Efficient Inter-Process Synchronization for Parallel Discrete Event Simulation on Multicores

ABSTRACT

We present a new technique for controlling optimism in Parallel Discrete Event Simulation. It is designed to be suitable for simulating models, in which the time intervals between successive events between different processes are highly variable, and have no lower bounds. The basic idea of our technique, called *Dynamic Local Time Window Estimates (DLTWE)*, is that each processor communicates time estimates of its next inter-processor event to its neighbors, which use the estimates as bounds for their local simulation time. We have implemented our technique in a parallel simulator for spatial stochastic simulation, and present an evaluation of its performance. We show that the DLTWE technique drastically reduces the frequency of rollbacks and enables speedups which is superior to that obtained by other works. We also show that the DLTWE technique significantly improves performance over other existing techniques for optimism control that attempt to predict arrival of inter-process events by statistical techniques.

1. INTRODUCTION

Discrete Event Simulation (DES) is an increasingly important tool for evaluating system models in all fields of science and engineering. To improve the capacity and performance of DES simulators, several techniques for Parallel DES (PDES) were developed in the 90's [21, 15, 20, 11]. Parallelization made it possible to simulate large system models, but it was challenging to achieve good speedup corresponding to the number of employed processors. A major difficulty for increasing efficiency through parallelization was that PDES requires fine-grained synchronization between processing elements, which was not easy to realize efficiently on multiprocessors at that time, given the comparatively long communication delays between processing elements. With the current advent of chip multicore processors, these delays have decreased, triggering the development of new techniques for PDES targeting multicores (e.g., [4, 23, 22, 31, 32]).

In PDES, the simulation model is partitioned onto logical processes (LPs), each of which evolves its sub-model along a local simulation time axis. LPs exchange timestamped events to incorporate inter-LP dependencies. Each LP must ensure that the processing of incoming events is correctly interleaved with local events. The problem with incoming events that violate an LP's local timestamp ordering (so-called *stragglers*) can in principle be handled in two ways: *conservative* approaches allow an LP to process an event only when it is guaranteed that no straggler will later arrive [21]; *optimistic* approaches allow stragglers by invoking suitable corrective action (rollback) [15, 20]. In purely conservative approaches, local execution of LPs may be blocked excessively unless inter-LP events can always be predicted long in advance (e.g., when simulating networks with long communication latencies), which most often is not possible. On the other hand, in optimistic approaches, performance may be damaged by excessive numbers of rollbacks. Many approaches to PDES therefore allow stragglers, but control the optimism by various heuristic techniques, based on, e.g., observed frequency of rollbacks [25, 5], patterns of past inter-LP messages [10], etc.

In this paper we present a new technique for controlling optimism in PDES. It is particularly designed for high efficiency when simulating models in which the time intervals between successive inter-LP events are highly variable and have no lower bounds. Such models pose severe difficulties for both conservative and existing variants of optimistic approaches. Our technique, called *Dynamic Local Time Window Estimates (DLTWE)*, increases efficiency by exploiting the opportunities for fast multicore inter-LP communication. DLTWE assumes that an LP can reasonably estimate timestamps of its next k outgoing inter-LP events, where k is a tuneable parameter of our technique. Each LP continuously communicates these estimates to its corresponding neighboring LPs, which use the estimates as bounds for advancing their local simulation time. Since the communicated timestamps are merely estimates, DLTWE does not rule out the occurrence of stragglers, meaning that each LP must perform rollbacks when needed. If the estimates are sufficiently accurate, then the number of rollbacks should be small, allowing the simulator to operate with high efficiency.

We have developed the DLTWE technique in the context of stochastic spatial simulation of models governed by the mesoscopic reaction-diffusion master equation (RDME) [12, 2]. Here the model's geometry is discretized into small sub-

volumes (a.k.a. voxels), each of which contains a discrete number of species (e.g., molecules). In each subvolume the species obey prescribed stochastic reaction laws and the species may move (by diffusion) to other neighboring subvolumes. When simulating an RDME model using PDES, the subvolumes are partitioned onto LPs. Hence, a diffusion event between two boundary subvolumes causes inter-LP communication. By the Markovian nature of the model, the waiting time of any event is an exponentially distributed random variable; thus the waiting time has a significant variance and no lower bound.

In the present paper we show that the DLTWE technique has small overhead when implemented on a shared-memory multicore processor. In our simulator, each LP maintains a list of future events, whose occurrence times have already been sampled; this is already a component of our technique for simulating RDME models [Anonymous, 2015].We show how the DLTWE technique can be tuned by limiting how far into the future DLTWE estimates will be provided: The cost of providing more accurate DLTWE estimates further into the future can be tuned, both against the cost of rollbacks caused by poor estimates and against achieving limited optimism in contexts where rollbacks are relatively inexpensive.

In the paper, we also demonstrate the effect of a technique to limit the cost or rollbacks by reversing only those processed events that are causally dependent on the straggler that caused the rollback in the first place. Less costly rollbacks also allow more optimism in the simulation, thereby limiting waiting and increasing overall simulation efficiency.

In our evaluation, we show how our implementation of the DLTWE technique enables speedups in parallel simulation of RDME models, which is superior to that obtained by other works. In particular, we compare the DLTWE technique against other existing techniques for optimistic control, such as the Probabilistic Adaptive Direct Optimism by Ferscha [10], and show that employing the DLTWE results in at least a doubling of parallel efficiency. We support this comparison by a detailed profiling of the simulator behavior, which shows how DLTWE significantly reduces both the cost of unnecessary blocking and of excessive rollbacks.

Outline. After reviewing related work in the next section, we review the class of spatial stochastic simulation models considered by our simulator in Section 3. A detailed description of our parallelization algorithm, including the DLTWE technique, is given in Section 4. Section 5 contains a detailed evaluation of the performance of our parallelization technique, including a detailed breakdown of the simulation effort, and a comparison with other techniques for optimism control. Section 6 contains conclusions and directions for future work.

2. RELATED WORK

Numerous methods for synchronization in PDES have been proposed. Extensive surveys are provided in [11, 6, 14]; here we can only review a selection.

Synchronization methods can be crudely classified into *conservative* [21] and *optimistic* [15, 20]. Each approach has its drawbacks, which subsequently proposed techniques aim to mitigate. For instance, *conservative time windows* [1] are used to increase parallelism in a purely conservative approach: this assumes that there is always a guaranteed lower bound on the delay until the next inter-LP event, which does not exist in stochastic simulations that we consider.

Optimistic approaches [15, 20] have the potential to achieve higher parallelism, but performance may be damaged by excessive rollbacks. Many techniques have been developed for controlling the optimism and limiting the frequency of rollbacks. One idea is to employ dynamically moving *time windows* that bound how far each LP can advance its local time (e.g., [27, 29]). Synchronization between time windows typically assumes frequent calculation of *global virtual time* (GVT), which is an expensive global calculation, for which a special high-speed network is recommended. A further development of these approaches is the class of "near-perfect" state information (NPSI) protocols, including the *elastic time algorithm* [28]. Here, the bound is based on (GVT) and information about future messages to neighboring LPs, which is computed and communicated over a special high-speed network. Our DLTWE approach is also based on controlling optimism by information propagated between LPs; however, we show how such an idea can be realized on a modern multicore without using a special high-speed network.

There are also approaches where optimism control can be performed by LPs based on locally available information, not requiring a special high-speed network. In some of these techniques, each LP autonomously regulates its event processing speed against parameters, such as frequency of rollbacks [25, 5]. Another approach is to use the pattern of past incoming inter-LP messages [10] to predict the time of the next incoming message by statistical techniques, thereby obtaining a bound for advancement of local time. We compare this approach to our technique in Section 5.5. A study where model-specific information has been used to extract additional synchronization information (in the form of an extended lookahead) is presented in [19]. Here, up to three model-dependent heuristics have been exploited to significantly increase parallel performance in the simulation of wireless sensor networks.

Other performance-enhancing techniques in optimistic PDES include to avoiding rollbacks due to out-of-timestamp-order when this is possible. In [3] the authors introduce a "lookback", a limited history of recent events. When a straggler event arrives, it is checked against the history, and if no causality error is found, the event is processed as if arriving in order. In [18], the authors view the processes in the simulator as objects of abstract data types, and messages as operations being performed on the objects. Some of the operations commute with each other, and hence rollbacks can be avoided.

Recently, techniques for PDES that specifically target multicores have been developed. One approach is to allow each subdomain to be accessed by several cores (e.g., [4, 23, 22]), thereby achieving better load balancing. We conjecture that for our work, this benefit would be more than outweighed by the cost of the necessary synchronization.

Exact parallel simulation of RDME models was previously addressed by [16, 30, 7]. The simulators are implemented in the distributed environment, where an LP simulates either a single subvolume [7, 30] or a larger subdomain [16]. In the latter case, synchronization with neighbors occurs solely at the domain boundaries, which increases the LP's private workload and reduces the communication cost per subvolume. As discussed by [7], conservative simulation of RDME models is infeasible due to the lack of lookahead. Hence, simulators rely on optimistic protocols. A reduction of rollback cost was previously implemented by a static distance from the GVT [16] or adaptive protocols, such as Breathing Time Warp [30].

3. SPATIAL STOCHASTIC SIMULATION

In this section, we review the class of spatial stochastic simulation models considered by our simulator.

The reaction-diffusion master equation (RDME) can be regarded as a framework to describe the dynamics of spatially extended Markovian processes of interacting entities. As the name suggests, the RDME is a suitable model for chemical reactions in a diffusive environment, but processes from biology, epidemics, and many other applications may also be successfully treated. In particular, the RDME is particularly suitable for systems where discrete effects (due to small populations) and thermal noise should not be neglected.

The spatial domain of interest is divided into *subvolumes*, each of which maintains a *copy number* (discrete count) of all participating *species*. The dynamics of the model is then a continuous-time Markov chain over the state space consisting of all copy numbers in all subvolumes. The state transitions fall in one of two categories, (i) a *reaction event* acts in a *single* subvolume by removing a combination of species and replacing it with a different combination, (ii) a *diffusion event* moves a single unit of one species from one subvolume to a neighboring subvolume, and hence changes the state of *two* subvolumes. The waiting time for each transition is exponentially distributed with an intensity that is proportional to the product of the copy numbers of the involved species.

As a concrete example, a reaction from the Lotka-Volterra predator-prey model described in Section 5.1 reads

$$B + C \xrightarrow{r} 2C, \tag{1}$$

that is, in a particular subvolume one unit of B (prey) is consumed and one unit of C (predator) is produced. The intensity for this event is proportional to the product of the number of B's and C's, where r is the constant of proportionality, later referred to as the reaction rate. At any time t, the waiting time to the next event is exponentially distributed with this intensity.

In a *spatial* context, prey in one subvolume can escape by moving to another subvolume. If B_i and B_j denote the population of preys in neighboring subvolumes i and j, then

$$B_i \xrightarrow{q_{ij}} B_j, \tag{2}$$

expresses the event that one unit of prey in the ith subvolume moves to the jth. The waiting time for this event is equal to the product of B_i and the transport rate constant q_{ij}. Depending on the scaling of this constant versus the spatial units, different types of transport may in principle be modeled. In this work we consider the *diffusive scaling regime*, in which $q_{ij} \propto h^{-2}$, with h a length-scale (e.g., a radius) of the subvolumes. Notably, with a finer discretization (i.e., $h \to 0$), the number of diffusion events will increasingly dominate the Markov chain.

It was Gillespie [13] who first detected the feasibility of simulating independent samples from master equations in general. For RDMEs one of the first practical sampling algorithms was proposed in [8], the Next Subvolume Method (NSM).

In this work we consider a related method, the All Events Method (AEM) [Anonymous, 2015]. The algorithm generates next event times for each reaction and diffusion in all subvolumes and stores them in an *event queue*. It proceeds by repeatedly selecting the event with the smallest time from the event queue, processes it by updating the state, and finally updates the event queue by sampling the next time for the event just processed. Also, at this stage, those rates which have changed due to the state update need to be rescaled (see [Anonymous, 2015] for details).

Being essentially a spatial extension of the Common Reaction Path method [24], the AEM has the benefit of being able to produce *coupled* trajectories, thus defining a consistent stochastic *flow*. Besides implying a much reduced variance in statistical estimators, this is also required when evaluating the effect of small perturbations or coefficient uncertainties in a strong sense (e.g., root-mean-square, see [Anonymous, 2015] for a discussion). Furthermore the property is of use for fitting of model parameters using numerical optimization routines.

Of relevance to the current application, the AEM stores the waiting times for the next instance of each reaction or diffusion event such that it is possible to accurately predict when specific events will happen, notably including diffusion events between subvolumes. Another feature of more practical nature is that, by seeding the random number generators in an identical way, the parallel simulations yield identical results independently on the number of LPs, thus ensuring correctness. These features come at a certain cost, however, as the AEM requires to store more entries in the event queue compared to, for example, the NSM.

4. PARALLEL IMPLEMENTATION

In this section we detail our parallelization of the All Events Method (PAEM), which implements the DLTWE for a completely general class of RDME models.

In our parallel simulator, the subvolumes of the simulation model are partitioned into subdomains, each of which is assigned to an LP. Each LP evolves the state of its subdomain while maintaining three main data structures: (i) the *subdomain state*, i.e., for each subvolume, the number of entities of each species as well as the timestamp of the last event affecting the subvolume, (ii) a time-sorted *event queue*, containing the next occurrence of each event type for each subvolume in its subdomain, and (iii) a *rollback history*, which

is a time-sorted sequence of events already processed by this LP.

Each LP advances the simulation by processing events that affect its subdomain. The LP repeatedly finds the next event for processing, either in its event queue or in a message from another LP, and processes it by (i) updating the states of affected subvolumes, (ii) adding the event to its rollback history, and (iii) adding the next event of the same type and subvolume to its event queue. If the event is a diffusion event which crosses a subdomain boundary, then a message is transmitted to the neighboring LP; each pair of LPs is connected by a FIFO channel in each direction.

Whenever an LP receives a diffusion message that causes a causality violation (i.e., it is a *straggler*), it must perform a *rollback* to the time immediately before the straggler's timestamp, using its rollback history. We have implemented two different versions of the rollback operation: a more costly simple rollback and a less costly selective rollback. The selective rollback is described further below. In the *simple rollback*, the local time of the LP is reset to the time immediately preceding the timestamp of the straggler, and the events in the rollback history that occur after this timestamp are processed "backwards". All diffusion messages that had been sent by the LP during the rollback interval must be undone by sending corresponding *anti-messages* to the corresponding LPs. An anti-message *cancels* any event that was sent earlier with the same or a later timestamp. The receipt of an anti-message triggers rollbacks at the receiving LPs if it cancels a message that has already been processed.

Since rollbacks triggered by stragglers hurt performance, an LP should try not to advance its local simulation time past the timestamp of any diffusion message that will be received in the future. For this purpose, we have developed the DLTWE (Dynamic Local Time Window Estimate) technique, whereby each LP communicates to each of its neighboring LPs an estimate of the timestamp of the next diffusion to respective LP; these estimates are obtained from the current contents of the event queue. An LP does not advance its local simulation past the time of the earliest incoming time estimate. Depending on the overall presence of DLTWEs between LPs, the optimism of the simulation is controlled.

To reduce the impact of rollbacks, we have developed a technique for *selective rollback*. An LP that receives a straggler or an anti-message performs a refined analysis before executing a rollback. Rather than merely comparing the timestamp of the incoming diffusion message with its local simulation time, the receiving LP traces the causality violations that are incurred by the incoming message. The LP finds the processed events that are causally dependent on the straggler or anti-message using the trace. Only these events are rolled back. The cost of selective rollback is typically significantly lower than the cost of simple rollback.

The simulator main loop. Algorithm 1 is a pseudocode description of the main loop executed by each LP. Lines 2 through 6 define the main data structures. These are

EventQueue is a time-sorted priority queue containing the scheduled local events;

SubvolumeState represents the state of each subvolume in the subdomain, i.e., the number of entities of each species in each subvolume, as well as the timestamp of the last event affecting the subvolume,

History is a time-sorted sequence of events already processed by th LP; old events are regularly removed from the history by fossil collection, which we do not further describe here,

Channels contains an incoming message channel for each neighboring LP, and

$Dltwe_{i,j}$ consists of a DLTWE estimate from LP_i to LP_j,defined for each pair of neighboring LPs.

For an event e, we let e.time denote its timestamp; for a diffusion event e, we let e.dest denote its destination subvolume. For a subvolume s, we let $\mathrm{dom}(s)$ denote the index of the LP to which s belongs.

The main simulator loop consists of two phases. The first phase (lines 8–18), finds the next event to be processed, as follows. First, For each incoming channel, the first message that is not canceled by a later anti-message in the channel, is retrieved by means of the function RetrieveMsg. Intuitively, the retrieved message is the first one in the channel that should be processed after all rollbacks induced by anti-messages have been performed. The earliest of these messages is assigned to e_{msg}. If e_{msg} is a straggler which violates causality in its destination subvolume (checked at line 9) then a rollback is performed. Second, the earliest event e_{local} in the event queue is read. If e_{msg} is earlier than e_{local} (line 12), then e_{msg} is assigned to e for processing. Otherwise, the event e_{local} is assigned to e for processing, but only if no DLTWE estimate is violated (line 15). The algorithm restarts the loop from line 8 if such a violation would occur.

The second phase (lines 19–30) updates the subdomain state of the LP by processing the event e that was selected in the first phase. It starts by checking whether e is a "local straggler", i.e., a local diffusion event that would cause a local causality error (line 19), in which case a rollback is necessary. Thereafter, e is processed by adding it to the event history (line 21), updating the states of affected subvolumes, and updating the times of future events in the event queue that are affected by the state change(s) (lines 22 through 26). If e is a diffusion to another subdomain, a message is sent (line 28) to the appropriate LP. After that, the DLTWEs are updated (line 30) to inform the neighboring LPs of the estimated times of the next diffusion events.

DLTWEs are computed based on outgoing diffusion events that can be found in a prefix of the event queue, thus not considering diffusion events that are scheduled far into the future. If no relevant diffusion events for a specific LP are found in the prefix, the corresponding DLTWE is set to infinity. The length of the considered prefix is a tuneable parameter of our approach. A short prefix induces less effort for DLTWE computation, but will generate DLTWE estimates for only a small subset of neighboring LPs, inducing more optimism in the simulation; too much optimism may result in high cost for rollbacks. A long prefix, on the

Algorithm 1: Main loop of Parallel AEM Simulator, executed by each LP.

```
1: LPs are indexed 1...N, Subvolumes of LP_i are indexed 1...n_i.
2: EventQueue                                  ▷ Time-sorted priority queue of scheduled events
3: SubvolumeState[1...n_i]                     ▷ Current state of subdomain
4: History                                     ▷ Event history, used for rollbacks
5: Channels[neighbor LP_j s]                   ▷ Incoming message channels, one for each neighboring LP
6: Dltwe_{i,j}                                 ▷ A DLTWE from LP_i to LP_j (defined for each pair of neighboring LPs)
7: while true do
       ▷ Earliest processable message in incoming channels
8:     e_msg ← earliest message in {e_rcv | chan ∈ Channels, e_rcv ∈ {RetrieveMsg(chan)}
9:     if e_msg.time < SubvolumeState[e_msg.dest].time then   ▷ If e_msg is a straggler.
10:        SelectiveRollback(e_msg)                           ▷ Then a rollback must be performed.
11:    e_local ← earliest event in EventQueue
12:    if e_msg.time ≤ e_local.time then                      ▷ If m precedes any local event
13:        e ← Pop e_msg from its message channel             ▷ The event e to be processed is from the incoming channels
14:    else
15:        while ∃ neighboring LP_j s.t. Dltwe_{i,j} ≤ e_local.time do ▷ If next event is later than some DLTWE
16:            if ∃e_new ∈ Channels[LP_j] or
                       exists message in other channel earlier than Dltwe_{i,j} then
17:                goto 8                                     ▷ restart loop from line 8
18:        e ← pop e_local from event queue                   ▷ Otherwise the event to be processed is the next local one
       ▷ Second phase begins here
19:    if e is a local diffusion event and SubvolumeState[e.dest].time > e.time then
20:        SelectiveRollback(e)
21:    Add e to History
22:    Update state of SubvolumeState[e.subvol]
23:    Update timestamps of affected future reactions/diffusions in EventQueue
24:    if e is a diffusion then
25:        if e.dest is local then
26:            Update SubvolumeState[e.dest]
27:        else
28:            Send diffusion message to owner of e.dest
29:    for each neighbor LP_j do
30:        Dltwe_{j,i} ← min({e_diff.time | e_diff ∈ prefix(EventQueue) and e_diff.dest is in the domain of LP_j} ∪ ∞)
```

other hand, costs more effort for DLTWE computation, and will avoid excessive cost of rollbacks, but may in some context also induce too little optimism. How to tune the prefix length to make this trade-off is examined in Section 5.4.

As an optimization, the updates of the DLTWEs at line 30 are performed only when necessary, i.e., when the estimated time of some future inter-LP diffusion event is updated. The DLTWE estimates are communicated using a single memory cell per direction and neighboring LP-pair, which is only written to when this results in a new value, to avoid unnecessary coherence traffic.

The SelectiveRollback function:
The function SelectiveRollback(e_{cause}), shown in Algorithm 2, reverses the effect of all events processed by an LP at or after time e_{cause}.time, that are causally dependent on e_{cause}. Each subvolume may be rolled back to a distinct timestamp. First, we let H contain the part of the history that may be rolled back (line 2). After that we define a set D of $\langle \text{subvolume}, \text{timestamp} \rangle$ pairs. Intuitively, D defines the timestamp to which events has to be rolled back to for each subvolume. Initially, if e_{cause} is a straggler, D must at least contain $\langle e_{\text{cause}}.\text{dest}, e_{\text{cause}}.\text{time} \rangle$ (line 4). If e_{cause} is an anti-message, D instead contains all destination subvolumes d of messages received from $\text{dom}(e_{\text{cause}})$ after time e_{cause}.time, and their respective time (line 6). Thereafter, D is completed to contain all $\langle \text{subvolume}, \text{timestamp} \rangle$ pairs that are causally dependent on e_{cause}, such that for every diffusion event between any two subvolumes s, s' at time t, we have that if $\langle s, t' \rangle \in D$ and $t > t'$, and there is no diffusion e_{diff} such that $t' < e_{\text{diff}}.\text{time} \leq t$, then $\langle s', t \rangle \in D$, or vice versa (line 7). In the main **while** loop, the subset of the history H is traversed backwards in time, event by event (line 8). Each event being incident on some subvolume occurring in D, after the corresponding time t, is rolled back. An event e is reverted by reversing the state changes of the affected subvolumes, and the rescheduling of affected events in the event history. If the rollback was initiated by an anti-message, and e is an incoming diffusion originating from a neighbor LP that did not send this anti-message, then e will be pushed back to the top of its originating message channel (checked at line 12). If one or more diffusion events have been sent to a neighbor LP_j during the rollback interval, a single anti-message will be sent, containing the timestamp of the earliest message sent to LP_j after t (starting at line 14).

Algorithm 2: Rollback events at or after time t.

```
1: function SelectiveRollback(event e_cause)
2:     H ← {e | e ∈ History and e.time ≥ e_cause.time}
3:     if e_cause is straggler then
4:         D ← {⟨e_cause.dest, e_cause.time⟩}
5:     else                                  ▷ e_cause is an anti-message
6:         D ← {⟨e.dest, e.time⟩ | e ∈ H, dom(e.subvol) =
                                        dom(e_cause.subvol)}
7:     D ← extend D under causal dependence
8:     while e ← pop latest event in H do
9:         if ∃⟨s, t⟩ ∈ D s.t.
                e.subvol = s and e.time ≥ t then
10:            revert e
11:            Pop e from History
12:            if e.subvol ∈ LP_j and not (isanti(e_cause) and
                    dom(e_cause.subvol) = LP_j) then
13:                push e back to front of Channels[LP_j]
14:    for each neighbor LP_j do ▷ send anti-messages
15:        e_diff ← earliest rollbacked diffusion to LP_j
16:        send anti-message with time e_diff.time to LP_j
```

The RetrieveMsg function:
The function RetrieveMsg(*chan*) returns the first message in the incoming channel *chan* that can be meaningfully processed, i.e., it is not undone by a corresponding anti-message already present in the *chan*. The function starts by finding the timestamp of the earliest anti-message in the channel (line 3). Thereafter, messages are popped and discarded from the channel, until either the first message preceding the earliest anti-message is encountered, or until the anti-message itself is encountered (lines 4–7). In the former case, the function returns the message immediately without popping it from *chan*. In the latter case, a rollback corresponding to the anti-message is performed (line 8) and the procedure is repeated. Thus, there are two possible states of the channel after the completion of the function: *a*) either there are no anti-messages left in the channel, or *b*) the first message in the channel is a diffusion event and has a time earlier than the time of the earliest anti-message in the channel.

Algorithm 3: Locating the first processable message.

```
1: function RetrieveMsg(channel chan)
2:     while chan contains anti-messages do
3:         e_anti ← earliest anti-message in chan
4:         for e ← first message in chan do
5:             if e.time < e_anti.time then return e
6:             pop e from chan
7:             if e = e_anti then break
8:         Rollback(e_anti.event)
9:     return first message in chan
```

5. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our parallelization technique. The aim is to answer the following questions.

How does our technique scale with the number of LPs? In Section 5.2 we determine the speedup obtained on benchmarks, and investigate the dependency on model parameters.

How does the parallelized simulator behave? In Section 5.3 we describe how the computation effort is spent on different activities, exposing potential bottlenecks.

How should the DLTWE technique be tuned? In Section 5.4 we describe how to tune the cost for the computation of the DLTWE estimate against the gain in reduced rollback frequency

How does the DLTWE technique compare to other techniques? In Section 5.5, we compare the DLTWE technique to adaptive optimism control techniques based on local history. In Section 5.6, we compare our parallel simulator to other existing simulators for RDME models.

The performance was evaluated on three sets of benchmarks, described in more detail in Section 5.1. All experiments were run on a 4-socket Intel Sandy Bridge E5–4650 machine, each socket having 8 cores and a 20 MB shared L3-cache. Hyperthreading was used, resulting in a total of 16 hardware threads per processor. An LP is always assigned to a single thread. Speedup is defined as the simulation time of the sequential algorithm (AEM) over the (wall-clock) simulation time of parallel algorithm (PAEM). Three-dimensional geometries were constructed using Comsol Multiphysics 4.3 and converted to computational models using the URDME framework [Anonymous, 2012]. The two-dimensional structured meshes used in the spatial predator and prey model were constructed using custom Matlab scripts. The resulting meshes were divided into subdomains using the multilevel k-way partitioning method provided by the `Metis` library [17]. `Metis` optimizes the partitioning for minimal number of diffusions crossing subdomain boundaries, while maintaining an equal number of subvolumes in each subdomain.

5.1 Benchmarks

We investigated the behavior of our simulator on three benchmarks. In the first benchmark we evaluated the scaling as a function of the geometry and the ratio of diffusion to reaction events (D:R). The D:R was measured during a sequential profiling run. The second benchmark is a spatial predator and prey model in two dimensions, which was previously used for performance evaluation by others [30]. The last benchmark is the simulation of the Min-protein system in a three-dimensional model of the *E. Coli* bacterium.

Reversible isomerization. We created spatial models from different three-dimensional geometries, namely a *sphere*, a *disc*, and a *rod*, all of equal volume. The RDME model considered consists of two freely diffusing species, A and B, each with initial copy numbers of 1000 per subvolume. We prescribed the simplest possible reversible isomerization

$$A \xrightarrow{c} B, \qquad B \xrightarrow{c} A, \tag{3}$$

where the reaction rate c is selected such that the D:R is 1 when both species diffuse at a diffusion rate of 1. The diffusion rates of both species were varied in $\{1, 100\}$, thereby increasing the D:R. We also varied the volume of the geometries in $\{1, 10, 100\}$. For the sphere and the disc we did this by increasing the radius, keeping the height of the disc at the constant value 0.2. For the rod, the radius was kept at the

value 0.2 while the length was increased. As all discretization parameters remained the same for all model configurations, the number of subvolumes in each model grew to approximately $\{1500, 15000, 150000\}$. In the following, we refer to the specific model configurations as [vx,dy], denoting that the model has a volume of x and that the diffusion rates for both species are y.

Spatial predator-prey model. This benchmark is the spatial extension of the Lotka-Volterra model, proposed by Schinazi [26]. We use the model parameters proposed by Wang et al. [30]. The system contains three species, A, B, and C, where the initial copy number for each is set to 1000 per subvolume. The model reads

$$\begin{aligned} A + B &\xrightarrow{0.01} A + 2B, \\ B + C &\xrightarrow{0.01}, 2C \\ C &\xrightarrow{1} \emptyset. \end{aligned} \tag{4}$$

The geometry is a two-dimensional square with a varying side of length $\{64, 200, 400\}$ units and with square subvolumes of unit area. The diffusion rates of species B and C are either $d_B^1 = 2.5$ and $d_C^1 = 5$, or $d_B^2 = 5$ and $d_C^2 = 10$, while $d_A = 0$ in all cases. In the first case, the D:R is approximately 1, and about 2 in the second.

A model of the Min-protein system. As a rather challenging benchmark we used a model of a Min-protein system in a three-dimensional model of an *E. Coli* bacterium [9]. The model contains five chemical species interacting in a system of five reactions. The geometry is pill-shaped, resulting from the union of a cylinder with two spheres (Figure 1). The complete set of reaction- and diffusion-rates can be found in [9], and the model is also available for download in the current release of URDME [Anonymous, 2012].We simulated the model at two different mesh resolutions, hence at two different ratios of reaction to diffusion events since the diffusion rate is inversely proportional to the square of the subvolume length. The *coarse mesh* (Figure 1A) contained 1555 subvolumes and the D:R was approximately 250. In the *fine mesh* (Figure 1B) the system consisted of 13307 subvolumes, and the D:R was about 1400.

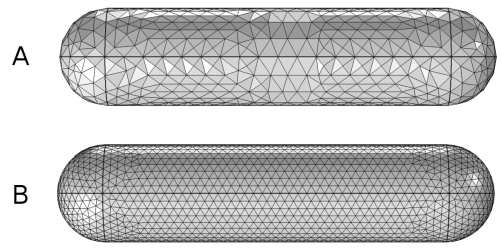


Figure 1: The spatial discretization of the *E. coli* bacterium geometry; coarse-grained (A) and fine-grained (B) tetrahedral meshes.

5.2 Scalability

In this section, we evaluate how the simulator performance scales with increasing LP count, and how the scaling depends on the particular model. In order to relate the models better to the measured performance we identify four potential performance indicators:

- *Subvolume count*: The number of subvolumes in the model.
- *The diffusion to reaction ratio (D:R)*: The ratio of simulated diffusion to reaction events.
- *Average degree*: The average number of neighbors of each LP.
- *Inter-LP diffusion ratio (Inter-LPD)*: The number of diffusions crossing subdomain boundaries over the total number of diffusions. We also tried including the reactions into the ratio, but this yielded a worse indicator. We discuss the impact of reactions separately under the D:R.

In Table 1 we present an overview of the benchmark configurations together with the introduced indicators. As the indicators *inter-LP diffusion ratio* and the *average degree* depend on the number of partitions, the values are listed for the partitioning to 16, 32 and 64 subdomains.

We furthermore list the sequential and parallel *simulation time* measured for all model configurations. Note that to measure the sequential time we used the sequential version of the algorithm (AEM), thus no parallelization overhead is included in the measurement. Moreover, the simulated time range was freely varied for each configuration, thus no direct relationship exists between simulation times shown in different rows.

Lastly, we list the *parallel efficiency* for all experiments and the same set of partitions. The parallel efficiency calculates as $T_1(T_N * N)^{-1}$, where T_1 is the sequential simulation time and T_N the parallel simulation time using N LPs.

We investigated the relationship of the introduced indicators to the measured parallel efficiency. To study the influence of the *inter-LP diffusion ratio* (inter-LPD) we observe the scaling of the rod, disc and sphere models at the [d1] configuration shown in Figures 2a, 2b and 2c. We see that large models (v100) scale significantly better than models of medium (v10) and small size (v1). As shown in Table 1, a large model size leads to a high private work-load per LP and thus a low inter-LPD. Furthermore, large models with a lower inter-LPD (e.g., rod) achieve a higher parallel efficiency than models with a higher inter-LPD (e.g., sphere). Hence, we find that the inter-LPD is an accurate indicator for the parallel performance of our simulator.

To study the impact of an increasing *diffusion to reaction ratio* (D:R) we present the scaling of the sphere model at configurations [d1] and [d100] shown in Figure 4. Here we find that the difference in parallel performance due to the increased D:R is small ($< 10\%$). Furthermore, for large models (v100) we observe that the parallel performance is independent of the D:R, as shown in Table 1. This is an unexpected finding, as we assumed that the D:R has a stronger influence on the scaling due to its effect on private workload.

To study how the parallel efficiency depends on the *average degree* in isolation, we compare different configurations of

Model	Conf.	#Subvol.	D:R	Avg. Degree			Inter-LPD.%			Time [s]				Efficiency		
				16	32	64	16	32	64	Seq.	16	32	64	16	32	64
Sphere	[v1,d1]	1437	1	7.6	8.8	10.3	24	32	42	198.7	34.7	29.9	21.9	0.36	0.21	0.14
	[v1,d100]	1437	105	7.6	8.8	10.3	24	32	42	216.4	45.5	40.5	37.9	0.3	0.17	0.09
	[v10,d1]	13575	1	7.8	8.8	10.8	12	15	21	259.8	37.7	23.1	13.7	0.43	0.35	0.3
	[v10,d100]	13575	107	7.8	8.8	10.8	12	15	21	293.7	43.3	28.2	17.1	0.42	0.33	0.27
	[v100,d1]	135228	1	7.9	9.8	10.8	6	8	10	545.1	64.1	31.1	12.6	0.53	0.55	0.68
	[v100,d100]	135228	109	7.9	9.8	10.8	6	8	10	476.1	53.3	24.5	11.6	0.56	0.61	0.64
Disc	[v1,d1]	1555	1	4.1	5	5.7	15	22	33	186.9	31.3	23.7	23.1	0.37	0.25	0.13
	[v1,d100]	1555	91	4.1	5	5.7	15	22	33	190.2	33.5	26.8	30	0.36	0.22	0.1
	[v10,d1]	13452	1	4.4	4.7	5.2	5	8	11	203.4	27.3	14.9	8.8	0.47	0.43	0.36
	[v10,d100]	13452	85	4.4	4.7	5.2	5	8	11	204.4	26.8	15.9	9.7	0.48	0.4	0.33
	[v100,d1]	125537	1	4.2	4.6	5.1	2	3	4	376.9	45.7	20.1	7.4	0.52	0.59	0.8
	[v100,d100]	125537	82	4.2	4.6	5.1	2	3	4	282.7	34.2	14.9	5.5	0.52	0.59	0.8
Rod	[v1,d1]	1429	1	1.9	1.9	2.8	13	27	54	174.7	27.1	22.9	27.1	0.4	0.24	0.1
	[v1,d100]	1429	90	1.9	1.9	2.8	13	27	54	177.6	31.1	30.1	33.8	0.36	0.19	0.08
	[v10,d1]	14000	1	1.9	1.9	2	1	2	5	224.2	28	14.7	8.3	0.5	0.48	0.42
	[v10,d100]	14000	90	1.9	1.9	2	1	2	5	232.6	27.7	15.6	9.3	0.53	0.47	0.39
	[v100,d1]	139139	1	1.9	1.9	2	0	0	0	325.5	40.1	18	6.4	0.51	0.56	0.79
	[v100,d100]	139139	91	1.9	1.9	2	0	0	0	357	42.2	19.7	6.9	0.53	0.57	0.81
Pred.-Prey	[n64,d1]	4096	1	4.1	4.7	4.9	6	10	14	203.5	29.2	22.8	12.2	0.44	0.28	0.26
	[n64,d2]	4096	2	4.1	4.7	4.9	6	10	14	324.6	51.6	41.2	33.4	0.39	0.25	0.15
	[n200,d1]	40000	1	4.1	4.6	5	2	3	5	371.2	44.2	22.7	11	0.53	0.51	0.53
	[n200,d2]	40000	2	4.1	4.6	5	2	3	5	592.6	72.3	34.6	20.9	0.51	0.54	0.44
	[n400,d1]	160000	1	4	4.6	5.1	1	2	2	286.1	31.5	13.8	5.8	0.57	0.65	0.78
	[n400,d2]	160000	2	4	4.6	5.1	1	2	2	387.1	50.5	22.2	9.1	0.48	0.54	0.67
Min-System	[coarse,–]	1555	304	5.2	7.5	8.7	20	27	38	126	29.7	23	24	0.27	0.17	0.08
	[fine,–]	13307	1517	4.9	7.3	8.9	10	14	20	539.9	80.4	53.1	34.1	0.42	0.32	0.25

Table 1: Overview of benchmark characteristics and results.

geometries with the same Inter-LPD. Namely, the models disc [v10,d1] and sphere [v100,d1], both of which have a Inter-LPD of 8% at the partitioning on 32 LPs. We find that the sphere model has a higher average degree than the disc model and the parallel efficiency is likewise increased. Nonetheless, as the models are of different subvolume sizes, we can not rule out the influence of unknown factors that correlate with the average degree.

Lastly, we observe the effect of the *subvolume count* indicator. It can be seen in Table 1 that a correlation with the Inter-LPD and thus the parallel efficiency exist. Furthermore, the efficiency for simulation of large models (v100) increases at increasing LP-count, which is not the case for small or medium size models. We suspect that this outcome is attributable to cache effects, as the partitioned model may fit better into core-local cache levels.

To visualize the correlation of the the *inter-LP diffusion ratio* and *subvolume count* indicators to the parallel efficiency we applied statistical curve fitting to the data for all [d1] models simulated on 64 LPs, as shown in Table 1. In Figure 3 we see the inter-LPD to parallel efficiency data fitted with a negative exponential function, and the subvolume count to parallel efficiency correlation fitted by a log-linear relationship.

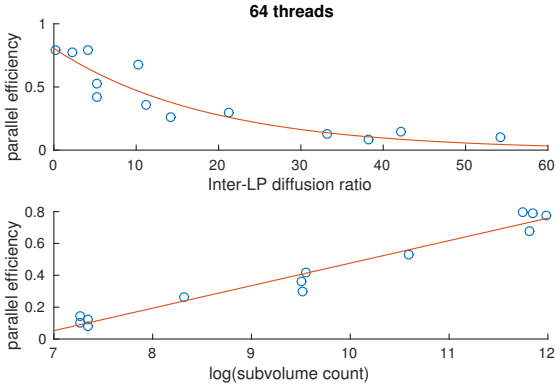
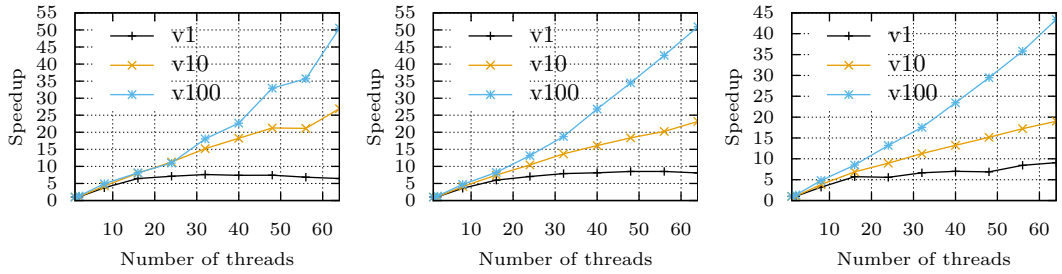


Figure 3: Curve fitting of the LPD and subvolume-count indicators to the parallel efficiency for all [d1] models simulated on 64 threads.

5.3 Detailed Behavior

In this section, we study in detail how the effort of the simulator is allocated. The DLTWEs were tuned to achieve the best performance for each model; hence the degree of optimism varies, and as a consequence the allocation of effort may be distributed differently. To measure the different parts of the effort, a lightweight instrumentation of the simulator was performed. The instrumentation allows us to break down the execution time into six parts of interest (line



(a) Rod[d1] (b) Disc[d1] (c) Sphere[d1]

Figure 2: Speedup for different configurations of the geometries rod, disc and sphere, the size is varied.

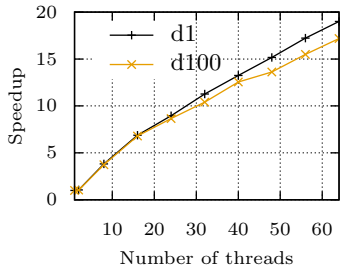


Figure 4: Speedup for the sphere[v10] model, D:R d1 and d100.

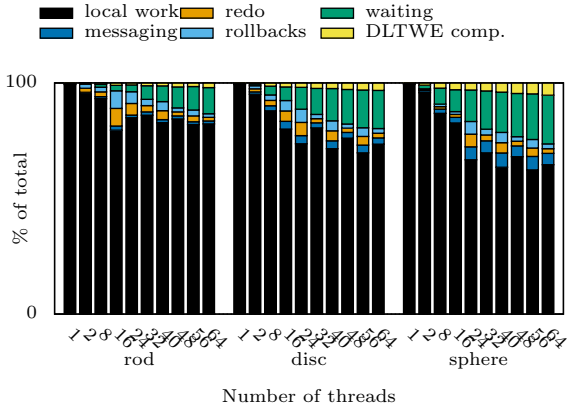


Figure 5: Breakdown of the execution time for the rod, disc and sphere models, size v100 and D:R d1.

numbers refer to Algorithm 1):

Waiting Time spent on blocking due to DLTWEs. (lines 15–17)

Rollbacks Time spent on processing anti-messages and rollbacks. (lines 8,9 and 20).

Redo Time spent on redoing work that has been undone by a preceding rollback. We estimate that the forward processing time is roughly equal to the backward processing time of an event, thus we estimate this value to be the same as Rollbacks.

Local work Time spent on processing of local events, other than events that could be attributed to Redo. (roughly lines 21–28, when e originates from the local event e_{local} at line 18)

Messaging Time spent on processing of diffusion messages other than anti-messages and messages that could be attributed to Redo. (roughly lines 21–28, when e originates from the message m at line 13)

DLTWE comp. Time spent on computing new DLTWEs, including scanning of the event queue. (line 30)

Of the above, Local work and Messaging are considered useful work, and the other parts are referred to as non-work.

The results of the breakdown analysis for the rod[d1], disc[d1] and sphere[d1] models, large size (v100), are shown in Figure 5. We see that the non-work part is completely dominated by blocking due to DLTWEs. Only a lesser part of the time is spent on rollbacks. Hence, the DLTWEs lead to a largely conservative execution for these models. For the sphere model, more time is spent on the processing of messages, in relation to the other models. This difference is explained by the increased connectivity of the sphere model, whose average degree is the double of that of the disc model, and five-fold in comparison to the rod.

In Figure 6, a corresponding breakdown analysis for the small models (v1) is shown. Here we see that a much larger portion of the non-work time is spent on rollbacks, for the disc and the sphere models. Apparently, for sufficiently small models, tuning the DLTWEs to allow for a more optimistic simulation is better. We also see that even more time is spent on processing messages, than in the case of the big (v100) models. Overall, more time is spent on parallel overhead, which is in line with our expectation, as the amount of private work is very small.

5.4 DLTWE Computation

In this section, we discuss how to tune the DLTWE computation, and we show how the selective rollback technique affects the performance in comparison to using non-selective

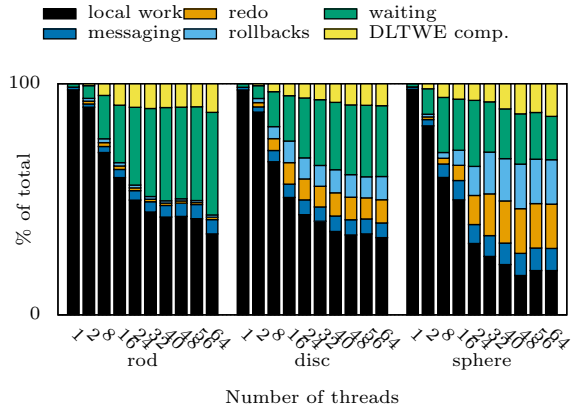


Figure 6: Breakdown of the execution time for the rod, disc and sphere models, size v1 and D:R d1.

rollbacks.

In our implementation, diffusion events are stored in a separate diffusion queue. The DLTWEs are produced by scanning the events in the diffusion queue (line 30 in Algorithm 1). The length of the prefix being scanned is a tunable parameter of our simulator, that affect the number of neighbors of each LP for which the DLTWEs are updated. Scanning a longer prefix of the diffusion queue results in a greater fraction of the DLTWEs being updated, and thus a more conservative simulation; furthermore, it requires more effort to update the DLTWEs. Scanning a shorter prefix results in fewer DLTWEs being updated (and thus set to infinity), and a more optimistic simulation.

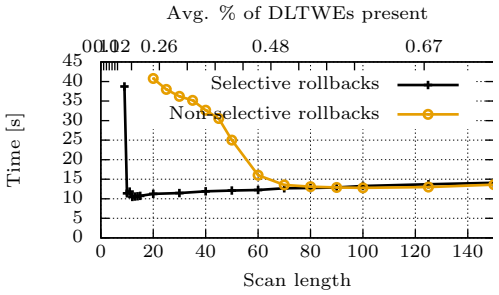


Figure 7: Total execution time for varying scan prefix lengths and average percentage of DLTWEs computed on the Predator-Prey[n200,d1] benchmark, on 64 threads.

We analyze how the performance depends on the length of the prefix, and also how this dependency is affected by the cost of rollbacks. For this, we run simulations under a range of prefix lengths, on the Predator-prey[n200,d1] model, both using the selective rollback technique and using the non-selective rollbacks. The results are displayed in Figure 7. On the x-axis above the plot, the performance is related to the average percentage of DLTWEs being updated. On the x-axis below, the performance is related to the scan prefix length. The execution time of the simulation using selective rollbacks and using non-selective rollbacks are shown for different lengths of the scan prefix. For decreasing length of the scan prefix, the performance of the non-selective rollback starts to decline when the percentage of the DLTWEs being updated passes below 50%. For the selective rollback, the performance does not start to decline until 5%.

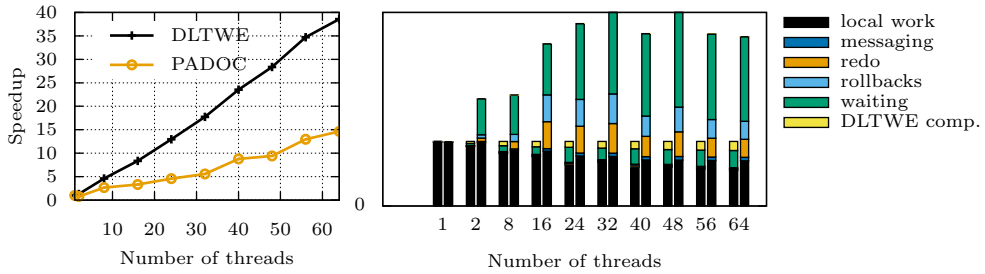
We see that in general the selective rollback technique always results in a superior or similar performance in comparison to the non-selective rollbacks. The optimal length of the scan prefix length for the two techniques are different, using selective rollbacks it is substantially shorter. This is because the effort of rollbacks is much smaller, and thus the performance improves, as optimism increases, even though the number of stragglers increase. We also see that in general, the best performance is achieved when quite a modest percentage of the DLTWEs between the LPs are known. A too conservative simulation is clearly not a winning option.

5.5 Comparison to other techniques.

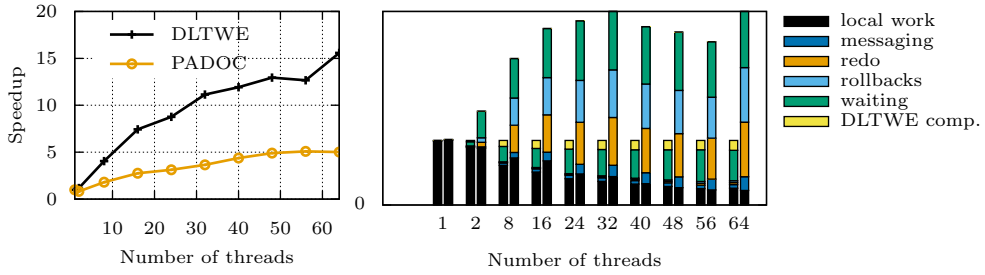
In this part we compare the DLTWE-synchronization technique to an adaptive protocol guided by the LP's local history, namely the Probabilistic Adaptive Direct Optimism Control (PADOC) proposed by Ferscha [10]. PADOC was implemented in our simulator, replacing the DLTWE synchronization. We have used non-selective rollbacks in this comparison, since it was more efficient when using the PADOC algorithm.

The PADOC algorithm relies on message arrival statistics that are continuously collected on each LP. At each advance of the local simulation time, the LP computes an estimate of the next message arrival time based on the statistics and the last arrival time. Depending on the distance from the current simulation time to the estimate, the LP decides to block for a constant amount of real time or or to proceed with optimistic execution of local events. To be exact, the decision is made by sampling of a sigmoidal probability density function described by a mean at the estimated future arrival time. The steepness of the probability distribution function is scaled with a constant in the range [0 1], where a value closer to 1 implies a stronger confidence in the estimator. In our experiments PADOC obtained the best performance at a scaling constant of 0.1. This suggest a large variance of the message arrival times in the simulations. We used the arithmetic mean as the estimator of message arrival statistics.

We evaluated PADOC on two benchmarks; the spatial predator and prey model at the [n400,d1] configuration, and the Min-system at the [fine] configuration. The speedup for both models simulated using the PADOC or DLTWE protocol is shown in Figure 8a, 8c. For both models, the DLTWE outperforms PADOC by a large margin. The breakdown of the execution time is shown in Figure 8b and 8d. For each LP count, the execution time is normalized to the DLTWE time, the left bar. We see that in general, DLTWE keeps the time spent on rollbacks at a very modest level. It should be noted that in the breakdown figures, the relative portion of the waiting time is slightly bigger for the DLTWE than if selective rollbacks would have been used. For PADOC, waiting for neighbors and performing rollbacks takes up a greater part of the total execution time. As the number of LPs increases, the failure to accurately predict arrivals of messages carries an increasingly significant cost.



(a) Speedup Predator-Prey (b) Breakdown Predator-Prey



(c) Speedup Min-System (d) Breakdown Min-System

Figure 8: Comparison of DLTWE and PADOC on the Predator-Prey[n400,d1] and Min-System[fine] models. In (a) and (c) the speedup of the DLTWE and the PADOC method is shown. In (b) and (d) a breakdown of how the time is spent is shown.

5.6 Relation to other works

In this section we discuss the performance of our simulator using the DLTWE technique in relation to other works. We would like to point out that it is difficult to make a fair comparison to other approaches, as previously used simulation algorithms and their implementations differ substantially from our approach. The single previously published RDME benchmark that can be found in the literature for the amount of LPs considered by us was the spatial predator and prey model presented by Wang et al. [30]. The parallel simulator used in the study is the Abstract Next Subvolume Method (ANSM), a distributed-memory implementation of the NSM using the Breathing Time-Warp protocol for synchronization. Our speedup and the ANSM speedup taken from [30], Figure 2b, are shown in Table 2.

Simulator (Protocol)	8 LPs	16 LPs	32 LPs	64 LPs
ANSM (BTW)	4x	6x	11x	20x
PAEM (DLTWE)	4.5x	8.4x	16.4x	33.9x

Table 2: Speedups obtained on the spatial predator prey model on a 200 x 200 grid using the ANSM and PAEM simulators.

6. CONCLUSION

We have presented a new technique for inter-LP synchronization in PDES. It is designed to be suitable when simulating models in which the time intervals between successive inter-LP events are highly variable and have no lower bounds, as in the spatial stochastic simulation that we have considered. Our DLTWE technique enables a detailed control of the amount of optimism in the simulation, which can be tuned to achieve desired accuracy of information communicated between LPs. We have shown how using a technique for selective rollbacks, the cost of optimism decreases, thus making it beneficial to allow for more optimism in the simulation.

With our implementation we have shown that the DLTWE technique is well suited to the setting of spatial stochastic simulations, and that it performs well on realistic problems in a shared memory environment. Notably, the DLTWE enables a parallel scaling which compares favorably to other inter-LP synchronization techniques described in the literature, as well as other parallelization efforts that have been reported in the literature.

7. REFERENCES

[1] R. Ayani and H. Rajaei. Parallel simulation using conservative time windows. In *24th WSC*, pages 709–717. ACM Press, 1992.

[2] D. Bernstein. Simulating mesoscopic reaction-diffusion systems using the gillespie algorithm. *Phys. Rev. E*, 71:041103, Apr 2005.

[3] G. Chen and B. Szymanski. Lookback: a new way of exploiting parallelism in discrete event simulation. In *PADS 2002*, pages 138–147, 2002.

[4] L. Chen, Y. Lu, Y. Yao, S. Peng, and L. Wu. A well-balanced time warp system on multi-core environments. In *25th ACM PADS*. IEEE, 2011.

[5] S. H. D. Ball. The adaptive time-warp concurrency control algorithm. In *Proceedings of the SCS Multi-conference on Distributed Simulation*, 1990.

[6] S. R. Das. Adaptive protocols for parallel discrete event simulation. *J. Oper. Res. Soc.*, 51(4):385–394, Apr. 2000.

[7] L. Dematté and T. Mazza. On parallel stochastic simulation of diffusive systems. In *Computational Methods in Systems Biology*, Lecture Notes in Computer Science, pages 191–210. Springer, Jan 2008.

[8] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *Systems biology*, 1(2):230–236, Dec. 2004. PMID: 17051695.

[9] D. Fange and J. Elf. Noise-Induced Min Phenotypes in E. coli. *PLoS Comput. Biol.*, 2(6):e80, June 2006.

[10] A. Ferscha. Probabilistic adaptive direct optimism control in time warp. In *9th PADS*, pages 120–129. IEEE, 1995.

[11] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[12] C. W. Gardiner. *Handbook of stochastic methods for physics, chemistry, and the natural sciences*. Springer, 2007.

[13] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, Dec. 1977.

[14] S. Jafer, Q. Liu, and G. A. Wainer. Synchronization methods in parallel and distributed discrete-event simulation. *Simul. Model. Pract. Th.*, 30:54–73, 2013.

[15] D. R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, July 1985.

[16] M. Jeschke, R. Ewald, A. Park, R. Fujimoto, and A. M. Uhrmacher. A parallel and distributed discrete event approach for spatial cell-biological simulations. *SIGMETRICS Perform. Eval. Rev.*, 35(4):22–31, Mar. 2008.

[17] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, Dec. 1998.

[18] H. Leong, D. Agrawal, and J. Agre. Using message semantics to reduce rollback in the time warp mechanism. In *Distributed Algorithms*, volume 725 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 1993.

[19] J. Liu and D. M. Nicol. Lookahead revisited in wireless network simulations. In *16th PADS*, pages 79–88. IEEE, 2002.

[20] B. D. Lubachevsky. Efficient parallel simulations of dynamic ising spin systems. *J. Comput. Phys.*, 75(1):103–122, Mar. 1988.

[21] J. Misra. Distributed discrete-event simulation. *ACM Comput. Surv.*, 18(1):39–65, 1986.

[22] A. Pellegrini and F. Quaglia. Transparent multi-core speculative parallelization of DES models with event and cross-state dependencies. In *SIGSIM-PADS'14*, pages 105–116. ACM, 2014.

[23] A. Pellegrini, R. Vitali, S. Peluso, and F. Quaglia. Transparent and efficient shared-state management for optimistic simulations on multi-core machines. In *MASCOTS'14*, pages 134–141. IEEE, 2012.

[24] M. Rathinam, P. Sheppard, and M. Khammash. Efficient computation of parameter sensitivities of discrete stochastic chemical reaction networks. *J. Chem. Phys.*, 132(3):034103–034103–13, 2010.

[25] P. L. Reiher, F. Wieland, and D. Jefferson. Limitation of optimism in the time warp operating system. In *21st WSC*, WSC '89, pages 765–770. ACM, 1989.

[26] R. B. Schinazi. Predator-prey and host-parasite spatial stochastic models. *Ann. Appl. Probab.*, 7(1):1–9, Feb. 1997.

[27] L. M. Sokol, D. P. Briscoe, and A. P. Wieland. Mtw: A strategy for scheduling discrete simulation events for concurrent execution. In *SCS Multiconference on Distributed Simulation*, pages 34–44, 1988.

[28] S. Srinivasan and P. F. R. Jr. Elastic time. *ACM Trans. Model. Comput. Simul.*, 8(2):103–139, 1998.

[29] J. S. Steinman. Breathing time warp. PADS '93, pages 109–118. ACM, 1993.

[30] 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. *Comput. Biol. Chem.*, 35(3):193–198, June 2011.

[31] J. Wang, K. Bahulkar, D. Ponomarev, and N. B. Abu-Ghazaleh. Can PDES scale in environments with heterogeneous delays? In *SIGSIM-PADS'13*, pages 35–46. ACM, 2013.

[32] J. Wang, D. Jagtap, N. B. Abu-Ghazaleh, and D. Ponomarev. Parallel discrete event simulation for multi-core systems: Analysis and optimization. *IEEE Trans. Parallel Distrib. Syst.*, 25(6):1574–1584, 2014.