LLPP: Assignment 3

Arveen Emdad Hamit Efe Çınar Iason Kaxiras

22 February 2024

1 Documentation

By default, the serial implementation will be selected. To choose different versions, pass in the `--pthread` flag to use C++ threads and `--omp` to use OpenMP. To use vectorization, pass in the `--vector` flag.

```
$ demo/demo --timing-mode --vector scenario.xml
```

By default, the current code utilizes OpenMP with vectorization. To change this behavior, you must comment out the `#pragma omp parallel for` directive in `ped_agent.cpp` under the corresponding function, `Tagent::computeNextDesiredPositionSIMD()` and rebuild the program.

Note: If multiple implementations are passed via command-line arguments, the last implementation passed will be used.

You can change the number of threads for C++ threads implementation using the `--nthreads n` flag where `n` is the number of threads.

```
$ demo/demo --timing-mode --pthread --nthreads 8 scenario.xml
```

This shell command also can be used as a shortway to show the speedup values for the scenarios:

```
for scenario in scenario.xml scenario_box.xml
do
echo "*** Scenario: $scenario ***"
./demo/demo --timing-mode --omp-col --tick-count 100 $scenario
echo ""
done
```

2 Machine Specifications

The experiments were ran on the Snowy compute cluster with the following specifications:

”Snowy consists of 228 compute servers (nodes) where each compute server consists of two 8-core Xeon E5-2660 processors running at 2.2 GHz. We provide 198 nodes with 128 GB memory (s1-s120, s151-s228) , 13 nodes with 256 GB (s138-s150) and 17 nodes with 512 GB (s121-s137). All nodes are interconnected with a 2:1 oversubscribed FDR (40 GB/s) Infiniband fabric. In total Snowy provides 3548 CPU cores in compute nodes” [1].

3 Questions

A. It is not solely enough to use ”omp paralel for” directive for this assignment's non sequential part since on the cases where the agents change regions, the threads in distinct cores taking care of these respective regions may either miss the task to move the agents or redo the task. Thus, there must be some synchronization method that is taking care of these cases.

B. Since the global lock solution behaves like a sequential version, where each agent takes the lock so as to move and prevent the others, the simulation would perform notably slow. It is because each agent requires its own time to be handled, which would get slower as the number of agents increase. Plus, except 1, all of the cores would be idle while the program is running, so it would be really inefficient. As for the second simple solution, having single lock for each location is too time and space costly as it would add the tasks of locking, unlocking, checking, waiting for every time an agent moves. So, even though optimizations via parallelization can be done, this solution would suffer from its over-costly synchronization method. As the extent of the scenario gets wider, the number of locks and tasks for agents will get slower, so this approach's performance would be affected by the extent of the scenario. However, it would perform better than the first solution since it allows parallel computing, so it would be much faster in terms of operation per second. Lastly, though it is not bad idea to divide the problematic agents (that are going to cross regions), it is not quite efficient to go over some portion of agents for the second time on every tick.

C. On the worst scenario for our solution, agents would cross the regions on the borders again and again in a loop, which would result in one critical region task for each agent in every tick function. Thus, the performance would correspond to the second simple solution suggested in 2.1, which means it would be still faster than the sequential version because of the parallelization, but it would still perform poorly in terms of pace.

D. Our solution would scale on a 100+ core machine with some potential inefficiencies. It would perfectly divide the agents between the cores but as the number of cores increases, idleness of some cores could potentially arise at the end of each tick call. The worst-case for that would happen when there is 301 agents and 100 cores. If each core's thread run with equal speed, then at the end all of the threads will wait for one thread with this static scheduling.

4 Plots

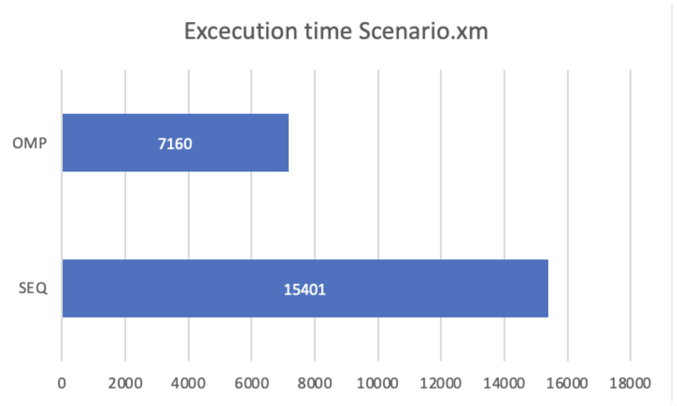

Figure 1: Execution Time of implementations running scenario.xml (in miliseconds)

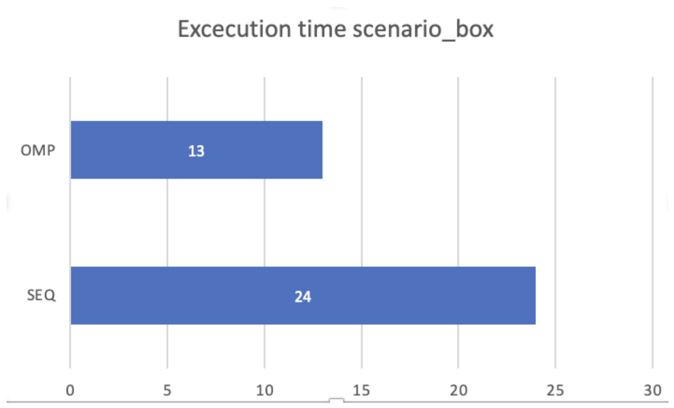

Figure 2: Execution Time of implementations running scenario_box.xml (in miliseconds)

Speedup values for scenario.xml and scenario_box.xml are 2.15098 and 1.84615 respectively.

5 Contribution

Arveen Emdad, Hamit Efe Çınar, and Iason Kaxiras have contributed equally to the solution of this assignment.

References

[1] *Snowy User Guide*. 2023. URL: https://www.uppmax.uu.se/support/user-guides/snowy-user-guide/. (accessed: 2024-01-22).