Optimistic Parallel Simulation of Tightly Coupled Agents in Continuous Time

Philipp Andelfinger and Adelinde Uhrmacher
Institute for Visual and Analytic Computing
University of Rostock, Germany
{philipp.andelfinger, adelinde.uhrmacher}@uni-rostock.de

***Abstract*—Agent-based simulations relying on synchronous state updates using a fixed time step size are considered attractive candidates for parallel execution in order to reduce simulation running times for large and complex scenarios. However, if the underlying models are formulated with respect to continuous time, a time-stepped execution may only approximate the strict model semantics. To simulate continuous-time agent-based models, parallel discrete event algorithms can be applied. Traditionally those are based on logical processes exchanging time-stamped events, which clashes with the properties of models in which tightly coupled agents frequently access each other's states. To illustrate the challenges of such models and to derive a solution, we consider the domain-specific modeling language ML3, which allows modelers to succinctly express transitions and interactions of linked agents based on a continuous-time Markov chain (CTMC) semantics. We propose an optimistic synchronization scheme tailored towards simulations of fine-grained interactions among tightly coupled agents in highly dynamic topologies. By restricting the progress per round to at most one state change per agent, the synchronization scheme enables efficient direct read and write accesses among agents. To maintain concurrency given actions that depend on dynamically updated macro-level properties, we introduce a simple relaxation scheme with guaranteed error bounds. Using an extended variant of the classical susceptible-infected-recovered network model, we demonstrate that the proposed synchronization scheme accelerates simulations even under challenging model configurations.**

***Index Terms*—parallel simulation, agent-based simulation, multi-agent simulation, optimistic synchronization**

I. Introduction

Agent-based simulation is an established method to study systems of interacting entities in domains such as sociology [1], traffic engineering [2], or systems biology [3]. A simple approach to the time advancement in agent-based simulations is taken by synchronous time-driven simulations, in which all agents' state transitions occur concurrently at discrete points in logical time. This approach provides ample opportunities for parallel execution: since the agents' transitions at a given time step are logically concurrent, the transitions can be assigned to different processing elements to reduce execution times. However, in many real-world systems, transitions may occur at arbitrary points in time [4]. If agent-based models are thus specified with respect to continuous time, an execution using the limited granularity given by a fixed time step size may cause deviations from an execution according to the strict model semantics. Further, conflicts may occur among the actions taken by agents within a time step [5]. To resolve conflicts, a time-driven simulation cannot rely on the precedence relation defined by the fine-grained transition times available in a continuous-time simulation. Further, the time step size defines a lower bound on the propagation delay of effects throughout the simulation [6], whereas in a continuous-time formulation, no such bound must be imposed.

On the other hand, while a continuous-time execution can represent the model semantics to machine precision, when considering parallelization to reduce execution times, a synchronization scheme is required to satisfy the resulting stricter ordering constraints. A variety of parallel discrete-event simulation (PDES) algorithms have been developed to ensure the efficient and correct execution of simulation models oriented typically around logical processes that exchange events specified in continuous logical time [7].

A challenge to an efficient execution using PDES is given by the tightly interlinked life courses of communities of agents encountered in many agent-based models [8]. To cater to this "tight coupling", multi-agent modeling and simulation environments such as Repast [9], Netlogo [10], or Mesa [11] allow agents to directly carry out read and write accesses to the attributes of other agents.

To illustrate the challenges of parallel simulation of tightly coupled agent-based models and to derive a solution, we consider the domain-specific modeling language ML3 [12], which allows modelers to succinctly express transitions and interactions of linked agents based on a continuous-time Markov chain (CTMC) semantics. We highlight key properties of ML3 models to determine the requirements for an efficient parallel execution and present a novel optimistic synchronization algorithm. In optimistic PDES, some computations are carried out speculatively without regard for potential violations of ordering constraints. If a violation occurs, the affected simulation state is rolled back to a previous correct state. To cope with the tight coupling among separate agents' states, the algorithm follows two main ideas. Firstly, agents may directly read and write other agents' states, without the exchange of events or messages as used in traditional PDES approaches. Secondly, a synchronous mode of time advancement restricts both the temporal deviation among processors and the rollback overhead. The proposed algorithm is evaluated using a variant

978-1-6654-3326-6/21/$31.00 ©2021 IEEE

of a classical epidemic network model [13], which we extend to stress the performance-critical aspects of ML3 models.

We summarize our contributions as follows:

- We describe performance-critical aspects of ML3 models and their implications for parallelization.
- We propose and detail a synchronous optimistic execution scheme for models of tightly coupled agents.
- Performance measurements under challenging model configurations using scenarios populated by up to 2^{26} agents demonstrate a speedup of up to 5.5 on 16 cores compared to an efficient sequential baseline.

The paper is structured as follows: in Section II, we describe key properties of the considered class of models based and their implications for parallel execution. In Section III, we propose an optimistic synchronization scheme tailored to the identified requirements. In Section IV, we describe our implementation of the algorithm and our sequential baseline. In Section V, we evaluate the performance of the algorithm. In Section VI, we discuss related work in optimistically synchronized discrete-event simulation and agent-based simulation. Section VII summarizes our results and concludes the paper.

II. Analysis

In ML3, the model behavior is specified using rules that define the possible agent state transitions in terms of their conditions, timing, and effects. Conditions act as guards to decide in which situation a transition may occur. Transitions for which all conditions are satisfied take place after a waiting time specified as a deterministic or stochastic delay in logical time. As a distinguishing feature of ML3, waiting times are defined in the form of rates of an underlying Continuous Time Markov Chain (CTMC). For instance, the following rule applies to agents with an incoming above 50000 units, and the associated transition increments the variable v at a rate of r:

```
Agent
  | ego.income > 50000 // guard expression
  @ r                  // waiting time expression
  -> v := v + 1        // effect
```

The specific times at which such *rate-driven* transitions occur are determined according to pseudo-random numbers drawn from exponential distributions. A transition occurring at an agent may instantaneously modify other transitions' rates at the current agent or other agents, which requires a reconsideration of future transition firing times. This model of execution is inspired by stochastic simulation algorithms of which key variants were originally proposed by Gillespie [14] to simulate biochemical reaction networks.

Two specific algorithms in this category include the *Direct Method* [14] and the *Next Reaction Method* [15]. The Direct Method and its variants select the next transition directly based on the current transition rates: similarly to the generation of pseudo-random numbers adhering to an empirical distribution, a uniform random variate determines the index of the next transition depending on their relative rates. A second uniform random variate is transformed according to the sum of all transition rates to generate the transition time. As each selection of a transition and its time requires the transition rates to be current, the Direct Method suggests a serialized mode of execution. As an alternative, the Next Reaction Method draws tentative timestamps for all possible transitions. The transition with the earliest timestamp is carried out, which may affect the rates of other transitions. Such dependent transitions are then rescheduled according to the updated rates. The Next Reaction Method may discard many tentatively scheduled transitions if the degree of coupling among agents is high. However, the scheduled events provide valuable information of the time and agent assignment of future transitions assuming independence of the transition rates. Since this information permits a speculative parallel execution of transitions, we rely on the Next Reaction Method throughout the paper.

The execution of ML3 models using the Next Reaction Method can be viewed as a special case of discrete-event simulation, which may suggest the use of well-established methods for parallelization. However, we identify three properties of ML3 models and, more generally, tightly coupled agent-based models that pose challenges for traditional PDES approaches:

1) Direct read and write accesses across agent boundaries to support tightly coupled life courses of agents,
2) State-dependent creation and removal of links and agents to account for the dynamic structure of systems,
3) Global access to the state of a population of agents to calculate macro-level properties which may influence the agent behavior at the micro level [16].

We briefly illustrate the occurrence and implications of each of these properties in turn, relying on an ML3 formulation of a migration model [12]. The model, which represents decision-making processes involved in migrations from Senegal to France, was originally developed in Netlogo [17]. The model excerpts shown are slightly simplified for brevity.

Direct state access across agent boundaries: as do other agent-based modeling and simulation environments, ML3 permits read and write accesses to arbitrary agents as part of the guard expressions, waiting time expressions, and effects that make up a rule. The following excerpt is part of the "effect" component of a rule in the migration model:

```
(ego.friends + ego.friends.collect(alter.friends) -
 ego.familyMembers() - ego).filter(ego.canMarry(alter))
```

The expression filters an agent's friends as well as its second-degree friends according to the predicate function `canMarry()`, which in turn accesses the agents' attributes. The accesses occur instantaneously during the transition. In a scenario in which an agent has exactly ten direct friends, each evaluation of the expression requires accesses to up to one hundred friends and friends-of-friends. Considering the execution of such a transition in a parallel simulation, we note that the agent must be able to access the other agents' states *at the logical time of the transition*. This can be achieved by maintaining a history of previous states, rolling back agents to previous states when required by write accesses, e.g., using the classical Time Warp algorithm for optimistic PDES [18].

However, the Time Warp algorithm assumes that the interactions among simulation objects occur through the exchange of events in a message-passing style, which would require a "read request" and "read response" event to carry out a single read access between two agents. Further, the two events involved in a read access would carry the same timestamp as the current transition. The resulting tight temporal coupling among the involved logical processes runs counter to the asynchronous mode of execution defined by the Time Warp algorithm.

State-dependent creation and removal of links: at a transition, an agent may create and remove links based on its own state, the state of other agents, or randomly. In the following excerpt from the migration model, an agent moving to a new address creates links to the current inhabitants of the address as well as the inhabitants of the neighboring addresses:

```
ego.friends += ?address.inhabitants +
               ?address.neighbors.collect(
                           alter.inhabitants) - [ego]
```

The resulting topology evolves dynamically over the course of the simulation, which has two key consequences for parallelization: firstly, if the simulation state is partitioned into logical processes, frequent repartitioning is required to minimize attribute accesses across process boundaries. In the presence of randomized link creations, the proportion of accesses across logical processes may be large even with frequent repartitioning. Secondly, even if a specific rule restricts its accesses to direct neighbors of an agent a_0, topology information cannot easily be exploited to determine independent transitions, as a concurrent transition of an agent a_1 may create or remove a link to a_1, which would invalidate the topology information.

Globally accessing the state of a population of agents: state variables of entire populations of agents may be accessed by an agent. As an example, in the migration model, a migrant randomly selects a new address from a global set of all addresses, filtering by country and current inhabitation:

```
Address.all.filter(alter.location = "host country" &&
                   !alter.hasInhabitants()).random()
```

Each of the addresses is represented as an agent. For this expression to be evaluated correctly, all previous updates to the set of addresses must be visible. More generally, by imposing ordering constraints among transitions at separate agents, the presence of accessing globally entire populations of agents can severely limit the concurrency among transitions. When strictly adhering to the model semantics, the extreme case of this type of global access at every transition implies a complete serialization of the simulation.

Overall, while the rate-based transition times in ML3 models necessitate the use of an optimistic approach to synchronization, the tight and difficult-to-predict coupling among agents suggests a scheme that emphasizes the efficiency of attribute accesses and limits the frequency and cost of rollbacks.

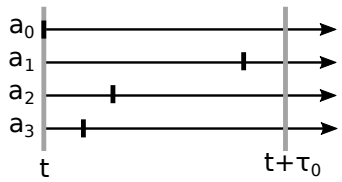
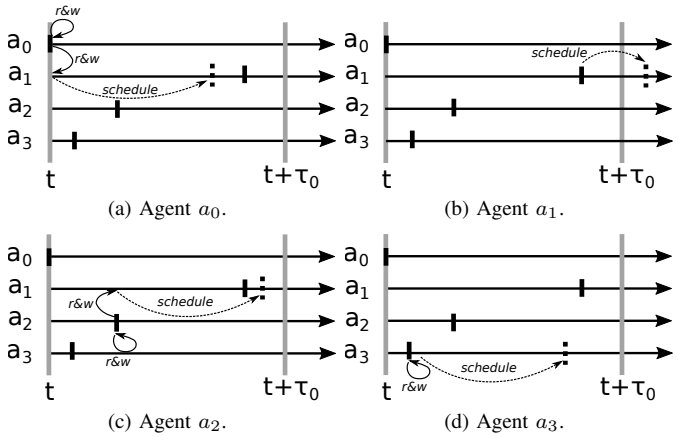


Fig. 1. Example of the transitions (vertical lines) of four agents a_i scheduled after a global synchronization point at logical time t. Initially, the event horizon limiting the speculative execution of transitions is set to $t + \tau_0$.



(a) Agent a_0. (b) Agent a_1.

(c) Agent a_2. (d) Agent a_3.

Fig. 2. Scheduled transitions and their effects. When executed in parallel, agents may carry out their transitions in any order. Our synchronization scheme guarantees that of the accesses to a_1 caused by the transitions of a_0, a_1, a_2, only the earliest access takes effect and is committed.

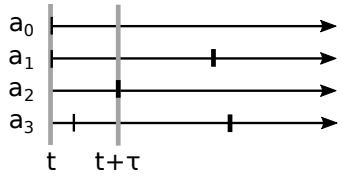


Fig. 3. Final scheduled transitions (wide lines) and committed agent accesses (thin lines) at the end of the round. Throughout the processing of transitions, the effective event horizon $t+\tau$ has been gradually reduced to guarantee that at most one access per agent is committed. In the example, the transitions at agents a_0 and a_3 and the associated write accesses are committed, whereas the transitions of agents a_1 and a_2 are deferred to future rounds.

III. Speculative Synchronization Scheme

Based on the properties of ML3 models detailed in the previous section, our central idea is to acknowledge that the considered models represent tightly coupled systems in which a typical transition directly and instantaneously affects several agents scattered throughout the topology, and in which the propagation of effects over time can be swift and difficult to predict. As a consequence, we propose a synchronous scheme that, instead of emphasizing the maximization of parallelism, aims to avoid some of the overheads for state accesses and rollbacks that may be incurred by more aggressive speculative schemes. Our synchronization scheme is based on the simple observation that starting from a global synchronization point, the earliest access in logical time at an agent can never be invalidated by another access to the same agent.

A. Overview

The simulation takes a round-based approach, wherein we ensure that at the end of each round, only the earliest access at each agent is committed. Hence, the delta in logical time by which the simulation can advance throughout a round depends on the interactions among the agents. We illustrate the desired behavior of the round-based execution on the example of simulation populated by four agents (cf. Figure 1). The current simulation time t is the timestamp of the earliest scheduled transition. During the current round, we consider all transitions up to $t+\tau_0$ for execution, where τ_0 is a tunable initial window size in logical time. Figure 2 shows the operations associated with the agents' transitions. For simplicity, we assume that all attribute accesses take place in read and write mode. While the operations of agents assigned to a single thread occur in order of ascending timestamps, the order of operations across threads is arbitrary. Importantly, multiple transitions executed speculatively within the round may access the same agent. Finally, Figure 3 shows the desired outcome of the round: the final window size τ guarantees that we commit exactly those transitions whose accesses occurred earliest in logical time for all accessed agents, along with any involved write accesses. Further, the scheduling of any events speculatively scheduled by the newly committed transitions takes effect.

B. Mechanism

Having illustrated the intended outcome of a synchronization round, we now describe the mechanisms by which this outcome is achieved. Algorithm 1 shows the main loop of our synchronization scheme as pseudo code. We assume that agents are assigned to threads by a static partitioning scheme, e.g., randomly. Each round of simulation begins with a global reduction to determine the minimum timestamp among all threads' scheduled transitions. The initial upper bound on the timestamp of transitions considered for execution, the *event horizon* is initialized based on the minimum timestamp and the tunable initial window size τ_0. Our choice of τ_0 is discussed in Section V. Each thread populates a list `window_transitions` holding all local transitions scheduled for execution before the initial event horizon (line 4). Subsequently, each thread executes the transitions in the list in order of ascending timestamp. The timestamp order of execution is exploited in line 6 to terminate the execution of transitions early if the event horizon does not permit any further progress in the current round.

As will be discussed below, the agent accesses carried out during transitions reduce the event horizon to guarantee that only at most one access at each agent is committed. While executing transitions, each thread populates an *access list* of agents accessed by the local agents, which may include agents assigned to other threads. By storing the agents at the thread the access originated from, the need for locking the access lists is avoided.

A global barrier ensures that the final event horizon of the current round has been determined (line 9). Now, all transactions with timestamps lower than the event horizon are committed, and new transitions scheduled in the process are enqueued. Agents whose transitions occurred at or after the event horizon are rolled back. Similarly, agent accesses that occurred earlier than the event horizon are committed and all other accesses are rolled back. To identify accessed agents, each thread iterates through all the threads' access lists. As the access lists are not written to, no locking is required. Once all threads have reached the final barrier, the simulation has advanced to the event horizon and may enter a new round.

In the pseudo code of Algorithm 1, the updates of the event horizon within a round were implicit. Algorithm 2 shows this aspect in detail: on an agent access, a per-agent mutex is obtained. If the new access arrives at the earliest observed logical time, the access is carried out, displacing a prior access

Algorithm 1: Main speculative parallel simulation loop.

```
1  while !termination_criterion do
2      event_horizon ← get_global_min_timestamp() + τ0
3      foreach thread in parallel do
4          window_transitions ← "local events before event_horizon"
5          foreach event in window_transitions do
6              if event.timestamp ≥ event_horizon then
7                  break
8              event.execute()
9          barrier()
10         foreach event in window_transitions do
11             if event.agent.earliest_access < event_horizon then
12                 commit transition, enqueue new events
13             else
14                 roll agent back to previous state
15         foreach list in access_lists do
16             foreach agent in list do
17                 if agent.is_local() and
18                 agent.earliest_change < event_horizon then
19                     commit transition, enqueue new events
20                 else
21                     roll agent back to previous state
22         barrier()
```

Algorithm 2: Wrapper for agent accesses.

```
1  procedure Agent::try_access(now):
2      lock(mutex)
3      if now < earliest_access then
           // access is earliest in round so far
4          if earliest_access ≠ ∞ then
5              self ← old_self // roll back previous access
               // defer transition associated with
               // previous access to future round
6              atomic_min(&event_horizon, earliest_access)
7          perform_access()
8          earliest_access ← now
9      else
           // access not earliest in round,
           // defer associated transition
10         atomic_min(&event_horizon, now)
11     unlock(mutex)
```

with larger timestamp, if any. Otherwise, the new access fails and the event horizon is updated to defer the new access to a future round, which globally restricts the processing of transitions at all threads.

We point out the close relationship of our synchronization scheme to the conflict resolution mechanisms used in time-driven agent-based simulations, in which all agents concurrently advance their states from logical time t to $t + \tau$. A number of approaches have been proposed and evaluated to resolve the potential conflicts that may emerge when accessing limited resources, e.g., if multiple agents move to the same location in the simulation space [5], [19], [20]. In a "push" approach, agents register their desired state accesses at the simulation objects, potentially displacing previously registered agents based on static or dynamic priorities. At the end of a round, the highest-priority registered agent gains access to the object. This process repeats until all agents have gained access to an object or given up, at which point logical time is increased to $t+\tau$. Both in this procedure and in our optimistic synchronization scheme, agents concurrently attempt to access other simulation objects, potentially displacing each other. In both cases, a round concludes once at most one access per entity can be committed, deferring displaced agent accesses to a future round. The key difference between conflict resolution in time-driven simulations and our synchronization scheme lies in the definition of the access priorities: in the considered class of agent models, the timestamps in continuous time associated with transitions can be interpreted as priorities prescribing an access order. In contrast, given a model that assumes concurrent transitions of all agents at fixed time steps, priorities within each time step must be defined separately based either on model properties or according to some other criterion, e.g., based on pseudo-randomness [5].

C. Global State Access

We briefly describe our simple mechanism to support globally accessing state variables of all agents to calculate a macro property of the system, e.g., the number of agents being in a specific state. Let us consider an extreme case in which all agents' transition rates depend on a macro property that is updated during every transition. In addition to requiring a rescheduling at all agents after every transition, every transition depends on the transition prior to it, resulting in a complete serialization of the simulation. Thus, as our intention is to execute transitions in parallel, a strict adherence to these global updates is infeasible. Instead, we approximate such global dependencies by notifying dependent agents only once the variable has changed by a configurable amount.

In the sequential case, this type of approximation is trivially implemented by storing the value of the global variable during the last notification and triggering new notifications once the change exceeds the threshold. However, during speculative parallel execution, the threads alter the global variable in an unpredictable order and may roll back prior changes. To identify the exact point in time at which the threshold is crossed, we maintain a variable `abs_change` holding the sum of the *absolute* changes to the global variable, independent of their sign. This variable provides an upper bound on the amount of change of the global variable in either positive or negative direction. During a parallel simulation round as described previously, `abs_change` is updated atomically by all threads. We terminate the round at the point $t_{\text{threshold}}$ in logical time when `abs_change` crosses the configured change threshold, which is a potential point in time at which the actual change threshold may have been crossed. After committing and rolling back transitions, the value of the global variable is consistent with the sequential simulation at $t_{\text{threshold}}$. We can now check whether the configured threshold has in fact been crossed by the transition that terminated the round, and trigger notifications if needed. Finally, `abs_change` is set to the actual change and the next round of simulation commences. This simple scheme guarantees notifications at exactly the same points in logical time as in the sequential simulation. In Section V, we explore the sequential and parallel simulation performance in the presence of an approximately updated global variable that represents a macro property on which the micro behavior of the agent-based model depends.

IV. Implementation

Our starting point is a sequential C++ simulator implementation created from scratch. The simulator employs Gillespie's *Next Reaction Method*: an event is scheduled for each rule whose conditions are satisfied, according to the current transition rate. The simulation advances by executing the transition associated with the earliest scheduled event, which may entail updates to zero or more conditions or rates and subsequent (re-)scheduling of events.

We implemented two mechanisms to avoid executing events that have been retracted in order to reschedule transitions: 1. removal from the pending event list, and 2. skipping based on an agent attribute. The first mechanism requires a data structure that permits efficient element removal. We employed the `set` container from the C++ standard template library to represent the pending event list, which is implemented as a red-black-tree and allows for logarithmic-time insertion and removal. In the second mechanism, events are stored in an STL `priority_queue` container, which internally relies on an implicit heap and in our experiments achieved vastly faster element insertion, but does not support efficient element removal. Each agent possesses one attribute per rule that holds the timestamp of the next transition. Whenever an event is scheduled or retracted, the corresponding attribute is updated. Thus, when considering an event for execution, the simulator can now compare the timestamp of the earliest event with the timestamp stored at the corresponding agent to decide whether the event has been retracted and should thus be skipped. With this second approach, an explicit removal of events is avoided, at the cost of retaining many of the retracted events in the list. Our experiments rely on the second mechanism, which during initial tests consistently outperformed explicit event removal.

The agent states are stored in two arrays, one holding the "current" states at logical time t, i.e., the lower bound of

the current window, and one holding the projected states to be determined during the current round. At the end of a round, the projected states are either rolled back to time t, i.e., discarded, committed by being copied to form the new "current" states at the lower bound of the next window. As described in Section III, during an agent access, the target agent may be rolled back. By restricting the state history to only the state at t, rolling back an agent identified by `agent_id` involves only the trivial and inexpensive assignment:

```
*this = previous_states[agent_id];
```

To achieve deterministic results, each agent draws pseudo-random numbers from a separate random number stream generated by the Xoroshiro128** generator [21], which passes the BigCrush test suite from the TestU01 library [22]. Since rollbacks require copying the previous random number generator state as part of the overall agent state, both the memory consumption and execution time of the simulation benefit from the generator's comparatively small state size of 128 bits.

For multi-threading, we employ POSIX threads and the associated facilities for barriers and mutual exclusion. Threads are pinned to physical CPU cores to improve cache usage and to avoid unnecessary non-uniform memory accesses when using fewer than the available number of cores. For atomic access to the shared upper bound on the current window (`event_horizon`), we employ the atomic operations library from the C++ standard template library.

V. Experiments

A. Setup

To evaluate the benefits and limitations of the proposed synchronization scheme, we constructed a simulation model that emphasizes the hard-to-parallelize properties of tightly coupled agent-based models as detailed in Section II. The model is based on an agent-based formulation of the classical susceptible-infected-recovered model as described by Macal [23], to which we introduce rate-based transition probabilities in continuous time: in place of the per-step transition probabilities of the original time-driven model, transition times are drawn from an exponential distribution according to rates dynamically updated based on the neighboring agents' states. An agent is infected by its neighbors at a rate equal to its number of infected neighbors. Recovery from an infection and the return to the susceptible state take place at a rate of 1.

Initially, each agent creates mutual links with 8 unique neighbors chosen uniformly at random. To exercise the capability to dynamically track inter-agent dependencies, the agents randomly move within the topology by cutting the ties to their current neighbors and selecting a new set of 8 random neighbors. The movement rate can be configured to depend on the overall number of infected agents, which represents the challenging case of a macro property changing when agents are newly infected or recovered. Given the proportion p of infected agents, an agent moves at a rate of $0.1 \times (1 - p)$. If the macro property is not taken into account, the movement rate is constant at 0.1.

Our experiments were conducted on a system equipped with two 16-core Intel Xeon E5-2683v4 CPUs and 256GiB of RAM, running CentOS Linux 7.9.2009. Hyperthreading was disabled. We plot averages of at least 5 runs for each data point with 95% confidence intervals. Where not otherwise stated, the simulation end time was scaled inversely with the number of agents to achieve tolerable execution times even for large populations. Sequential simulation runs showed that the events in the simulation are fine-grained, with processing times per transition of 2 to 10μs including all overheads.

The correctness of the parallelized implementation was verified by direct comparison of agent states and timestamps of scheduled transitions at the end of the simulation, which were observed to be identical between the sequential and parallel simulations. In the parallel simulations, agents are assigned to threads by an ascending identifier. Since links among agents are created and changed uniformly at random, this is equivalent to a static random partitioning.

B. Results

In our synchronization scheme, the maximum progress per round depends on the initial window size τ_0. The choice of τ_0 involves a model-specific tradeoff between the opportunity for parallel processing on the one hand, and the overhead of executing events that are subsequently rolled back on the other hand. To account for the simulation conditions at runtime, we periodically set τ_0 to a multiple w of the effective window size τ observed at the end of a completed round. As the window size can only decrease during a round, w should be set to a value above 1 to avoid a gradual decay of τ_0 towards 0. We observed only little effect of the frequency of the adaptation and thus used a fixed period of 100 rounds. Figure 4 shows the speedup achieved by parallel execution of the modified susceptible-infected-recovered model using 16 threads, varying the window size factor w. The global counter of infected agents was disabled. We observe that substantial speedup is achieved beyond 2^{18} agents at all values of w. Simulations using values of 2.5 and 5.0 achieve somewhat higher performance than those using 1.25. The difference diminishes with increasing agent counts. The choice of w

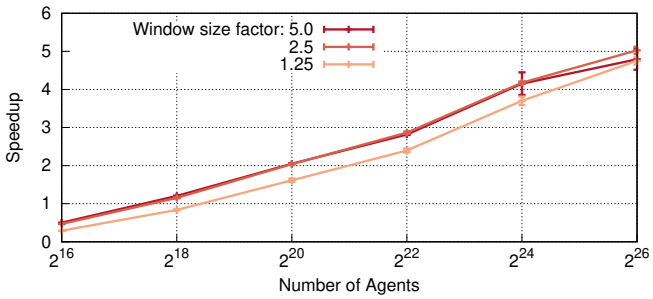


Fig. 4. Speedup over sequential execution depending on the window size factor using 16 threads. The smallest window size of 1.25 results in an overly conservative execution. Best performance was achieved using a factor of 2.5 or 5.0. We note that the simulation performance is quite robust to changes in the window size factor.

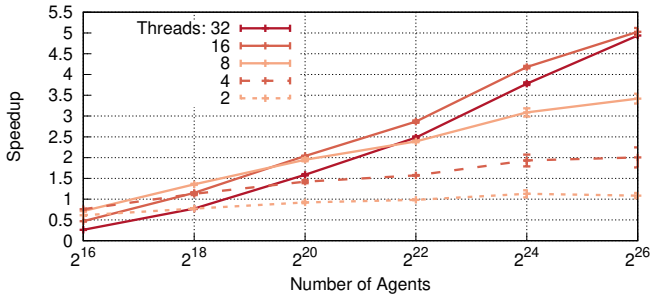


Fig. 5. Speedup over sequential execution depending on the number of threads. Performance increases are observed at 2^{18} agents and beyond. Due to non-uniform memory access among the two 16-core CPUs in our test system, peformance does not improve beyond 16 threads.

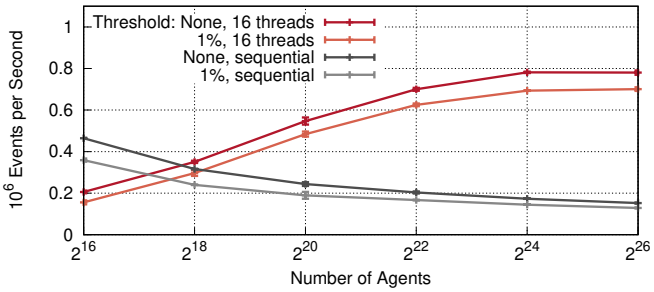


Fig. 6. Committed transitions per second wall-clock time with and without a global counter updated whenever the number of infected agents has changed by 1% of the total agents. While the notification overhead reduces the absolute processing rates, the speedup over the sequential case is largely maintained.

strongly affects the rollback ratio, i.e., the number of rollbacks per transition. In a simulation of 2^{20} agents with $w = 1.25$, the ratio is only 0.27. With $w = 2.5$ and $w = 5.0$, the ratio increases to 1.43 and 3.84, i.e., the number of transitions that are rolled back exceeds the number of committed transitions. On the other hand, larger values of w still allow for more transitions to be committed in each round, decreasing the number of rounds to complete the simulation. When setting w to 1.25, 2.5, and 5.0, a total of $2\,845\,400$ transitions were committed within $25\,942$, $18\,467$, and $17\,237$ rounds.

Figure 5 shows speedup results when varying the number of threads, with the global counter of infected agents disabled. At 2^{18} agents, most parallel configurations start to outpace the sequential execution, with the exception of the configurations using 2 and 32 threads. With 2 threads, the overhead involved in executing events speculatively is only amortized by parallel execution around 2^{22} agents and beyond. Even at 2^{26} agents, the speedup is only 1.1. In contrast, the larger parallelism of the runs with 32 threads enables substantial performance gains at large agent counts, with a maximum speedup of 4.9 at 2^{26} agents. The highest observed speedup of 5.0 was achieved using 16 threads, which coincides with the number of physical cores on each CPU socket of our test system. We ascribe the slightly lower performance with 32 threads to the non-uniform memory access among cores on different sockets. Overall, we observe that although the speedup scales far from linearly with the number of CPU cores, our synchronization scheme substantially accelerates simulations of large populations.

Figure 6 compares the number of committed transitions per second wall-clock time with the global counter disabled and enabled, i.e., with or without taking into account the macro property of the number of infected agents an individual's movement rate When enabled, all the agents' movement rates are newly calculated every time the number of infected agents has increased or decreased by 1% of the overall agent count. To avoid performance artifacts depending on the simulation duration in logical time, in contrast to the previous experiments, the simulations were terminated at 5 units of logical time independently of the number of agents. As expected, the induced recalculation of rates and rescheduling of events decreases performance both in the sequential and parallel case. At 2^{26} agents, the commit rate is reduced by about 16% from $152\,000$ to $128,000$ transactions per second in the sequential case, and by 10% from $780\,000$ to $700\,000$ in the parallel case. Although the absolute performance is lower when considering the macro property, a higher speedup of 5.5 is observed.

VI. Related Work

The proposed synchronization scheme is closely related to Steinman's classical *breathing time buckets* algorithm [24]. Breathing time buckets is a synchronous algorithm in which the processing within each round is bounded by the timestamp of the earliest newly created event. Any computation beyond this point in logical time, referred to as the *event horizon*, is rolled back. Newly scheduled events are only sent among logical processes at the end of a round, leading to an execution scheme that alternates between the processing and exchange of events. Our approach differs in its deviation from a purely event-oriented representation of state changes and in its definition of the event horizon: as detailed in Section II, the considered agent-based models combine individual state transitions at an agent with read and/or write accesses as well as potential rescheduling operations at neighboring agents, all occurring at the same point in logical time. Since the breathing time buckets algorithm delays event exchanges to the end of a round, any such combined operation would require multiple rounds to complete. Instead, our algorithm permits direct access to the neighboring agents' states. Such inter-agent accesses may be invalidated throughout the processing of a round, requiring rollbacks. To minimize the required state history per agent and to avoid costly interactions among threads to signal access invalidations, the algorithm maintains the invariant that at the end of a round, only at most one access per agent is committed. As an agent may only schedule events targeting itself, the creation and subsequent execution of an event within the same round would constitute two accesses to the agent, which is not permitted in our algorithm. Thus, our definition of event horizon is stricter than the one used in the breathing time buckets algorithm.

Several mechanisms for synchronization and state accesses have been proposed that depart from the classical parallel and distributed simulation paradigm wherein simulation objects are assigned to logical processes and all interactions among objects are represented in the form of event exchanges. Ghosh and Fujimoto proposed the concept of space-time memory, which introduces a versioning of variables in Time Warp simulations to correctly handle concurrent accesses in shared memory settings [25]. Chen et al. aggregate logical processes to groups within which variables can be accessed directly [26]. Marziale et al. propose the dynamic combination of simulation objects based on runtime information regarding the frequency of mutual accesses [27]. Pellegrini et al. described a scheme that achieves a transparent versioning of global variables [28]. Substantial performance gains are achieved by avoiding some of the event exchanges required in traditional implementations. Ianna et al. proposed an optimistic parallel simulation system in which all threads obtain events from a single shared list [29]. Events are processed without a fixed mapping between threads and logical processes or simulation objects, while still avoiding conflicting state accesses. Pellegrini and Quaglia presented a mechanism for transparent access to the state of arbitrary simulation objects in optimistic simulations [30]. This is achieved by an operating system-level redirection of memory accesses. To guarantee correctness, the current event is suspended and a new event is scheduled that handles the actual state access. As in our synchronization scheme, these approaches loosen the mapping between threads and simulation objects encountered in traditional PDES approaches. Our work differs in avoiding the use of events to represent state accesses across objects altogether, and in the synchronous execution scheme that reflects the tightly coupled nature of the agent-based models.

In 2020, Chen et al. presented a mechanism for agent interaction in optimistically synchronized distributed simulations [31] in which interactions are mediated by mailboxes holding timestamped messages. Since our work assumes a shared memory execution environment, agent interactions take place without mediation. By permitting at most one access to each agent in a round, we avoid maintaining a state history similar to a mailbox beyond a "current" and "projected" agent state. We leave an exploration of variants of our synchronization scheme with per-agent state histories within each window to future work.

The parallel and distributed execution of the stochastic simulation algorithm in its traditional domain of biochemical systems has been explored by several authors. The Next Subvolume Method, which is a spatial extension of the Next Reaction Method based on a regular grid, has been executed using variants of the Time Warp algorithm [32]–[34]. As in our work, constraining the optimistic execution proved beneficial for performance [34]. Similarly, Goldberg et al. employ Time Warp to execute the Next Reaction method. In contrast to our synchronization scheme, Time Warp requires interactions among simulation objects to be mediated through events and allows logical processes to progress asynchronously. Still, a comparison of the performance our synchronization scheme when executing ML3 models as compared to the use of a Time Warp kernel is an interesting avenue for future work.

A variety of methods have been proposed to control the degree of optimism in speculative synchronization schemes by limiting the relative progress of logical processes to balance the exploitation of parallelism and the overhead for rollbacks [35]–[38]. In our synchronous approach, the progress within each window is limited by an upper bound in logical time calculated based on the effective size of a previous window. During each round, the effective window size is gradually reduced. Since threads promptly terminate the current round once the current window size prevents any further progress, the sensitivity to the initial window size is low (cf. Section V), avoiding the need for sophisticated mechanisms to determine a suitable initial window size.

While outside the scope of our present work, some agent-based models may benefit from efficient mechanisms to support global range queries, e.g., to select all agents within a certain age group. Several efficient algorithms for this purpose targeting distributed environments have been proposed and evaluated in the literature [39], [40].

Finally, many-core accelerators such as graphics processing units (GPUs) are a natural fit for parallel simulation of tightly coupled agent-based systems. The benefits of the presented approach in the context of agent-based and optimistically synchronized discrete-event simulation on GPUs [41], [42] will be explored as part of our future work.

VII. Conclusion

We presented a speculative synchronization approach targeting models of tightly coupled agents in continuous time. By employing a synchronous execution scheme in shared memory, we enable direct attribute accesses across agents without mediation through events and restrict the state history required for rollbacks to only a single entry per agent. On the example of an extended susceptible-infected-recovered agent-based model that emphasizes the challenging characteristics of the considered model class, we showed that our synchronization scheme can accelerate simulations of 6.7×10^7 agents by a factor of up to 5.5 using 16 cores.

Our experiments focused on agent-based models in which event times are determined based on dynamically updated transition rates, and in which state updates at one agent entail instantaneous accesses to other agents' attributes. However, the mechanism for speculative direct access to simulation objects is general. Thus, a promising direction for future work lies in exploring the benefits of our synchronous speculative scheme when executing discrete-event models of other systems of tightly coupled entities.

Acknowledgement

Financial support was provided by the Deutsche Forschungsgemeinschaft (DFG) research grant UH-66/15-1 (MoSiLLDe).

REFERENCES

[1] N. Gilbert and K. Troitzsch, *Simulation for the social scientist*. McGraw-Hill Education (UK), 2005.

[2] A. L. Bazzan and F. Klügl, "A review on agent-based technology for traffic and transportation," *The Knowledge Engineering Review*, vol. 29, no. 3, p. 375, 2014.

[3] G. An, Q. Mi, J. Dutta-Moscato, and Y. Vodovotz, "Agent-based models in translational systems biology," *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, vol. 1, no. 2, pp. 159–171, 2009.

[4] F. Willekens, "Continuous-time microsimulation in longitudinal analysis," in *New Frontiers in Microsimulation Modelling*, A. Zaidi, A. Harding, and P. Williamson, Eds. Ashgate, 2009, pp. 413–436.

[5] M. Yang, P. Andelfinger, W. Cai, and A. Knoll, "Evaluation of conflict resolution methods for agent-based simulations on the GPU," in *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2018, pp. 129–132.

[6] T. Köster, P. J. Giabbanelli, and A. Uhrmacher, "Performance and soundness of simulation: a case study based on a cellular automaton for in-body spread of HIV," in *2020 Winter Simulation Conference (WSC)*. IEEE, 2020, pp. 2281–2292.

[7] R. M. Fujimoto, "Parallel and distributed simulation systems," in *Proceeding of the 2001 Winter Simulation Conference*, vol. 1. IEEE, 2001, pp. 147–157.

[8] J. Noble, E. Silverman, J. Bijak, S. Rossiter, M. Evandrou, S. Bullock, A. Vlachantoni, and J. Falkingham, "Linked lives: The utility of an agent-based approach to modeling partnership and household formation in the context of social care," in *Proceedings of the 2012 Winter Simulation Conference (WSC)*, 2012, pp. 1–12.

[9] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex Adaptive Systems Modeling with Repast Simphony," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, p. 3, 2013.

[10] S. Tisue and U. Wilensky, "Netlogo: Design and implementation of a multi-agent modeling environment," in *Proceedings of the Agent Conference on Social Dynamics: Interaction, Reflexivity and Emergence*, 2004, updated 2013.

[11] D. Masad and J. L. Kazil, "Mesa: An agent-based modeling framework," in *Proceedings of the 14th Python in Science Conference*, 2015.

[12] T. Warnke, O. Reinhardt, A. Klabunde, F. Willekens, and A. M. Uhrmacher, "Modelling and simulating decision processes of linked lives: An approach based on concurrent processes and stochastic race," *Population studies*, vol. 71, no. sup1, pp. 69–83, 2017.

[13] M. J. Keeling and K. T. Eames, "Networks and epidemic models," *Journal of the Royal Society Interface*, vol. 2, no. 4, pp. 295–307, 2005.

[14] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of computational physics*, vol. 22, no. 4, pp. 403–434, 1976.

[15] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *The Journal of Physical Chemistry A*, vol. 104, no. 9, pp. 1876–1889, 2000.

[16] F. Squazzoni, "The micro-macro link in social simulation," *Sociologica*, vol. 2, no. 1, pp. 0–0, 2008.

[17] A. Klabunde, S. Zinn, F. Willekens, and M. Leuchter, "Multistate modelling extended by behavioural rules: An application to migration," *Population studies*, vol. 71, no. sup1, pp. 51–67, 2017.

[18] D. Jefferson and H. Sowizral, "Fast concurrent simulation using the Time Warp mechanism. Part I. Local control," Rand Corporation, Santa Monica, CA, Tech. Rep., 1982.

[19] W. J. Tan, P. Andelfinger, W. Cai, A. Knoll, Y. Xu, and D. Eckhoff, "Multi-thread state update schemes for microscopic traffic simulation," in *2020 Winter Simulation Conference (WSC)*. IEEE, 2020, pp. 182–193.

[20] W. J. Tan, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "Causality and consistency of state update schemes in synchronous agent-based simulations," in *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2021, pp. 57–68.

[21] D. Blackman and S. Vigna, "Scrambled linear pseudorandom number generators," *arXiv preprint arXiv:1805.01407*, 2018.

[22] P. L'Ecuyer and R. Simard, "TestU01: A C library for empirical testing of random number generators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 4, pp. 1–40, 2007.

[23] C. M. Macal, "To agent-based simulation from system dynamics," in *Proceedings of the 2010 Winter Simulation Conference*. IEEE, 2010, pp. 371–382.

[24] J. Steinman, "SPEEDES: Synchronous parallel environment for emulation and discrete event simulation," in *SCS Western Multi-Conference on Advances in Parallel and Discrete Simulation*, vol. 23, 1991, pp. 1111–1115.

[25] K. Ghosh and R. M. Fujimoto, "Parallel discrete event simulation using space-time memory," Georgia Institute of Technology, Tech. Rep., 1994.

[26] L.-l. Chen, Y.-s. Lu, Y.-p. Yao, S.-l. Peng *et al.*, "A well-balanced Time Warp system on multi-core environments," in *2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 2011, pp. 1–9.

[27] N. Marziale, F. Nobilia, A. Pellegrini, and F. Quaglia, "Granular time warp objects," in *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2016, pp. 57–68.

[28] A. Pellegrini, R. Vitali, S. Peluso, and F. Quaglia, "Transparent and efficient shared-state management for optimistic simulations on multi-core machines," in *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2012, pp. 134–141.

[29] M. Ianni, R. Marotta, D. Cingolani, A. Pellegrini, and F. Quaglia, "The ultimate share-everything PDES system," in *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2018, pp. 73–84.

[30] A. Pellegrini and F. Quaglia, "Cross-state events: A new approach to parallel discrete event simulation and its speculative runtime support," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 48–68, 2019.

[31] S. Chen, M. Hanai, Z. Hua, N. Tziritas, and G. Theodoropoulos, "Efficient direct agent interaction in optimistic distributed multi-agent-system simulations," in *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2020, pp. 123–128.

[32] M. Jeschke, A. Park, R. Ewald, R. Fujimoto, and A. M. Uhrmacher, "Parallel and distributed spatial simulation of chemical reactions," in *22nd Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 2008, pp. 51–59.

[33] L. Dematté and T. Mazza, "On parallel stochastic simulation of diffusive systems," in *International Conference on Computational Methods in Systems Biology*. Springer, 2008, pp. 191–210.

[34] B. Wang, B. Hou, F. Xing, and Y. Yao, "Abstract next subvolume method: A logical process-based approach for spatial stochastic simulation of chemical reactions," *Computational Biology and Chemistry*, vol. 35, no. 3, pp. 193–198, 2011.

[35] P. M. Dickens, D. M. Nicol, P. F. Reynolds Jr, and J. M. Duva, "Analysis of bounded Time Warp and comparison with YAWNS," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 6, no. 4, pp. 297–320, 1996.

[36] A. Ferscha, "Probabilistic adaptive direct optimism control in Time Warp," in *Proceedings of the ninth Workshop on Parallel and distributed simulation*, 1995, pp. 120–129.

[37] H. Rajaei, R. Ayani, and L.-E. Thorelli, "The local Time Warp approach to parallel simulation," in *Proceedings of the seventh Workshop on Parallel and distributed simulation*, 1993, pp. 119–126.

[38] J. Wang and C. Tropper, "Optimizing Time Warp simulation with reinforcement learning techniques," in *2007 Winter Simulation Conference*. IEEE, 2007, pp. 577–584.

[39] R. Ewald, D. Chen, G. K. Theodoropoulos, M. Lees, B. Logan, T. Oguara, and A. M. Uhrmacher, "Performance analysis of shared data access algorithms for distributed simulation of multi-agent systems," in *20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*. IEEE, 2006, pp. 29–36.

[40] D. Chen, R. Ewald, G. K. Theodoropoulos, R. Minson, T. Oguara, M. Lees, B. Logan, and A. M. Uhrmacher, "Data access in distributed simulations of multi-agent systems," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2345–2360, 2008.

[41] X. Liu and P. Andelfinger, "Time Warp on the GPU: Design and assessment," in *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2017, pp. 109–120.

[42] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "A survey on agent-based simulation using hardware accelerators," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–35, 2019.