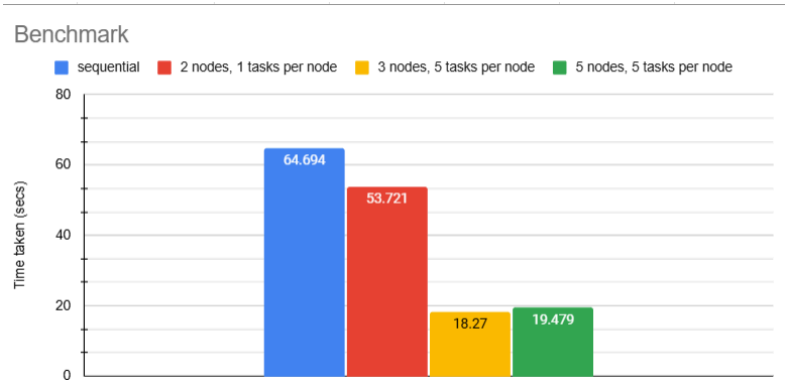
- With 2 nodes and 1 task per node, the max ratio of 1.185 indicates moderate speedup compared to the sequential benchmark.
- Increasing to 3 nodes with 5 tasks per node significantly improves performance, achieving a max ratio of 3.517.
- Using 5 nodes with 5 tasks per node shows a slight decrease in speedup efficiency, with a max ratio of 3.275, suggesting diminishing returns as more nodes are added. This might

be because the assigning nodes are different in types `[i7-7700*4&xs-4114*1]` which might take longer and more complex in communication time.

Impact of Stations and Trains

The number of stations in a line and the trains being simulated greatly affect speedup. The parallelization strategy delegates train assignment and queuing to the corresponding platforms, eliminating inter-train dependencies. Similarly, stations autonomously send train data using the `MPI_Alltoallv` construct during each simulation tick, reducing inter-station communication overhead.

2 nodes, 1 tasks per node	bench-seq	opt avg	opt min	opt med	iteration 1	iteration 2	iteration 3
min	0.501	0.687	0.679	0.689	0.679	0.689	0.693
stations	1.417	1.061	1.04	1.066	1.04	1.066	1.077
traffic	64.036	54.792	53.708	55.097	53.708	55.571	55.097
trains	63.303	53.903	52.833	54.389	52.833	54.488	54.389
big	63.872	54.430	53.96	54.652	53.96	54.652	54.678
max	64.694	54.558	53.721	54.863	53.721	55.089	54.863
max ratio	1.185791181						
3 nodes, 5 tasks per node	bench-seq	opt avg	opt min	opt med	iteration 1	iteration 2	iteration 3
min	0.501	0.969	0.966	0.968	0.968	0.972	0.966
med	1.417	1.025	1.019	1.025	1.025	1.031	1.019
stations	64.036	18.510	18.488	18.493	18.488	18.493	18.548
traffic	63.303	18.303	18.237	18.26	18.237	18.26	18.413
trains	63.872	18.306	18.234	18.285	18.234	18.285	18.399
max	64.694	18.391	18.27	18.321	18.583	18.270	18.321
max ratio	3.517635118						
5 nodes, 5 tasks per node	bench-seq	opt6 avg	opt6 min	opt6 med	iteration 1	iteration 2	iteration 3
min	0.501	1.220	1.205	1.211	1.243	1.205	1.211
stations	1.417	1.261	1.242	1.261	1.279	1.242	1.266
traffic	64.036	19.517	19.012	19.713	19.713	19.826	19.012
trains	63.303	19.421	19.333	19.4205	19.508	19.333	20.106
big	63.872	19.498	19.321	19.4975	19.674	19.321	19.825
max	64.694	19.752	19.479	19.6	19.479	20.178	19.6
max ratio	3.275258619						



III. Bonus

This code still run well when number of line > 3

Appendix

- Code to generate test cases:

python3 gen_test.py 30 10 10 500 15 500 > min.in

python3 gen_test.py 2000 10 10 4000 1000 5000 --num_ticks_to_print 1 > trains.in

python3 gen_test.py 3000 10 10 2000 1500 5000 --num_ticks_to_print 1 > stations.in

python3 gen_test.py 2000 10 10 1000 2000 5000 --num_ticks_to_print 1 > traffic.in

python3 gen_test.py 3000 10 10 3000 3000 5000 --num_ticks_to_print 1 > max.in

- Configuration nodes and tasks per node:

2_1:

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
```

3_5:

```
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=5
#SBATCH --time=00:10:00
```

5_5:

```
#SBATCH --nodes=5
#SBATCH --ntasks-per-node=5
#SBATCH --constraint="[i7-7700*4&xs-4114*1]"
```

- Benchmark logs:

https://drive.google.com/file/d/189wij6JgO-A5tC2B_jKl2MpEkldliuZ5/view?usp=sharing

- Google Sheets of the benchmark:

https://docs.google.com/spreadsheets/d/1qqkV4LnuxPtovFv3arEod4-qlrPVcw6wLfZ4WqwB_ao/edit?usp=sharing

- Tests used for benchmarking:

https://drive.google.com/file/d/1eIC7ZTq2NydvXsl59qFi-h2FvVwj48w8/view?usp=sharing