Advancements in Traffic Simulations with *multiMATSim*'s Distributed Framework

Sara Moukir[1,3], Miwako Tsuji[2], Nahid Emad[1], Mitsuhisa Sato[2] and Stephane Baudelocq[3]
[1]*University of Paris Saclay, France*
[2]*R-CCS RIKEN, Japan*
[3]*Eiffage Energie Systèmes, France*

Keywords: Road Traffic Simulation, Big Data Analysis, High Performance Computing, Complex and Heterogeneous Dynamic System, Unite and Conquer, MATSim, Parallel Computing.

Abstract: In an era characterized by massive volumes of data, the demand for advanced road traffic simulators has reached an even greater scale. In response to this call, we propose an approach applied to *MATSim*, specifically called *multiMATSim*. Beyond its tailor-made implementation in *MATSim*, this innovative approach is designed with generic intent, aiming for adaptability to a variety of multi-agent traffic simulators. Its strength lies in its blend of versatility and adaptability. Fortified by a multi-level parallelism and fault-tolerant framework, *multiMATSim* demonstrates promising scalability across diverse computing architectures. The results of our experiments on two parallel architectures based on x86 and ARM processors systematically underline the superiority of *multiMATSim* over *MATSim*. This especially in load scaling scenarios. We highlight the generality of the *multiMATSim* concept and its applicability to other road traffic simulators. We will also see how the proposed approach can contribute to the optimization of multi-agent road traffic simulators and, impact the simulation time thanks to its intrinsic parallelism.

1 INTRODUCTION

Traffic simulation holds an instrumental position in areas like urban planning, shaping infrastructure, and crafting effective mobility strategies. With the progression of technology and advancements in modeling methodologies, simulators rooted in the multi-agent paradigm have risen as the preferred method for capturing the evolving nature of traffic systems. These tools, characterized by their intricate detail, are adept at replicating individual vehicular and road user interactions within a digital realm. Yet, as scenarios grow in intricacy and the quest for accuracy intensifies, performance and scalability concerns become glaringly apparent(Nguyen et al., 2021).

At the heart of such simulations, notably in *MATSim*(Horni et al., 2016), stands the convergence mechanism. Through a series of iterations, agents continually refine and enhance their choices, pinpointing optimal routes and timings in response to the prevailing network state and decisions made by fellow agents. The swiftness of this convergence is paramount, dictating how long the simulation takes to reach a steady state that mirrors real-world traffic patterns.

In response to these pressing concerns, we've ventured into an innovative route aimed at hastening the convergence timeline. Drawing inspiration from the "Unite and Conquer"(Emad and Petiton, 2016) (UC) principle, our tactic deviates from the classic "Divide and Conquer" perspective. Instead of dissecting a challenge, "Unite and Conquer" revolves around disseminating crucial data across co-methods. Each can then address the overarching challenge autonomously, refining the conditions for recommencement and speeding up convergence. The approach adopted is invariably influenced by the nature and context of the problem at hand.

Leveraging this mindset, we birthed *multiMATSim*. In this exploration, we probe into *multiMATSim*'s ability to scale and evaluate its efficacy across a spectrum of computational setups, encompassing the esteemed Fugaku supercomputer(Sato et al., 2020). Intriguingly, while *MATSim* faltered in its performance on Fugaku, potentially owing to its CPU structure, *multiMATSim* consistently showcased commendable outcomes. These revelations not only underscore the resilience and versatility of our universal model but also illuminate ways in which traffic simu-

Moukir, S., Tsuji, M., Emad, N., Sato, M. and Baudelocq, S.
Advancements in Traffic Simulations with multiMATSim's Distributed Framework.
DOI: 10.5220/0012452600003636
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 1*, pages 374-385
ISBN: 978-989-758-680-4; ISSN: 2184-433X
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

lation tools can be adeptly tailored to fit the nuances of varied computing environments.

2 RELATED WORK

Road traffic simulation tools have become a focal point in academic circles, especially when bolstered by parallel and distributed computing methods to expedite runtimes. Diverse approaches have been charted to navigate this multifaceted domain.

Take, for instance, the strategy of network division: here, the entire roadway system is broken down into several sub-networks, with each segment overseen by an individual computational entity. At its surface, this technique appears advantageous due to its built-in parallel capabilities. Yet, it presents a pronounced complication. As an agent transitions from one segment to another, it mandates a dialogue between the corresponding computational entities. Such inter-entity communications, essential for system-wide consistency, can be resource-intensive and time-consuming(Potuzak, 2020).

Esteemed simulation platforms like TRANSIMS, AIMSUN, and Paramics(Nguyen et al., 2021) have gravitated towards this method, though their foundational structures were not natively agent-centric.

Venturing into the territory of inherently agent-based simulation systems, distributing agents over numerous computational entities has become a normative practice. However, this method wrestles with a recurrent hurdle: the indispensable inter-entity updates to maintain a holistic network integrity(Mastio et al., 2018).

MATSim stands out with its trajectory of enhancements. Beyond mere adoption of libraries tailored for concurrent computation(Ma and Fukuda, 2015), the system has seen transformative changes in its core components. The Replanning component, as an example, was fine-tuned to boost its rate of convergence(Zhuge et al., 2021). Furthermore, the simulator's journey marked a shift from a "chronologically driven" model to one steered by events, as evident in the transitions from QSIM to JDEQSIM, eventually leading to HERMES(Horni et al., 2016).

In resonance with these developmental strides, our research brings forth an inventive algorithmic rendition for *MATSim*'s Replanning component, tapping into the virtues of concurrent processing. The primary goal is achieving brisker convergence, thereby streamlining execution durations.

3 ROADWAYS TO PARALLELISM: FROM *MATSim* TO *multiMATSim*

MATSim is an open-source framework, written in Java, dedicated to large-scale agent-based transport simulations(Horni et al., 2016). Stemming from the pioneering works in agent-based traffic modeling, *MATSim* has been continuously improved over the years.

In this simulation framework, each agent represents a virtual entity that can sense, think, and act within the environment. These agents are designed to mimic real-world individuals, ensuring they exhibit human-like behaviors in a transport setting.

To kickstart the simulation, each agent is assigned an initial plan crafted using various data sources. These sources, such as census data, are gathered from the communities of the geographical areas under simulation. This data-driven approach allows for a synthetic population that closely mirrors real-world demographics and behaviors.

As agents navigate the road network based on their initial plans, each plan is evaluated and scored using predefined criteria. This score encompasses elements like travel duration, mode selection, and sequence of activities, reflecting the plan's alignment with the agent's preferences, goals, and limitations.

The simulation process isn't static. As it progresses, a designated subset of agents enters the Replanning phase(Horni et al., 2016). Here, agents, starting from their original plan, create duplicates and introduce modifications based on experiences from previous iterations. This procedure fosters a dynamic and evolving simulation environment, with agents increasingly diversifying their strategies.

The entire simulation operates in cycles, with each iteration representing a 24-hour period. Through these repeated rounds, agents continually refine and diversify their plans, striving for optimal solutions. This iterative approach, combined with the adaptability introduced by the Replanning phase, ensures the system moves closer to desired outcomes as the simulation advances. The user sets the number of iterations through the variable *max*. Although *MATSim* doesn't have an inherent stopping criterion, there are some proposed stopping criteria in the literature based on convergence measures(Horni and Axhausen, 2012). By default, $max = 300$ is often found in some *MATSim* scenarios. Figure 1 illustrates this mechanism.

Operational Modules. After gaining a fundamental understanding of *MATSim*'s operation, it's crucial

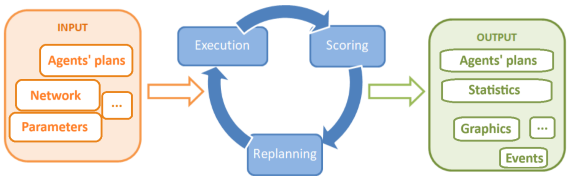

Figure 1: Iterative *MATSim* loop.

to delve into its core components. *MATSim* functions around five foundational elements: Input, Scoring, Execution, Replanning, and Output(Horni et al., 2016). Let's explore each of these in detail :

- **Input.** Essential inputs include the initial plan of each agent and the network file detailing the road and public transport networks specific to the simulated area.
- **Execution Module (Mobsim).** Simulates each agent's daily activities, capturing their movements from one location to another.
- **Scoring Module.** Assesses daily plans, considering the agent's Mobsim performance using a utility function that incorporates movement and activity facets.
- **Replanning Module.** Adapts plan components, like departure times or transportation mode, in response to the traffic conditions based on the plan scores.
- **Output.** Outputs include plan scores, revised plans for agents that experienced Replanning, daily event logs, and statistics in text files and graphical formats.

An overview of the sequence in which *MATSim*'s core modules operate is presented in Algorithm 1. Having elucidated the foundational elements of *MATSim*, it's imperative to focus our attention on the Replanning module. This component serves as a linchpin in our approach, shaping the dynamics and outcomes of the simulations. We will now unpack the intricacies and nuances of the Replanning process.

3.1 Replanning Module in *MATSim*

During the Replanning phase, a fraction R of agents is given the opportunity to alter their existing plans, leading to the generation and subsequent selection of new plans. By default, *MATSim* subjects only 10% of agents to this phase (i.e., $R = 0.1M$, with M the total number of agents in a scenario). However, users possess the flexibility to modify this proportion.

It's crucial to note that an agent's set of plans is not infinite. Here, we opt to denote this value with the variable *nb_plans*. While we frequently encounter it set as $nb\_plans = 5$ in many scenarios, it's important to note that it can be adjusted as needed. As time progresses, plans with the least scores get "overwritten" or replaced. Once agents select a new plan, they are resimulated on the transportation network. The executed plans are then scored, ushering in the generation and selection of fresher plans. The goal is to conduct enough iterations until reaching a point where the plan scores of the agents no longer show improvements, indicating a state of equilibrium.

MATSim offers a range of strategies for the Replanning phase(Horni et al., 2016) :

- **Plan Selector.** Among several strategies available in *MATSim*, we focus on the *ChangeExpBeta* method, which selects plans based on the probability derived from $e^{\Delta_{score}}$, with Δ_{score} denoting the score difference between two plans.
- **Route Innovation.** A strategy that tweaks an existing random plan through re-routing trips. Route decisions are influenced by the traffic conditions observed in the preceding iteration. Multiple routing algorithms, like Dijkstra and A*, may be invoked.
- **Time Innovation.** Adapts a randomly chosen existing plan by shifting all activity end times, either backwards or forwards.
- **Mode Innovation.** Modifies a random existing plan by transitioning to a different transportation mode.

3.1.1 Strategy Weights and Experiments

We'll represent the relative weights of these strategies as $\rho_{PlanSelector}$, $\rho_{reroute}$, ρ_{time}, and $\rho_{transportmode}$.

Every strategy module carries a weight, which dictates the likelihood of that module's action being chosen. To maintain a balanced system, if the cumulative weights of strategy modules don't total one, *MATSim* normalizes them.

Algorithm 1: MATSim algorithm.

```
Start. Choose $max$, $nb\_plans$ and the strategies probabilities
weights such as
$\rho_{PlanSelector} + \rho_{reroute} + \rho_{time} + \rho_{transportmode} = 1$, $R$ ...
Iterate. For $l = 1, \dots, max$
    Run one iteration of MATSim for the $M$ agents: Mobsim,
    Scoring and Replanning only by the portion of $R$ agents to which
    the Replanning applies.
```

Having delineated the weight distribution of strategies within the Replanning module, it is imperative to emphasize that this distribution serves as the foundational framework for the implementation of *multiMATSim*. A detailed exposition of this implementation will be presented in the subsequent section.

3.2 Presentation of *multiMATSim*

Given the previously discussed nuances of *MATSim*, one notable challenge remains its convergence rates in complex transportation systems. Slow convergence inevitably results in prolonged simulation times and heightened computational requirements. In response, we've formulated an approach drawing from the Unite and Conquer methodology, designed specifically to enhance *MATSim*'s convergence efficiency.

In *MATSim*, convergence is the point where each agent's plan scores stabilize, indicating that the simulation has achieved a balanced state regarding the choices of travel plans. Essentially, agents have settled on their optimal routes based on their interactions within the transport system.

Our specific adaptation of the Unite and Conquer approach entails running multiple *MATSim* instances concurrently. Although *MATSim* already employs some multi-threading, we've instituted a deeper level of parallelism that remains largely untapped. These simultaneous instances are consistent in their scenarios and features but diverge in the weights given to their Replanning strategies. Consequently, even when starting with the same set of initial plans, the differing weights lead to varied plans being produced during each Replanning phase across instances.

By distributing the weights differently, we are able to explore a wider range of potential solutions and cover a broader set of possible plan variations.

In addition to this, our approach sets up, every $step$ iterations, a communication between the instances so as to make an exchange of plans between them. After a given iteration, instance i (with $i \in \{1,...,N\}$ and N the number of instances) will know the scores of other instances for a given agent a_k, (with $k \in \{1,...,M\}$ and M the number of agents).

Consequently, instance i will know which instance has the plan with the best score for agent a_k. It can decide to recover it or not, according to a very specific criterion. This criterion is as follows: consider the instance α and the instance β (α, $\beta \in [1;N]$).

For a given agent a_k, instance α gets score S_α^k and instance β gets score S_β^k, with $S_\alpha^k < S_\beta^k$ and $S_\beta^k > S_i^k$ $\forall i \in [1;N]$.

The α instance could then simply retrieve the plan associated with the S_β^k score from β and run the next iteration. However, if each instance retrieves for each agent each time the plan of the instance that made the best score, we would have instances that would all execute the same plan for each of the agents.

The objective is to establish a criterion that enables instances to determine whether it is beneficial to adopt the plan of another instance, even if their own plan has a lower score than that of the other instance for a specific agent.

To develop this plan exchange criterion, we have chosen to focus on the dispersion measure between the scores obtained by different instances for the same agent. By analyzing the dispersion measure, we can indirectly assess the performance of plans and determine whether plan exchanges should be pursued. The criterion ensures that plan exchanges are limited to instances that have obtained the worst scores relative to other instances for each agent.

By employing this criterion, we can leverage the performance of plans as an indirect validation mechanism for plan exchanges. It allows for the possibility of adopting more favorable plans while still maintaining a limit on the exchanges, ensuring that they occur primarily between instances with lower-performing plans.

This criterion is the following: consider the vector V_{S^k} of size N, containing the scores of all the instances for the agent a_k.

If the absolute value of the subtraction of S_α^k (α being the instance concerned by this decision) by S_β^k is greater than the standard deviation of V_{S^k}, then the plan exchange is favorable and α recovers the plan associated with the score S_β^k from the β instance.

The chosen criterion for plan evaluation is preliminary and focuses on the scoring distance between two plans, deeming one more suitable based on its closeness to the best plan. Future research will refine this criterion, with potential machine learning integration. Using an approach inspired by the Monte Carlo method, *MATSim*'s subsequent iterations will prioritize higher-scoring plans. While this method aims for improved plan quality, it's an initial step, and upcoming work will delve into advanced techniques for better plan assessment and selection. Let's summarize the essential parameters listed in Algorithm 2:

- N: choose the number of *MATSim* instances that will run in parallel
- *max*: *MATSim* stops after a predefined number of iterations. It doesn't stop by itself when convergence is obtained.
- M is the number of agents in the current scenario of *MATSim*
- R is subset of agents that undergo the Replanning module
- $step$: Choose a number of iteration steps between which communication will take place between the different instances
- $iter\_processing$: $= max \div step$, this is the number of times the processing and communications are performed.

Algorithm 2: Multi-MATSim

```
**Start.** Choose parameters
**Iterate.** For $l = 1, ..., iter\_processing$, **do in parallel**
    **Computation** Run the $i^{th}$ instance of MATSim for step iterations (Mobsim and Scoring for the $M$ agents, Replanning for the $R$ agents).
    **Communications**
    **Send** the scores and the plans of all $M$ agents to others $N-1$ ranks / instances
    **Receive** the scores and plans of all $M$ agents of the others $N-1$ ranks.
    **Computation** of the best score obtained among the $N$ ranks for each agent, and identification of the associated rank. During this step, the calculation of the standard deviation $std_k$ between the $N$ scores for an agent $a_k \forall k \in [1,M]$ is performed.
    **Communications (exchange plans)**
    • Let agent $a_k$ and its score obtained by the current instance $i = S_i^k$.
    • Consider that the best score among the $N$ scores obtained by all instances for agent $a_k$ is $S_{best}^k$ such as : $S_{best}^k > S_i^k \forall i \in [1,N]$
      **for** each agent $a_k \forall k \in [1,M]$ :
          **if** $|S_{best}^k - S_i^k| > std_k$ **then** get the plan associated to $S_{best}^k$ for the agent $a_k$
```

- Choose the weights for probabilities associated with Replanning strategies. The assigned weights are such that $\rho_{PlanSelector} + \rho_{reroute} + \rho_{time} + \rho_{transportmode} = 1$. Each instance i has a different distribution of weights.
- Set $nb\_plans = 1$. In this way, each instance will focus on optimizing a single plan.

Consider an example with only two *MATSim* instances ($N = 2$) and a single agent ($M = 1$), illustrated on Figure 2. Two *MATSim* instances, i and j, operate on separate computing nodes (x and y). Although these instances share the same scenario, geographical area, and simulated agent, they generate different plans due to distinct Replanning strategies. Here is an explanation of the elements in Figure 2:

- **Run *MATSim* for *step* Iterations.** Each instance runs *MATSim* for a defined number of iterations (*step*), including a 24-hour simulation, plan scoring, and Replanning to generate optimized plans.
- **STOP.** After completing these iterations, *MATSim* instances i and j temporarily halt their execution.
- **Send and Receive Plans.**They exchange the plans they executed and the associated scores for agent a_k. Thus, i sends its plan and its score S_i^k to j, while j does the same with its plan and score S_j^k.
- **Compute S_{best}^k and std_k.** Now, i and j have each other's computational information. They determine which of the two plans achieved the best score, denoted as S_{best}^k (either $S_{\text{best}}^k = S_i^k$ or $S_{\text{best}}^k = S_j^k$).
- The decision is based on calculations involving standard deviation operations (involving std_k) for agent a_k in order to measure the relative distance between the score obtained by the instance in question and the best score. If the difference is significant, the instance opts for the plan with the best score. Otherwise, it retains its own plan.
- These steps repeat for a defined number of *MATSim* iterations (step), ensuring the optimization of each agent's plans.

Of course, in a realistic scenario, there are many more agents, but this example helps to understand how it works for a single agent, and it's the same in a real scenario with a larger number of agents. Similarly, for the instances, we limited it to 2 in this example, but in our experiments, we increased this number.

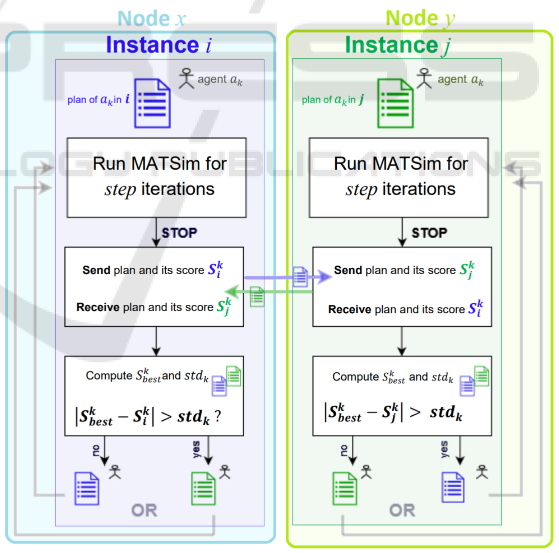
Node x
Instance i
plan of $a_k$in i
agent a_k
Run MATSim for *step* iterations
STOP
Send plan and its score S_i^k
Receive plan and its score S_j^k
Compute $S_{best}^k$and std_k
$|S_{best}^k - S_i^k| > std_k$?
no
yes
OR
Node y
Instance j
plan of $a_k$in j
agent a_k
Run MATSim for *step* iterations
STOP
Send plan and its score S_j^k
Receive plan and its score S_i^k
Compute S_{best}^k and std_k
$|S_{best}^k - S_j^k| > std_k$
no
yes
OR

Figure 2: Illustration of the *multiMATSim* algorithm with two ranks / instances and one agent.

Revisiting the Unite and Conquer approach, it's essential to delineate how it seamlessly integrates with *MATSim* to birth the *multiMATSim* paradigm. The Unite and Conquer method enhances convergence by sharing intermediate solutions between co-methods. In the context of *MATSim*, multiple instances with varied inputs are deployed. While each instance is adept at independently solving the problem, they achieve accelerated convergence by exchanging intermediate solutions. A more rapid average plan score convergence signifies agents efficiently producing higher-quality travel plans as the simulations unfold.

Finally, *MultiMATSim* utilizes distributed computing, allocating one *MATSim* instance to each computing node. Notably, it is also fault-tolerant, allowing it to continue running with the remaining 'active' instances in the event of a software error.

4 METHODOLOGY

In this study, we embark on an exploration of *multiMATSim* using the Los Angeles 0.1% scenario as our foundational ground. Our primary objective is to assess scalability by increasing the number of instances, and consequently, the number of computational nodes. To this end, we will conduct experiments with both $N = 4$ and $N = 8$. While our current experiments with 4 and 8 nodes may not provide a complete assessment of scalability, they serve as an initial exploration and offer valuable insights that set the stage for a more comprehensive analysis as we delve into the results. Additionally, we will introduce a new scenario, still centered on Los Angeles but with a higher agent count: Los Angeles 1%. This scenario engages 191,649 agents, approximately a tenfold increase compared to our initial scenario. Our aim is to evaluate the scalability of our approach as data per node grows, without introducing any unintended artifacts. This comprehensive investigation will provide valuable insights into the performance of *multiMATSim* in scenarios with varying computational demands.

4.1 Technical Specifications

We had the privilege of gaining access to two distinct high-performance computing platforms: Ruche and Fugaku. Ruche(of Paris-Saclay, 2020), a collaborative effort between the computing centers of CentraleSupélec and École Normale Supérieure, features a high-performance computing cluster that became operational in 2020. Ruche is equipped with approximately 3500 processors and includes 8 nodes optimized for GPU computing. Fugaku, on the other hand, stands as a world-renowned supercomputer, recognized for its exceptional computational power. Developed by RIKEN in partnership with Fujitsu, Fugaku is the second most powerful supercomputer globally as of 2023(TOP500, 2023). It is built on the Armv8-A architecture with SVE extensions, delivering unmatched performance across scientific simulations, artificial intelligence, and various research domains. With its A64FX processors and versatile capabilities, Fugaku has made significant contributions to climate modeling, medical research, materials science, and more[2]. Comprising over 150,000 nodes, Fugaku continues to drive groundbreaking advancements in global scientific research(Sato et al., 2020).

Please refer to Table I for node-level specifications for each of the two supercomputers.

4.2 Experimental Protocol

As a reminder, each instance runs on its dedicated computing node with *multiMATSim* method. Moreover, although multithreading is a built-in feature in *MATSim*, we specifically configured it to utilize 20 threads, up from the default 8. Surprisingly, despite having 40 or more available cores on each node, utilizing only 20 threads resulted in the most remarkable performance on both platforms. Several hypotheses could account for this phenomenon: the reduction of resource conflicts, improved utilization of cache memory, or even decreased thread management overhead. This configuration appears to have struck an optimal balance between task parallelism and hardware capabilities. This blend of distributed instances and multithreading highlights our multi-level parallelism approach, leveraging both task and data parallelism to enhance simulation performance. Concerning the variables of *multiMATSim*, we have chosen the following values: $step = 50$, $max = 300$. Concretely, this means there will be a total of 300 iterations, with an exchange every 50 iterations.

5 RESULTS AND DISCUSSION

We provide a comprehensive account of the performance outcomes obtained using *multiMATSim* across two scenarios: 0.1% and 1% agent representation of Los Angeles. Both scenarios underwent tests with 4 and 8 parallel instances. The following results and measured performances were obtained on Ruche cluster. The following observations are illustrated in Figure 3 (and Figure 5, 6) for 0.1% scenario and Figure 4 for 1% scenario. It displays the progression of the average scores of plans executed by all agents in the scenario. The black curve represents the progression in *MATSim* 0.1% / 1% (baseline), while the colored curves depict the various instances of *multiMATSim*. Please note that in Figures 3, 4, 5 and 6 the charts display square peaks resulting from the stop-and-restart procedures in MATSim. After 80% of the $step$ iter-

Table 1: Specifications for a Single Node in Each Supercomputer.

Specification	**Ruche**	**Fugaku**
CPU Reference	Intel Xeon Gold 6230	Fujitsu A64FX
CPU Architecture	x86 (Cascade Lake)	Armv8.2-A SVE 512 bit
Total Cores per Node	40 (2 CPUs, 20 cores/CPU)	48 cores (compute) + 2/4 (OS)
Processor Base Frequence	2.10 GHz	Normal: 2 GHz, Boost: 2.2 GHz
Cache	L1: 32 KB per core (instruction) + 32 KB per core (data) L2: 1 MB per core L3: Up to 27.5 MB (shared)	L1 : 64 KB per core (instruction) + 64 KB per core (data) L2 : 32 MB (8MB per 12-core group) L3: -
SIMD Extensions	AVX-512	SVE (Scalable Vector Extension)

ations, these peaks represent the average of the historical best scores during the last *step* iterations for each agent. However, emphasis should be placed on the overall evolution of the average scores, not these artifacts.

5.1 Results: Scalability

5.1.1 Los Angeles 0.1

- **Total Execution Time.** The time taken for the entirety of the 300 iterations was strikingly similar across configurations. The standard *MATSim* culminated in 8h, *multiMATSim* with 4 instances concluded in 8h40, and the setup with 8 instances wrapped up in about 8h45. Notably, with the 8 instances configuration, the convergence score was swiftly surpassed shortly after the maiden data exchange. For clarification, our goal is not to achieve a shorter total execution time, but rather a faster convergence time.
- **Speedup.** As we had previously presented, *multiMATSim* with 4 instances delivered a remarkable acceleration in convergence. It achieved a convergence score of 105 in just 1.5 hours (iteration 52, right after the first exchange), contrasting starkly with the standard *MATSim* which required 6 hours for the same score (speedup of 4.0). The ultimate anticipation was regarding the performance with 8 instances. Interestingly, the speedup with 8 instances mirrored that of the 4 instances configuration.
- **Comparison Between 4 and 8 Instances.** Referring to Figure 3, an immediate spike in the performance of *multiMATSim* with 8 instances is observable right after the first data exchange at the 50th iteration. As touched upon earlier, our past work showcased the sterling results with the 4 instances setup. The current analysis affirms that while the 4 instances framework maintains its exemplary performance, merely doubling the instances to 8 does not definitively outpace the former configuration. However, given the pronounced leap post the initial exchange at the 50th iteration with 8 instances, an earlier data exchange could potentially have augmented the speedup, making the 8 instances setup potentially more advantageous than its 4 instances counterpart.

5.1.2 Los Angeles 1

- **Total Execution Time.** The 1% scenario, being inherently more data-intensive, demonstrated extended completion durations. The baseline *MATSim* necessitated a substantial 36 hours to complete 300 iterations. On the other hand, *multiMATSim*, across its 4 and 8 instances configurations, hovered around 50 hours. *MATSim* required 29 hours to achieve the convergence score. Remarkably, *multiMATSim* with its 4 and 8 instances setup reached this score in approximately 6 hours.
- **Speedup.** For the 1% scenario, a speedup of 4.8 was observed, slightly surpassing the 4.0 speedup of the 0.1% scenario. This emphasizes the increased efficiency of the *multiMATSim* as the data density grows.
- **Comparison Between 4 and 8 Instances.** As illustrated in Figure 4, the performance dynamics between the 4 and 8 instances setups reveal similar patterns as with the 0.1% scenario. No definitive speedup benefit was witnessed when transitioning from 4 to 8 instances. However, an identical artefact was noticed, with the data exchange occurring at the 50th iteration. Had this exchange been initiated earlier, especially with the 8 instances configuration, a faster convergence might have been realized, primarily with instance 3 exhibiting swifter convergence patterns, mirroring the 0.1% scenario observations.
- **System Stability.** A captivating trend emerged in the 1% scenario. Compared to the 0.1% scenario, the score oscillations were discernibly reduced,

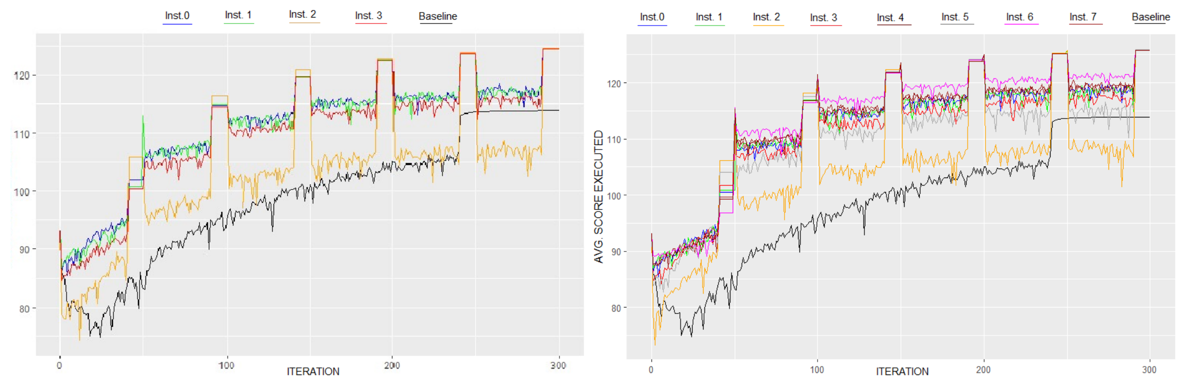

Figure 3: Average Scores of Executed Plans of *multiMATSim* 0.1% across Iterations: 4 and 8 Instances Side by Side (with $step = 50$).

suggesting enhanced system stability when handling an increased agent count.

5.2 Discussion: Scalability and Performance Insights

Our results call for a closer examination of the performance nuances of *multiMATSim* in comparison to standard *MATSim*.

- **Effect of Communication.** One of the foundational strengths of *multiMATSim* lies in the strategic exchange of information between instances. While the impact is clearly demonstrated with 4 instances, the inclusion of 8 instances, although not necessarily yielding superior results in this context, showcases undeniable potential, particularly with the marked score increase following the initial exchange. Had this first exchange occurred earlier, we might have witnessed even better scalability with 8 instances. Such an improvement could be attributed to a broader and quicker exploration of potentialities with 8 instances. Optimizing the granularity, frequency, and timing of these exchanges can be a potential avenue for further refining performance. This may involve sophisticated strategies like adaptive exchange intervals based on observed system performance or even agent-specific data exchanges.
- **Trade-off Between Duration and Convergence.** Although the total execution time may increase more or less, we witness a markedly quicker convergence. Furthermore, the surge in data processed within a single node (via the 1% scenario) attests to the effective scalability of the approach. The heightened resource allocation, paired with data amounts proportional to these resources (during the transition from 4 to 8 nodes with 4 to 8 instances), shows no performance degradation. This hints at a potential for even better scalability with suitable tuning, especially when considering a $step < 50$. This assumption prompted us to delve deeper into experiments by adjusting the $step$ value to assess its impact. The findings can be found in Section C.
- **Influence of Strategy Parametrization on Execution.** The influence of Replanning strategy parametrization on execution duration is significant. Specifically, Instance 2 had an execution time up to 1.5 times longer than its peers. This, in a system with synchronous communications, affects the entire execution of *multiMATSim*. With $nb\_plans = 1$, *planselector* is neutral. Then, lowering its weight amplifies other strategies, slowing execution. However, Instance 2 still outperforms the sequential version in terms of convergence. Possible reconfiguration could enhance its performance.
- **Stark Difference in Total Execution Times.** The variance in total execution times between *MATSim* 0.1% and *multiMATSim* 0.1%, and *MATSim* 1% and *multiMATSim* 1% scenarios demands attention. The disparity can be attributed to the weightage of Replanning strategies, increased number of agents, and the time-consuming processing of XML files in the 1% scenario. A reduction in the frequency of data exchanges might be more beneficial in data-intensive setups.

5.3 Results and Influence of *Step* Value Variation

Upon observing encouraging results with *multiMATSim* at $step = 50$, both for $N = 4$ and $N = 8$, we noticed that the improvements with $N = 8$ were only modest compared to those with $N = 4$, despite dou-

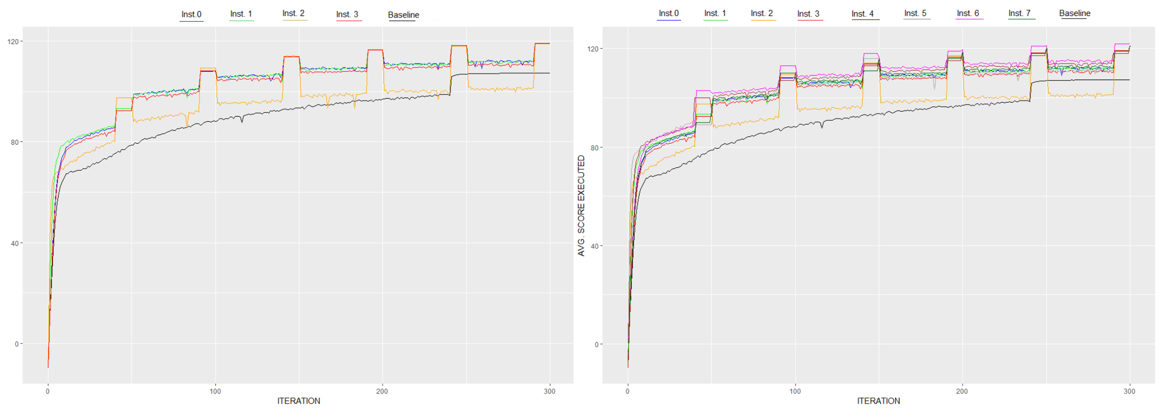

Figure 4: Average Scores of Executed Plans of *multiMATSim* 1% across Iterations: 4 and 8 Instances Side by Side (with $step = 50$).

bling the resources. This observation prompted us to explore a lower *step* value, aiming to assess the impact of earlier exchanges when using a larger number of instances. We thus opted to conduct experiments using $step = 25$ and $N = 8$, enabling exchanges every 25 iterations among the 8 *MATSim* instances. Figure 5 displays the average score of the executed plans in *multiMATSim* for the Los Angeles 0.1% scenario, using $N = 8$ and $step = 25$. Figure 6 shows similar data, but for $N = 4$. While the graphical results for the Los Angeles 1% scenario are not shown here, we noticed the same patterns. The observations are as follows:

Configuration: $step = 25$**; Instances:** $N = 8$ (Figure 5):

- Some exchanges, notably from Instance 6, not only match but even surpass the convergence score of 105 right after the initial exchange.
- Some instances require more iterations after the first exchange to match or exceed this score.
- The convergence score of 105 is achieved within roughly 45 minutes.

In light of the promising results observed with the current setup, we felt compelled to further assess the impact of $step = 25$ while retaining $N = 4$.

Configuration: $step = 25$**; Instances:** $N = 4$ (Figure 6):

- There's no immediate benefit observed after the first exchange.
- However, performance improvements become evident following the second exchange.

Observation Summary. The results underscore the effectiveness of the *multiMATSim* approach in hastening convergence compared to the baseline. The frequency of data exchanges plays a crucial role in performance, yet, even when increasing the number of instances, the added execution time remains minimal. This sets the stage for a deeper analysis in the subsequent sections.

5.3.1 Analysis and Discussion

We observed and analyzed the following three phenomena:

- **Synergistic Effect of Instance Count and Exchange Frequency.** The initial exchange at 25 iterations manifests benefits for $N = 8$ (as seen in Figure 5) but not for $N = 4$ (as per Figure 6). With an augmented number of instances, there's presumably a heightened diversity in generated plans. This could signify that during the preliminary stage, there's an enhanced probability of possessing significantly divergent plans between instances, rendering early exchanges beneficial. With $N = 4$, the diversity might not be adequately expansive for the early exchange to be effective.
- **Exploratory Space Influence.** An expanded exploratory space is available with $N = 8$, and the exchange at 25 iterations could enable the superior dissemination of propitious strategies across this space (as depicted in Figure 5). For $N = 4$ (as seen in Figure 6), the same initial exchange might not infuse ample novelty.
- **Inference.** The apparent lack of benefits with a preliminary 25 iteration exchange for $N = 4$, as opposed to $N = 8$, implies that the utility of exchanges hinges not merely on plan maturity, but also on the diversity of plans accessible for exchange. This underlines the paramountcy of contemplating both the instance count and exchange frequency when optimizing convergence speed and quality.

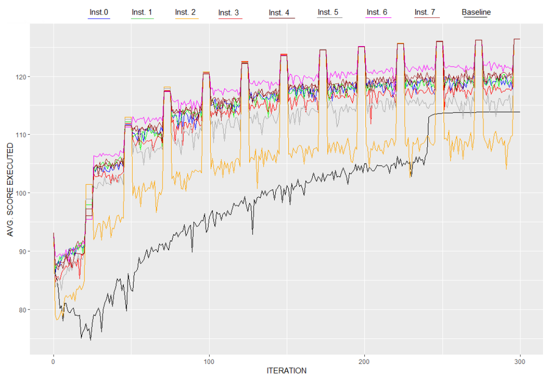

Figure 5: Average Scores of Executed Plans of *multiMATSim* 0.1% across Iterations: 8 Instances with $step = 25$.

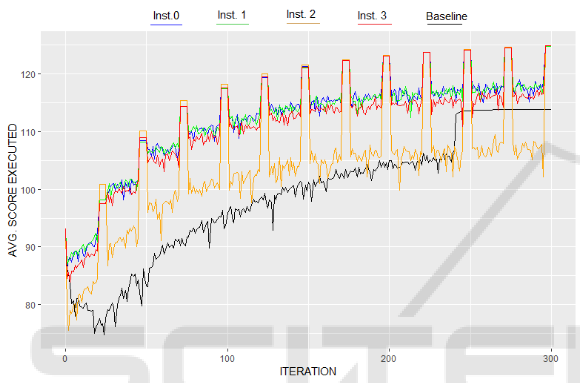

Figure 6: Average Scores of Executed Plans of *multiMATSim* 0.1% across Iterations: 4 Instances with $step = 25$.

5.4 Performance Comparison Between A64FX and Intel Xeon Gold 6230 for *MATSim*

In the following sections, all analyses used $step = 50$. While the results were based on Ruche's nodes, *MATSim*'s performance on Fugaku was markedly slower. Therefore, for the 0.1% scenario, we shifted to comparing the durations of a limited number of iterations. As *MATSim* underpins *multiMATSim*, this clearly had implications for it.

1. **Execution Duration.**
   - *A64FX*. For a typical *MATSim* LA 0.1% iteration, the execution time is 10 minutes.
   - *Intel Xeon Gold 6230*. Under similar conditions, the Intel Xeon Gold 6230 CPU completed an iteration in just 2/3 minutes.
2. **Processing Efficiencies.**
   - *A64FX*. Despite being designed for HPC workloads, efficiency for predominantly sequential programs like *MATSim* is lower, with notable observations of pipeline stalls, particularly for loops with a complex body.
   - *Intel Xeon Gold 6230*. Demonstrating better adaptability for sequential or slightly parallel applications, this architecture exhibited a superior ability to process *MATSim* efficiently.

5.4.1 Results on Fugaku

Although the execution time of *MATSim* was longer on Fugaku, we still wanted to verify our approach. We launched *multiMATSim* LA 0.1% $\times$ 4 with a reduced number of iterations (60), exchanging at the 50th iteration. The behavior was consistent with that on Ruche, with a convergence that seemed faster compared to *MATSim* LA 0.1%. It took about 8 hours on Fugaku to reach the score of 105 with *multiMATSim* LA 0.1%, while it took about 30 hours with baseline *MATSim* LA 0.1%.

5.4.2 Architectural Differences

The profiler revealed differing behaviors of the JVM, with JVM settings varying between the CPUs. The number of calls to the garbage collector remained the same between the two CPUs, but the GC intervention times were much longer on the A64FX.This might simply be due to the architectural differences between the two CPUs and may not be related to the difference in performance. Other differences were noted, such as thread management or memory management, which can be explained in the same manner.

5.5 Discussion on Performance Differences

The observed performance variations between the A64FX and Intel Xeon Gold 6230 architectures when executing *MATSim* are intriguing. These differences can be elucidated by examining the interplay of architectural attributes and *MATSim*'s inherent software characteristics.

5.5.1 Sequential Nature of *MATSim*

MATSim's design leans heavily towards sequential execution, with only intermittent multithreading on specific modules. Such a design paradigm is intrinsically reliant on stable and robust CPU performance.

- The A64FX's compact core structure and diminished resources for out-of-order optimization are not optimally suited for applications that are chiefly sequential.

- *MATSim*'s inability to exploit SIMD instructions means that SVE functionalities within the A64FX are not leveraged.
- The longer pipeline structure of A64FX, prone to stalls for loops with intricate bodies, may act as a bottleneck for *MATSim*, which is replete with complex looped structures.

5.5.2 Java and *MATSim*

It is possible that the JVM operates different optimizations for the two distinct CPU architectures. Some studies(Poenaru et al., 2021)(Jackson et al., 2020) have revealed that the performance of A64FX could be greatly enhanced through the use of libraries specifically compiled by Fujitsu. However, the only available version of OpenJDK compiled by Fujitsu is OpenJDK 11, which is relatively old. For instance, during an iteration of *MATSim* on Los Angeles 0.1% with Fugaku, the execution time reduces from approximately 15 minutes using OpenJDK 11 (compiled by Fujitsu) to 10 minutes using OpenJDK 17 (compiled by GCC). The improvement can be partly attributed to the fact that OpenJDK version 17 incorporates specific optimizations for ARM processors. It is plausible that a version of OpenJDK 17 or later, compiled by Fujitsu, could yield even better results.

5.5.3 Intel Xeon Gold 6230's Affinity

Considering the Intel Xeon Gold 6230 architecture:

- Its potential for a more generous die-size provides enhanced resources for out-of-order executions, which can be a boon for predominantly sequential tasks such as *MATSim*.
- This architecture's universalistic approach might encompass optimizations that harmoniously cater to both sequential and sporadically parallel tasks.

5.5.4 Concluding Thoughts

While the A64FX boasts commendable prowess, particularly in memory-focused benchmarks and power efficiency, its design principle appears to be skewed towards applications that are intensively parallel. In contrast, the Intel Xeon Gold 6230, possibly equipped with larger core sizes and generalized optimizations, seems more in tune with *MATSim*'s operational patterns.

The insights gathered suggest that harnessing *MATSim*'s potential on architectures similar to A64FX would entail a significant overhaul to maximize its parallel processing proficiencies. Conversely, future computational architectures might need to strike a balanced chord, catering seamlessly to both parallel and sequential workflows for holistic efficiency.

6 CONCLUSION

In our investigation of the *multiMATSim* method, the outcomes regarding scalability have been heartening. Although limiting our tests to 4 and 8 nodes doesn't capture the full essence of scalability, the data offers informative and optimistic views on how *multiMATSim* responds with an increasing number of nodes. These initial outcomes lay the groundwork for deeper dives into how our method scales as we enhance computational resources. Additionally, as the per-node load grows, we noticed a reliable enhancement in performance. Augmenting the computational nodes while simultaneously upping the *MATSim* instances for *multiMATSim* maintains steady performance. Given the pronounced improvements in average plan scores, it's plausible that initiating exchanges earlier might yield a pronounced acceleration. Such outcomes leave us optimistic about achieving better results with heightened load, alluding to the potential of augmented horizontal and vertical scalability. For our forthcoming experiments, the LA 10% scenario emerges as a favorable choice, pressing our system's limits with additional instances, nodes, and optimized parameters. On a distinct note, while running our framework on two varied CPU architectures - x86 and ARM, we detected subtle behavioral variations. Although the ISA (Instruction Set Architecture) may not be the sole determinant, the differences in core sizes between these CPUs probably have a role. In this context, our intention is to delve into Amazon's Graviton 3, an ARM-centric CPU boasting larger cores. It's pertinent to mention that the A64FX, which is tailored for intensive parallel tasks using innovations like SVE or HBM memory, might not reach its full potential with a primarily sequential tool like *MATSim*. Given the fact that *MATSim* is optimized more for the x86 architecture and the opportunities presented by a JVM library recently launched by Fujitsu, there's an open avenue for additional inquiry. Still, in terms of convergence velocity on Fugaku, *multiMATSim* manages to surpass *MATSim*. Our grand vision is to gauge the universality of this Unite and Conquer strategy on alternative multi-agent traffic simulators, such as SUMO(Alvarez Lopez et al., 2018) or POLARIS(Auld et al., 2016), underscoring its extensive relevance.

REFERENCES

Alvarez Lopez, P., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wießner, E. (2018). Microscopic traffic simulation using sumo. In *IEEE Intelligent Transportation Systems Conference (ITSC)*.

Auld, J., Hope, M., Ley, H., Sokolov, V., Xu, B., and Zhang, K. (2016). Polaris: Agent-based modeling framework development and implementation for integrated travel demand and network and operations simulations. *Transportation Research Part C: Emerging Technologies*.

Emad, N. and Petiton, S. (2016). Unite and conquer approach for high scale numerical computing. *International Journal of Computational Science and Engineering*, 14:5–14. hal-01609342.

Horni, A. and Axhausen, K. (2012). Matsim agent heterogeneity and a one-week scenario. Technical Report 836, Institute for Transport Planning and Systems (IVT), ETH Zurich, Zurich.

Horni, A., Nagel, K., and Axhausen, K. W. (2016). *Introducing MATSim*, chapter Introducing MATSim, pages 3–8. Ubiquity Press, London. License: CC-BY 4.0.

Jackson, A., Weiland, M., Brown, N., Turner, A., and Parsons, M. (2020). Investigating applications on the a64fx. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 549–558, Los Alamitos, CA, USA. IEEE Computer Society.

Ma, Z. and Fukuda, M. (2015). A multi-agent spatial simulation library for parallelizing transport simulations. In *2015 Winter Simulation Conference (WSC)*, pages 115–126.

Mastio, M., Zargayouna, M., Scémama, G., and Rana, O. (2018). Distributed agent-based traffic simulations. *IEEE Intelligent Transportation Systems Magazine*, 10.

Nguyen, J., Powers, S. T., Urquhart, N., Farrenkopf, T., and Guckert, M. (2021). An overview of agent-based traffic simulators. *Transportation Research Interdisciplinary Perspectives*, 12:100486.

of Paris-Saclay, U. (2020). Mésocentre de l'université paris-saclay. Accessed on: August 17, 2023.

Poenaru, A., Deakin, T., McIntosh-Smith, S., Hammond, S., and Younge, A. (2021). An evaluation of the fujitsu a64fx for hpc applications. In *Cray User Group 2021*. Cray User Group 2021 ; Conference date: 03-05-2021 Through 05-05-2021.

Potuzak, T. (2020). Reduction of inter-process communication in distributed simulation of road traffic. In *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–10.

Sato, M. et al. (2020). Co-design for a64fx manycore processor and "fugaku". In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, Atlanta, GA, USA.

TOP500 (2023). The list: June 2023. Accessed on: August 17, 2023.

Zhuge, C., Bithell, M., Shao, C., Li, X., and Gao, J. (2021). An improvement in matsim computing time for large-scale travel behaviour microsimulation. *Transportation*, 48.