Distributed parallel Qsim implementation in Rust

Janek Laudan, Paul Heinrich, Kai Nagel

April 2024

1 Introduction

MATSim (Multi Agent Transport Simulation) [HNA16] is a well established software framework for agent-based traffic modelling. Operating on a mesoscopic scale, it supports simulating urban regions encompassing millions of individual travelers. As the capability of the model expands, so does the desire to enhance its scale or detail, increasing the computational demands of the software. At the same time, CPU (Central Processing Unit) clock rates have plateaued for almost two decades, and the miniaturization in semiconductor design is approaching physical limits [Lei+20]. Realizing performance improvements in MATSim to accommodate computationally expensive simulations while maintaining fast execution times requires leveraging the parallel computing capabilities of modern computer hardware.

Traffic in MATSim is modelled using synthetic persons which travel along a simulated road network to reach designated activity locations. Over multiple iterations of the same simulated day, synthetic persons explore various strategies to maximize their utility. One iteration consists of three phases: (1) the MobSim (Mobility Simulation) phase, where synthetic persons execute their daily plans; (2) the scoring phase, where executed plans are evaluated; and (3) the replanning phase, where a fraction of the synthetic population invents new plans to test during the next iteration.

Among the three phases outlined above, phases 2 and 3 are embarrassingly parallel problems[1]. In contrast, distributing the MobSim phase onto parallel computing hardware is more challenging, as synthetic persons interact while travelling within the simulation. Therefore, when developing a new parallel architecture for MATSim we focus on the MobSim phase, assuming that phase 2 and 3 can be integrated later.

The default implementation of the MobSim, QSim, already facilitates multicore systems using Java's concurrency primitives and a shared memory algorithm. Implemented more than a decade ago by Dobler and Axhausen [DA11], this approach effectively scales up to 8 processes. Executing the MobSim with more processes does not improve execution times, as noted by Graur et al. [Gra+21]. They hypothesize that this limitation is caused by memory bus saturation. In response, their MobSim implementation, Hermes, is single threaded and focuses on optimizing cache locality as well as memory footprint at the cost of flexibility of what can be modelled. Conversely, Strippgen [Str09] developed a MobSim implementation suitable for GPU (Graphics Processing Unit) hardware which supports massive parallelism, yet integration into the existing simulation framework proved to be challenging.

Instead of improving execution times on a single machine, we propose a distributed message passing algorithm for the MobSim that facilitates scaling traffic simulations across multiple machines. This approach offers numerous advantages. Firstly, it maintains the flexible programming model inherent to CPU-based programming, in contrast to hardware accelerators. Secondly, by distributing the simulation's load across multiple machines, we effectively prevent the saturation of the memory bus, as each machine manages only a portion of the traffic model. Implementing a distributed algorithm allows us to tap into the power of HPC (High-Performance Computing) clusters, which are typically composed of many interconnected machines. Moreover, with each process handling just a fraction of the traffic simulation, it should also become possible to improve cache locality, improving execution times on single multicore machine as well.

A distributed queue simulation was initially proposed by Nagel and Rickert [NR01] and Cetin, Burri, and Nagel [CBN03] with promising results. More recently, Wan et al. [Wan+23] implemented a distributed version of MATSim. However, their fastest execution times were observed with four processes achieving a speedup of two — less than what is offered by the current parallelization approach. Another recent development is Mobiliti, intro-

[1] https://en.wikipedia.org/wiki/Embarrassingly_parallel

duced by Chan et al. [Cha+18], which is a discrete event simulation that focuses on optimistic synchronization methods.

2 Methodology

The proposed algorithm by Cetin, Burri, and Nagel [CBN03] employs domain decomposition of the network graph to distribute the simulation workload across processes. Multiple processes may be executed in parallel on the same multicore machine, or each process can be run on separate machines. In most HPC setups, a combination is used, with multiple machines executing several processes. Each process is responsible for one domain derived from the domain decomposition phase, performing a single threaded traffic simulation for the network segment within its domain. Vehicles crossing a domain boundary are transmitted as messages to the corresponding process. Likewise, information on available storage capacities is communicated to neighbor processes. Messages are exchanged between neighboring domains once per simulated time step, consolidating all vehicle crossings and capacity updates into a single message per neighboring process.

Cetin, Burri, and Nagel [CBN03] find that the runtime of their algorithm is constrained by the latency of message exchanges, indicating that a new implementation should support low-latency networking hardware, such as Infiniband[2] or OmniPath[3]. The conventional high-level abstraction to utilize this hardware is MPI (Message Passing Interface) [Mes23] which Java — the programming language for MATSim — does not natively support. Attempts to run a Java setup with available libraries, such as Open MPI [VRS16][Gab+04], were unsuccessful. Consequently, we opted to develop a prototype in Rust, which allows direct interfacing with the C implementation of Open MPI [LH24].

This prototype can process standard MATSim input files, enabling the use of existing simulation scenarios. Domain decomposition is performed using the METIS algorithm [KK98], with the option to assign node weights that reflect the anticipated computational load for each network node. METIS balances the node-weights across different graph partitions. For the presented results, the computational load for each network node is estimated by parsing all plans of the synthetic population and counting the number of vehicles crossing each node. This way, the computational load is balanced over the simulated day, but may vary for particular time steps in the simulation.

After partitioning the network graph, each process loads its part of the network and all synthetic persons performing their first activity within its domain. The traffic simulation executed in each process mirrors the current QSim implementation described in Horni, Nagel, and Axhausen [HNA16, ch. 1]. The isolation of processes — interacting only through message exchanges — simplifies the implementation by eliminating the need for managing parallel access of data structures. At the end of a simulation time step, each process collects all vehicles crossing into another domain. The same is done for storage capacity updates of incoming links that are shared with neighbor domains. The collected information is sent to the corresponding neighbor domains in a single message, so that only one message per neighbor domain is sent per time step. After sending all messages, each process awaits messages from its direct neighbors, placing incoming vehicles on the appropriate links and updating storage capacities of outgoing links, based on the received information.

3 Main Results

The prototype implementation is tested using the existing MATSim Metropole Ruhr scenario [Rak+24]. This scenario includes a synthetic population of 491,175 persons for the 10% sample and a detailed traffic network of 547,011 nodes and 1,193,056 links. Of the four modes covered in this scenario, car, and bicycle trips are executed as network modes, while ride and walk are simulated as teleported modes (see [HNA24, ch. 12.3.1, 12.3.2]). Synthetic persons using pt (public transit) in the original scenario are excluded from the test setup because the prototype implementation does not cover pt simulation. For comparison, a single iteration of the 10% sample is run with both the current QSim and the prototype implementation across various numbers of processes. Additionally, a 1% sample of the synthetic population is tested with the prototype. These tests are conducted on an HPC cluster powered by Intel Xeon Platinum 9242 processors, each offering 48 CPU-cores. The networking infrastructure relies on OmniPath 100.

Figure 1 presents the performance outcomes of the different setups. The RTR (Real-Time Ratio), indicating the ratio between simulated time and the wall clock time required for the simulation, is used as a measure. As a

[2] https://en.wikipedia.org/wiki/InfiniBand
[3] https://en.wikipedia.org/wiki/Omni-Path

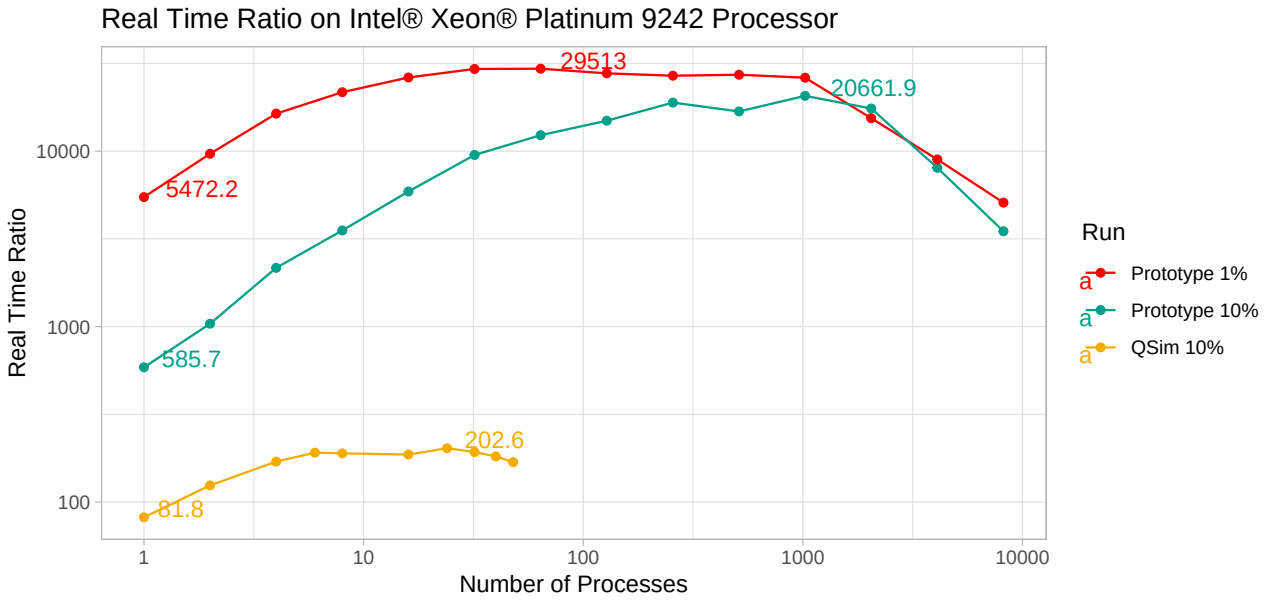

Figure 1: Real-Time Ratios of different benchmark runs

baseline, the 10% sample of the Metropole Ruhr scenario run with the current QSim implementation achieves an RTR of 82 on a single process and a maximum RTR of 202 with 24 processes, yielding a 2.5x speedup. Notably, the speedup plateaus beyond 6 processes, already achieving an RTR of 191. In contrast, the 10% sample run with the prototype reaches an RTR of 586 for a single process and a peak RTR of 20,662 for 1024 processes, resulting in a 35x speedup compared to the single-process setup. Relative to the current QSim, this represents a 102x speedup. The machines executing the simulation provide 48 CPU-cores, so that runs with up to 32 processes are conducted on a single physical machine. Simulation runs with 64 or more processes are performed on multiple machines, with process communication facilitated by the networking hardware.

Figure 1 also demonstrates the RTRs for a smaller 1% sample of the synthetic population, which runs approximately 10x faster on a single process compared to the 10% sample. The optimal RTR for this setup was achieved with 64 processes, after which it levels off for more processes. It is significant that both the 1% and 10% samples simulated with the prototype achieve RTRs in the same order of magnitude. However, for more than 2048 processes, the observed RTRs drop significantly below peak performance. Given that the HPC cluster's network operates in a tree structure, we speculate that for counts exceeding 2048 processes, messages must traverse more edges within the communication network, impacting performance.

4 Conclusion

We have successfully developed a prototype of a distributed MobSim, demonstrating the feasibility of extending the MATSim framework into a distributed traffic simulation. Our prototype outperforms the existing QSim implementation, achieving a speedup of 100 in a real-world scenario. With an RTR of 20,000, it's now possible to simulate an entire day in just 4.3 seconds. Profiling indicates that with a sufficient number of processes, the computational time for traffic simulation becomes negligible, with the bulk of runtime spent on message exchanges. While this limits further speedups under the implemented messaging methodology, it facilitates the execution of significantly larger simulation scenarios with similar RTRs. Nonetheless, the declining RTR values for extensive process numbers, as shown in Figure 1, suggest that achieving such ideal conditions requires further investigation.

The presented results demonstrate that a distributed MobSim implementation can lead to significant speedups. The subsequent step is to incorporate this methodology into the existing framework. To preserve the comprehensive functionality of the current framework without the need for extensive redevelopment, an integration must be compatible with the JVM (Java Virtual Machine) ecosystem and capable of leveraging high-performance networking hardware. Infinileap [KRS21] and hadroNIO [RKS21] are promising candidates to utilize high-performance networking within the JVM.

References

[CBN03] Nurhan Cetin, Adrian Burri, and Kai Nagel. “A large-scale agent-based traffic microsimulation based on queue model”. In: *IN PROCEEDINGS OF SWISS TRANSPORT RESEARCH CONFERENCE (STRC), MONTE VERITA, CH*. 2003.

[Cha+18] Cy Chan et al. “Mobiliti: Scalable Transportation Simulation Using High-Performance Parallel Computing”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, Nov. 2018, pp. 634–641. ISBN: 9781728103235, 9781728103211. DOI: 10.1109/ITSC.2018.8569397.

[DA11] Christoph Dobler and Kay W Axhausen. *Design and implementation of a parallel queue-based traffic flow simulation*. en. 2011. DOI: 10.3929/ETHZ-B-000040273.

[Gab+04] Edgar Gabriel et al. “Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation”. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer Berlin Heidelberg, 2004, pp. 97–104. DOI: 10.1007/978-3-540-30218-6_19.

[Gra+21] Dan Graur et al. “Hermes: Enabling efficient large-scale simulation in MATSim”. In: *Procedia Comput. Sci.* 184 (Jan. 2021), pp. 635–641. ISSN: 1877-0509. DOI: 10.1016/j.procs.2021.03.079.

[HNA16] Andreas Horni, Kai Nagel, and Kay W Axhausen. *The Multi-Agent Transport Simulation Matsim*. en. Ubiquity Press, July 2016. ISBN: 9781909188754. DOI: 10.5334/baw.

[HNA24] Andreas Horni, Kai Nagel, and Kay Axhausen. *MATSim User Guide*. Technische Universität Berlin. Mar. 2024.

[KK98] George Karypis and Vipin Kumar. “Multilevelk-way Partitioning Scheme for Irregular Graphs”. In: *J. Parallel Distrib. Comput.* 48.1 (Jan. 1998), pp. 96–129. ISSN: 0743-7315. DOI: 10.1006/jpdc.1997.1404.

[KRS21] Filip Krakowski, Fabian Ruhland, and Michael Schöttner. “Infinileap: Modern High-Performance Networking for Distributed Java Applications based on RDMA”. In: *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, Dec. 2021, pp. 652–659. ISBN: 9781665408783, 9781665408790. DOI: 10.1109/ICPADS53394.2021.00087.

[Lei+20] Charles E Leiserson et al. “There’s plenty of room at the Top: What will drive computer performance after Moore’s law?” en. In: *Science* 368.6495 (June 2020). ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aam9744.

[LH24] Janek Laudan and Paul Heinrich. *Parallel Qsim Rust*. Apr. 2024. DOI: 10.5281/zenodo.10960723.

[Mes23] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*. Nov. 2023.

[NR01] Kai Nagel and Marcus Rickert. “Parallel implementation of the TRANSIMS micro-simulation”. In: *Parallel Comput.* 27.12 (Nov. 2001), pp. 1611–1639. ISSN: 0167-8191. DOI: 10.1016/S0167-8191(01)00106-5.

[Rak+24] Christian Rakow et al. *MATSim Metropole Ruhr v1.4.1-parallel-benchmark*. Apr. 2024. DOI: 10.5281/zenodo.10959019.

[RKS21] Fabian Ruhland, Filip Krakowski, and Michael Schöttner. “hadroNIO: Accelerating Java NIO via UCX”. In: *2021 20th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, July 2021, pp. 25–32. ISBN: 9781665432818, 9781665432825. DOI: 10.1109/ISPDC52870.2021.9521601.

[Str09] David Strippgen. “Investigating the technical possibilities of real-time interaction with Simulations of mobile intelligent particles”. en. PhD thesis. Technische Universität Berlin, Oct. 2009. DOI: 10.14279/depositonce-2272.

[VRS16] Oscar Vega-Gisbert, Jose E Roman, and Jeffrey M Squyres. “Design and implementation of Java bindings in Open MPI”. In: *Parallel Comput.* 59 (Nov. 2016), pp. 1–20. ISSN: 0167-8191. DOI: 10.1016/j.parco.2016.08.004.

[Wan+23] Lin Wan et al. “PATRIC: A high performance parallel urban transport simulation framework based on traffic clustering”. In: *Simulation Modelling Practice and Theory* 126 (July 2023), p. 102775. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2023.102775.