Exposing Inter-Process Information for Efficient Parallel Discrete Event Simulation of Spatial Stochastic Systems

Jonatan Lindén, Pavol Bauer, Stefan Engblom and Bengt Jonsson
Dept. of Information Technology, Uppsala University
{jonatan.linden,pavol.bauer,stefane,bengt}@it.uu.se

ABSTRACT

We present a new efficient approach to the parallelization of discrete event simulators for multicore computers, which is based on exposing and disseminating essential information between processors. We aim specifically at simulation models with a spatial structure, where time intervals between successive events are highly variable and without lower bounds. In Parallel Discrete Event Simulation (PDES), the model is distributed onto parallel processes. A key challenge in PDES is that each process must continuously decide when to pause its local simulation in order to reduce the risk of expensive rollbacks caused by future "delayed" incoming events from other processes. A process could make such decisions optimally if it would know the timestamps of future incoming events. Unfortunately, this information is often not available in PDES algorithms. We present an approach to designing efficient PDES algorithms, in which an existing natural parallelization of PDES is restructured in order to expose and disseminate more precise information about future incoming events to each LP. We have implemented our approach in a parallel simulator for spatially extended Markovian processes, intended for simulating, e.g., chemical reactions, biological and epidemiological processes. On 32 cores, our implementation exhibits speedup that significantly outweighs the overhead incurred by the refinement. We also show that our resulting simulator is superior in performance to existing simulators for comparable models, achieving for 32 cores an average speedup of 20 relative to an efficient sequential implementation.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of simulation—*Discrete Event,Parallel*

Keywords

Parallel Discrete-Event Simulation, PDES, Optimism control, Multicore, Spatial Stochastic Simulation

SIGSIM-PADS '17, May 24–26, 2017, Singapore.
© 2017 ACM. ISBN 978-1-4503-4489-0/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3064911.3064916

1. INTRODUCTION

Discrete Event Simulation (DES) is an important tool in a wide-ranging area of applications, such as integrated circuit design, modeling of biological systems, epidemics, network analysis, etc. To improve performance and accommodate for large scale models, a vast repertoire of techniques have been developed for Parallel DES (PDES) during the last 30 years [12, 17, 24, 26]. New synchronization techniques have been triggered by the advent of multicore processors (e.g., [4, 27, 34]). Still, achieving good performance and speedup for larger number of processing elements has proven to be very difficult in the general case.

In PDES, the simulation model is partitioned onto *logical processes* (LPs), each of which processes timestamped events to evolve its partition along a local simulation time axis. Events that affect the state of neighboring LPs are exchanged to incorporate inter-LP dependencies. A major challenge in PDES is to ensure that each LP's processing of incoming events from other LPs is correctly interleaved with its local events, to guarantee that causally dependent events are processed in the right order. If an LP could know the timestamps and causal dependencies of future incoming events, it would be able to optimally advance its local simulation as far as possible, without encountering incoming events that arrive "too late" (so called *stragglers*), thereby violating its local timestamp ordering. Unfortunately, LPs do not in general have this information e.g., since there is no shared state, and since incoming events may result from still unprocessed events in other LPs. To handle this lack of information, several approaches have been developed: *conservative* approaches introduce additional synchronization which guarantees that no stragglers will arrive [26], possibly causing significant performance loss by excessive synchronization and blocking of local LP execution; *optimistic* approaches allow stragglers by invoking suitable corrective action (rollback) [17, 24], possibly damaging performance by excessive checkpointing and processing of rollbacks. Many intermediate techniques have been proposed that allow stragglers, but control the optimism by various heuristic techniques (see for example the surveys [5, 16]).

In this paper, we show that availability of precise information about timestamps of future incoming inter-LP events is a crucial building block for the design of efficient PDES algorithms. We show this by parallelizing a widely used algorithm for simulation of spatial stochastic systems in two steps: we first design a natural parallelization, in which each LP has access to modestly precise information about future incoming inter-LP events. We thereafter refine this natural parallelization algorithm, by restructuring each LP's simulation algorithm in a way which exposes more precise information about future inter-LP events. Both algorithms include a synchronization mechanism for disseminating the available information about future inter-LP events. Our experimental evaluation shows that the refinement increases the parallel efficiency to an extent that significantly outweighs its overhead. The refinement also results in an efficient parallelization, which outperforms other existing parallelizations of the original simulation algorithm.

We consider stochastic spatial simulation of models governed by the Reaction-Diffusion Master Equation (RDME) [13]. The RDME describes a spatial Markov process, where the spatial domain is discretized into *subvolumes*, also known as voxels, each containing discrete numbers of entities (e.g., proteins) that evolve by performing reactions local to a subvolume or diffusions to neighboring subvolumes. In general, each subvolume can host several types of reactions and diffusions with different combinations of entities. The inter-event times between reactions and diffusions are stochastic, highly variable and without a lower bound. Chains of events may propagate fast over the spatial domain, making parallelization particularly challenging.

The most important algorithm for simulating this class of models is the Next Subvolume Method (NSM) [9]. In the NSM, the event queue contains the timestamp for the next event of each subvolume, but does not say *which* reaction or diffusion will happen. In other words, the reactions and diffusions in a subvolume are aggregated into one *subvolume event*. Only when processing a subvolume event, it is determined (by a weighted coin toss) whether it is a reaction (and of which type) or a diffusion to some neighboring subvolume. The advantage of the NSM is that the event queue contains as many entries as there are subvolumes, thereby adding modestly to the memory requirements. The NSM algorithm has been used to study, e.g., protein fluctuations taking part in cell division [10], regulatory processes relevant for differentiation of stem cells [32], or the polarization of yeast cells [20].

A natural parallelization of NSM, here called *Direct Parallel NSM* (Direct PNSM), has been proposed in, e.g., [7,18]. It partitions the subvolumes and event queue onto LPs. The event queue of each LP represents the timestamp of the next event in each of its subvolumes. However, since the event queue does not say which type of event will be processed, the timestamp of the next diffusion to a particular neighboring LP is not represented, making it hard to communicate precise information for optimism control.

In this paper, we therefore propose a refinement of Direct PNSM, called *Refined PNSM*, motivated by the need to disseminate accurate information about timestamps of future inter-LP diffusions. Refined PNSM differs from Direct PNSM in that each LP explicitly keeps the outgoing inter-LP diffusions to each neighboring LP separate. That is, outgoing inter-LP diffusions are *not* included in the aggregated subvolume event of their respective subvolume. Instead, each LP maintains, for each of its neighboring LPs, a separate event queue for outgoing diffusions to that neighbor, which explicitly represents the timestamp of the next occurrence of each outgoing diffusion. This allows the LP to disseminate precise information about timestamps of outgoing inter-LP diffusion events to neighboring LPs. For the dissemination of information from these diffusion queues, we use the *Dynamic Local Time Window Estimates* (DLTWE) technique developed in our previous work [2]. A disadvantage of Refined PNSM is that more memory is required for representing the diffusion queues, and that effort is required to update the DLTWE estimates from these queues.

The DLTWE method for synchronization between LPs in our Refined PNSM is a further development of the method introduced in our previous work [2]. There it was used to parallelize the All Events Method (AEM) [1], which is an alternative to NSM for simulation of RDME models. The AEM does not aggregate reactions and diffusions in a subvolume, but maintains the next timestamp of all reactions and diffusions of the subvolumes in the event queue. The advantage of AEM is that it can be used for parameter sensitivity estimation, but its disadvantage is that the event queues require large amounts of memory and significant effort for maintaining them, causing sequential AEM to be significantly slower than sequential NSM. This disadvantage also applies to the parallelization of AEM. Therefore, parallel AEM achieves poor speedup relative to sequential NSM. However, its speedup over sequential AEM is quite good, due to the availability of precise information about timestamps of future inter-LP diffusions, as we showed in our previous work [2]. More details on the comparison with parallel AEM is found in Section 5.7.

We have implemented Refined PNSM with optimism control based on dissemination of information in its diffusion queues. We have also implemented Direct PNSM, both in a purely optimistic version without optimism control, as well as with optimism control based on the information available in its event queues: we have spent significant effort to investigate how to best use this information. The algorithms are compared on a representative set of benchmarks comprising unstructured and structured meshes. On 32 cores, Refined PNSM achieves an efficiency ranging between 50–103% for large models, and in average 35% for small models, compared to an efficient sequential simulation without any code for parallelization. In comparison to the Direct PNSM (with optimism control), the Refined PNSM shows an increase in efficiency between 24–84% for the large models. A detailed analysis of our optimism control in the Refined PNSM shows that rollbacks are almost eliminated, and that the amount of blocking is modest.

We also compare our resulting simulator, based on Refined PNSM, to other existing parallel simulators for the RDME that have been reported in the literature. We show that our simulator is superior in performance. E.g., in comparison to the simulator reported by Wang et.al. [33], we achieve a performance speedup on 32 cores which is approx. 63% better than theirs.

In summary, the contributions of this paper include:

- a methodology for parallelizing simulation algorithms based on the insight that availability and dissemination of precise information about future inter-LP events is crucial for efficiency,
- an efficient parallelization of the NSM simulation algorithm, which is superior in performance to other simulators reported in the literature, and
- experimental evidence that restructuring a simulation algorithm to expose precise information about future inter-LP events increases the parallel efficiency to an extent that significantly outweighs its overhead.

The work of this paper is to be integrated into the upcoming version 1.3 of the Unstructured RDME simulation framework (URDME) [8].

Organization of Paper The next section reviews related work. In Section 3, we describe the stochastic simulation framework at which our work is aimed, and the NSM algorithm. In Section 4, we describe the parallelization of the NSM by Direct and Refined PNSM, and their respective optimism control techniques. In Section 4.4, the algorithm is explained in detail. The experimental evaluation is found in Section 5. There we tune the optimism control parameters for the Direct and Refined PNSM algorithms, and compare their performance. The performance of the Refined PNSM algorithm is also compared to that of other comparable works. Section 6 contains the conclusions.

2. RELATED WORK

Numerous methods for parallelization of discrete event simulation (PDES) have been proposed. Here we will only review a selection and refer to the comprehensive surveys in [5, 12, 16].

Approaches to parallelization in PDES are coarsely classified into *conservative* [26] and *optimistic* [17,24]. Optimistic approaches [17,24] have the potential to achieve a higher degree of parallelism, but performance may be reduced by excessive rollbacks. To limit the frequency of rollbacks, various optimism control techniques have been developed.

While we only consider shared memory, we find it instructive to look at methods designed for distributed systems as well. Several techniques let LPs regulate their event processing rate in response to various statistics, such as the frequency of rollbacks [28], or the expected timestamps of future incoming events as estimated from statistics of past incoming events [11]. Another idea is to employ moving constant size *time windows*, often computed using model-specific knowledge, that bound how far each LP can advance its local time (e.g., [22, 29, 31]). We employ a similar concept in our optimism control. However, whereas the mentioned approaches use static information about minimum sizes of time windows, based on, e.g., known delays in communication channels, our time windows change dynamically and are based on dynamic and continuously updated information from neighboring LPs, enabling more accurate optimism control, even in situations where there are no minimum static time windows.

A further development of these approaches is the class of "near-perfect" state information (NPSI) protocols, including the *elastic time algorithm* [30]. Here, the time window is based on GVT and information about future messages to neighboring LPs, which is computed and communicated over a special high-speed network. Our optimism control can be seen as a refinement of this approach, where accurate information on future inter-LP messages is disseminated and used by neighboring LPs. We also realize this idea on a modern multicore without using a dedicated high-speed network.

PDES on multicores have gained increased attention lately. As an example, load balancing can be improved on multicores by allowing subdomains to be globally accessible by all cores (e.g., [4,27]). This adds cost for synchronizing data accesses across cores. Marziale et al. [25] tries to remedy the cost of inter-core accesses by grouping domains of a certain granularity to one single LP. Lin et al. [21] employed a technique called Multi-Level Queuing, in order to minimize the communication latency among threads in a multicore RDME simulator. Wang et al. [34] present a NUMA-optimized version of the general optimistic simulator ROSS [3]. We believe that our simulator would also gain from additional memory optimizations, however such an improvement is orthogonal to what we present here.

Parallel simulation of RDME models using the Next Subvolume Method was previously addressed by [7, 18, 21, 33]. The simulators are implemented in MPI, where each LP simulates a subvolume [21,33] or a subdomain [7,18]. Dematté and Mazza [7] first proposed that optimistic PDES is favorable for solution of RDME models. Control of optimism was realized by a static time window [18] or Breathing Time Warp [33]. In Section 5.7, we compare our implementation to measurements reported by Wang et al. [33] and Lin et al. [21].

The synchronization method for optimism control of Refined PNSM is a further development of the method introduced in our previous work [2]. There it was used to parallelize the All Events Method (AEM) [1], which is an alternative to NSM for simulation of RDME models. The AEM does not aggregate reactions and diffusions in a subvolume, but maintains the next timestamp of all reactions and diffusions in the event queue. The advantage of AEM is that it can be used for parameter sensitivity estimation, but its disadvantage is that the event queues require large amounts of memory, causing sequential AEM to be significantly slower than NSM. For this reason, Parallel AEM achieves poor speedup relative sequential NSM. However, its speedup over sequential AEM is quite good, due to the availability of precise information about timestamps of future inter-LP events (see [2] and Section 5.7).

3. SPATIAL STOCHASTIC SIMULATION

The Reaction-Diffusion Master Equation (RDME) [13] describes systems where entities, called *species*, diffuse over a discretized space and may undergo transitions, or reactions, when in proximity to each other. The dynamics of the transitions are described by a spatially extended Markov process. The RDME is thus frequently used to model biological systems where the copy number (discrete count) of chemical species is low and discrete effects therefore play an important role.

The spatial domain is divided into subvolumes, each of which maintains the *copy number* of all participating species. The dynamics of the model is a continuous-time Markov chain over the state space, which consists of all copy numbers in all subvolumes. Two types of transitions are possible; (a) in a *reaction* a combination of chemical species residing in a subvolume reacts and produces a new combination within the same subvolume, and (b) in a *diffusion* a single entity of a chemical species moves to a neighboring subvolume. The next occurrence time for each transition is exponentially distributed with a rate that is proportional to the product of the copy numbers of the involved species, as given by the law of mass action.

Practically, the RDME is simulated using sampling methods, that produce single trajectories from the relevant probability space. Since the advent of the original algorithm, known as the Gillespie's Direct Method [15], numerous such sampling methods have been proposed. For spatial models, the most commonly used exact sampling method is the Next Subvolume Method [9] (NSM).

The NSM algorithm takes the form of standard DES, which proceeds by repeatedly *a*) selecting an event from the event queue, *b*) processing it by updating the simulation state, i.e., modifying the population in one or more subvolumes, and finally *c*) updating other scheduled events in the event queue. The last step consists of updating the next occurrence times of events whose rates depends on the copy numbers of the subvolumes modified in step *b*, and subsequently sorting the event queue. The new next occurrence times are obtained either by rescaling of the old occurrence time or by sampling [14].

An important property of the NSM is that the events in the event queue are *not* the next occurrence times of each reaction and diffusion within each subvolume. Instead, all reactions and diffusions within a subvolume are aggregated into a single subvolume event, whose rate is the sum of the individual rates of the aggregated reactions and diffusions. Thus, the event queue contains the next occurrence time of the next reaction or diffusion within each subvolume. Upon executing the aggregated subvolume event, a random draw decides *which* particular reaction or diffusion event occurred in that subvolume. This significantly reduces the size of the event queue and improves simulation efficiency.

4. PARALLELIZATION

In this section, we present our parallelization of NSM. In Section 4.1, we first present a straightforward parallelization, called Direct PNSM. Thereafter, in Section 4.2, we present a refined version of the Direct PNSM, the Refined PNSM. Direct and Refined PNSM both have parameters that can be tuned to control the optimism, as described in Section 4.3. Both methods are exact parallelizations of the NSM, i.e., no additional error in the solution is introduced.

4.1 Direct PNSM

The Direct Parallel NSM (Direct PNSM) is a straightforward parallelization of the NSM algorithm to optimistic PDES based on Time Warp [17]. The subvolumes of the simulation model are partitioned into approximately equally sized subdomains, each of which is assigned to an LP. Each LP simulates the dynamics of the local subdomain while maintaining three main data structures: (a) the *subdomain state*, i.e., for each subvolume, the copy number for each species, (b) a time-sorted *event queue*, containing the occurrence time of future events in its subdomain, and (c) a *rollback history*, which is a time-sorted sequence of events that have already been processed.

Each LP advances the simulation by finding the next event to process, either from the top of the event queue or from a message that has been received from another LP. In the NSM, a local event specifies the subvolume to be processed. A random draw then decides which individual transition occurs in the subvolume. Thereafter, the event is processed by (a) updating the states of affected subvolumes, (b) adding the event to the rollback history, and (c) if the event was taken from the local event queue, determining a new next occurrence time for it.

If the event is a diffusion to another LP, a timestamped message is transmitted to the neighbor. In practice, the information is written into a bidirectional FIFO channel. Upon arrival, the message must be processed in the correct temporal order with respect to the local events of the receiving LP.

Whenever an LP receives a diffusion message that causes a causality violation (wrong temporal order of event updates), a so-called *straggler*, it must perform a rollback to a local time before the straggler's timestamp, using information from the rollback history. To do so, events are processed "backwards" until such a previous time is reached. In addition, all diffusion messages that have been sent by the LP during the rollback interval must be undone by sending *anti-messages* to the corresponding LPs. An anti-message cancels any event that was sent earlier with the same or a later timestamp. Anti-messages can cause cascading rollbacks that may involve several LPs, and are costly to resolve. In our implementations we use a refinement of the rollback technique, called *selective rollback* [2]. It is an adaptation of the breadth-first rollback mechanism [6], and prescribes that an LP that receives a straggler or an anti-message reverts only the events that are causally dependent on the received straggler or anti-message.

Rollbacks are undesirable, as the processing of rollbacks degrades performance. Hence, an LP should ideally not advance its local simulation time past the timestamp of a diffusion message that will be received in the future. For this purpose, we would like to design an optimism control strategy. Such a strategy involves the computation of *time window estimates*, which represent when the next diffusion message will be sent to a given neighbor, based on some available information in the inter-LP diffusion process. These estimates are then communicated to the respective neighbors via shared variables. The receiving neighbor is then able to use the estimates to derive a bound on its local simulation. If the timestamp of the next local event is larger than this bound, the neighbor blocks the execution, thereby causing waiting time, but reducing the risk of rollbacks.

However, it is not feasible to construct an accurate estimator of future inter-LP diffusion times in the Direct PNSM, since the method aggregates inter-LP diffusions into subvolume events. In a nutshell, the problem can be described as follows. Consider a subvolume on the boundary of an LP, which contains a number of intra-LP and a number of inter-LP diffusions, as well as a number of reactions. Since the NSM is used, the next occurrence time of the subvolume specifies solely when the next event in the subvolume occurs, without specifying which of the reaction or diffusion events it will be. Which event precisely occurs is determined only when the next occurrence time is reached, thus the simulator can not know the time of a specific inter-LP diffusion event in advance.

As the model dynamics is stochastic and variable in time, the best alternative for deriving time window estimates in the Direct PNSM algorithm is to compute the expected time of the next diffusion event from the inter-LP diffusion rates and the current local simulation time. The estimates are communicated to neighbors and updated at each simulator loop. We refer to this best effort technique as *Time Window Estimates*(TWE) in the evaluation. The estimates can be tuned with a scaling parameter, as described in Section 4.3.

4.2 Refined PNSM

To alleviate the problem of limited information about future messages in the Direct PNSM, we have developed a refined approach, dubbed Refined PNSM. It is an extension

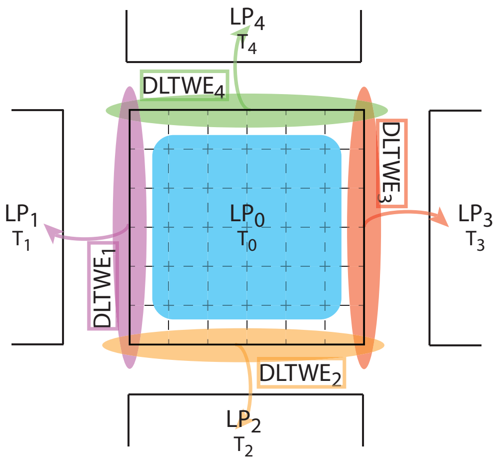


Figure 1: Schematic drawing of the Refined PNSM algorithm. The domain is divided into several LPs (solid lines), each operating at a local simulation time T_i. Each LP contains a number of subvolumes (dashed lines), within which a set of reactions and diffusions take place. We store the future scheduled events within the LP in different event queues. All reactions and internal diffusions (blue overlay) are stored in a local event queue. The local event queue contains aggregated next occurrence times for each subvolume. All outgoing inter-LP diffusions (red, green, cyan and yellow overlay) are stored in inter-LP queues, one per neighbor. The inter-LP event queues contain explicit next occurrence times and are used to compute DLTWEs. The DLTWEs are communicated to the LP's neighbors, which use them to derive a bound on their local simulation. A straightforward way to use the bounds are to advance the local simulation if the resulting local simulation time satisfies $T_i < DLTWE_i$, otherwise block the execution in order to minimize the risk of rollbacks.

to the direct approach. We propose that outgoing inter-LP diffusions are not contained in aggregated subvolume events, but instead form separate events with an explicit next occurrence time. These timestamps are accurate predictors of inter-LP diffusion events and can be used for optimism control. LP-local events are still aggregated per subvolume, which is more memory efficient.

In practice, within each LP we create (i) one event queue containing the aggregated subvolume events, representing reactions and intra-LP diffusions, and (ii) for each of its neighbors an event queue containing all outgoing inter-LP diffusions to that neighbor (see Figure 1).

The prediction of future inter-LP messages in the Refined PNSM is straightforward. The next occurrence times of the outgoing inter-LP diffusions are extracted from the top of the inter-LP queues. Then, they are communicated to each neighbor as a time window estimate, and updated whenever the top entry of the queue changes. This is essentially the Dynamic Local Time Window Estimate (DLTWE) technique, introduced in our previous work [2].

The DLTWEs are accurate predictors of future inter-LP diffusions, which a receiver may use to significantly reduce the amount of rollbacks. In contrast to the Direct PNSM, they accurately predict the times at which the inter-LP event will occur. However, rollbacks cannot be completely ruled out. The estimates are updated frequently due to state changes in the model, and may be updated to an earlier time as well. Thus, a neighbor may read an estimate, decide that it is safe to progress with local simulation, upon which the estimate is updated to an earlier time. The corresponding message from which the estimate was derived may then cause a rollback at arrival.

4.3 Tuning of Optimism Control

Typically, adaptive PDES can be tuned for optimal performance. In particular, the goal is to find the optimal trade-off between the cost of over-optimism ("too many rollbacks") against the cost of lost opportunity ("too much waiting time").

For the Direct PNSM and Refined PNSM, this tuning is given in terms of a parameter n, which defines how many of the received time window estimates should be used to compute a bound for the local simulation. Our experiments have shown that if all estimates are used, the result is an overly conservative simulation. By only considering a subset of the communicated estimates, the level of optimism can be increased. A further observation is that some neighbors may generate significantly more messages than others, thus provoking more rollbacks. The parameter n thus refers to the set of n neighbors, which have incurred the highest amount of rollbacks on the LP during the simulation. An exception is made for LPs with a few number of neighbors, where the DLTWEs of all neighbors are used.

Furthermore, in the Direct PNSM, the expected next diffusion time is also scaled with a tunable parameter $k \geq 0$. The communicated time window estimate then equals $t + \sigma \cdot k$, where t is the local simulation time of the sending LP and σ is the expected inter-event time to the next inter-LP diffusion event. A small k implies a larger cost of lost opportunity, while a large k leads to a larger amount of rollbacks.

4.4 Detailed Algorithm Description of the Refined PNSM

Table 1: Structure of an LP.

```
structure LP$_i$:
    $m$                          ▷ Number of neighbors
    $n$                          ▷ Optimism control parameter
    EventQueue[$0 \ldots m$]     ▷ Time-sorted queues
                                 ▷ of events
    SubvolumeState[$1 \ldots n_i$] ▷ State of subdomain
    History                      ▷ History of past events
    Channel[$1 \ldots m$]        ▷ Channels of incoming messages
    Dltwe[$1 \ldots m$]          ▷ Incoming DLTWEs
```

Below we outline the structure of an LP$_i$ with m neighbors. Each LP contains:

- *EventQueue*$[0 \ldots m]$ is an array of priority queues, containing scheduled events sorted by timestamp. Here, *EventQueue*$[1 \ldots m]$ are the inter-LP queues, each containing the diffusion events destined for a particular

Algorithm 1: Main loop of the Refined PNSM algorithm, executed by each LP_i.

```
while true do
    ▷ First phase: find the next event to process
    $e_{\mathrm{msg}} \leftarrow$ earliest message in $\{$ RetrieveMsg($Channel[j]$) $\mid 1 \le j \le m\}$ ▷ Peek at top of each message channel
    $e_{\mathrm{local}} \leftarrow$ earliest event in $\{\mathrm{peek}(EventQueue[j]) \mid 0 \le j \le m\}$
    if $e_{\mathrm{msg}}$.time $\le e_{\mathrm{local}}$.time then          ▷ If $e_{\mathrm{msg}}$ precedes any local event
        $e \leftarrow$ pop $e_{\mathrm{msg}}$ from its message channel  ▷ The event $e$ to be processed is from the incoming channels
    else
        HotDltwe $\leftarrow$ {indices of the $n$ channels having caused the most rollbacks}
        while $\exists j \in$ HotDltwe s.t. $Dltwe_j < e_{\mathrm{local}}$.time do ▷ If next event is later than any selected DLTWE
            if ($Channel[j]$ has message) or (for some $k \neq j, \exists e_m \in Channel[k]$ s.t. $e_m$.time $< Dltwe_j$) then
                restart main loop
        $e \leftarrow$ pop $e_{\mathrm{local}}$ from event queue       ▷ Otherwise the next event to be processed is local
    ▷ Second phase: process selected event
    if $e$.time $< SubvolumeState[e.\mathrm{dest}]$.time or $e$ is an anti-message then
        SelectiveRollback($e$)                  ▷ Revert state if $e$ is a straggler (which may be local) or an anti-message
        if $e$ is an anti-message then restart main loop
    add $e$ to $History$
    update $SubvolumeState[e.\mathrm{subvol}]$
    update timestamps of affected future reactions/diffusions in $EventQueue[0 \dots m]$
    if $e$ is a diffusion then
        if $e$.dest is local then
            update $SubvolumeState[e.\mathrm{dest}]$
        else
            send diffusion message to $\mathrm{LP}_j$ where $e.\mathrm{dest} \in \mathrm{LP}_j.SubvolumeState$
    for each neighbor $\mathrm{LP}_j$ do
        $\mathrm{LP}_j.Dltwe_i \leftarrow$ time of peek ($EventQueue[j]$) ▷ Update the DLTWEs of neighbors
```

neighbor, and *EventQueue*[0] is the intra-LP queue, containing subvolume events that aggregate reactions and diffusions within the LP;

- *SubvolumeState* is an array representing the state of each subvolume in the subdomain, i.e., the number of entities of each species, and the timestamp of the last event affecting the subvolume (i.e., each subvolume has an individual timestamp),
- *History* is a time-sorted sequence of events already processed by the LP; old events are regularly removed from the history by fossil collection,
- *Channel*$[1 \dots m]$ is an array of incoming message channels, one for each neighbor, and
- $Dltwe_{1 \dots m}$ is a set of incoming DLTWE estimates, one for each neighbor.

For an event e, we let e.time denote its timestamp; for a diffusion event e, we let e.dest denote its destination subvolume, which may reside in a different LP.

The main simulator loop consists of two phases, the selection of the next event to process, and the processing of the event. It has some similarities to the algorithm we previously presented in [2], where, in addition, the functions RetrieveMsg and SelectiveRollback are explained in more detail.

The first phase (lines 2–11), selects 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 anti-messages occurring in the channel have been processed. The earliest message is assigned to e_{msg}. Second, the earliest local event e_{local} from all event queues is read. If e_{msg} is earlier than e_{local} (line 4), then e_{msg} is assigned to e for processing. Otherwise, the event e_{local} is assigned to e for processing, but only if none of the n selected DLTWE estimates is violated (line 8). If a DLTWE estimate would be violated, the LP blocks until a message from the corresponding neighbor, or an earlier message from another neighbor, is received, at which time the main loop is restarted, in order to process the message (line 10).

The second phase (lines 12–24) updates the subdomain state of the LP by processing the event e that was selected in the first phase. If e is an aggregated event, the type of transition and the destination are resolved by a random draw. It starts by checking whether e is a straggler or a local diffusion which violates causality in its destination subvolume, or if it is an anti-message; in all three cases a rollback is necessary. The rollback is performed by the function SelectiveRollback(e) (line 13), which reverses the effect of all events, that are causally dependent on e. Thus, subvolumes may be rolled back to different times. The latter is also the reason why we might have rollbacks induced by other local events. In case the rollback performed was due to an anti-message, the main loop must restart (line 14), since the state of the subvolume of the currently selected event e may have changed. For a detailed description of the rollback function, we refer to our previous work [2].

After these checks, the selected event e is processed by adding it to the event history (line 15), 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 16 through 20). If e is a diffusion to another subdomain, a message is sent (line 22) to the appropriate LP. After that, the DLTWEs are updated (line 24) to inform the neighbors of the possibly new estimated times of the next diffusion events.

5. EVALUATION

In this section, we evaluate the efficiency and scalability of the Refined PNSM and the Direct PNSM algorithms. We look at the following five questions:

- *How do we tune the optimism control?* (Section 5.4)
- *How effective is the Refined PNSM algorithm?* (Section 5.5)
- *How big is the overhead of the Refined PNSM algorithm?* (Section 5.6)
- *How well does the Refined PNSM compare to other works?* (Section 5.7)
- *Which model parameters affect the performance?* (Section 5.8)

5.1 Algorithms

To evaluate and compare the Direct PNSM and the Refined PNSM we implemented both algorithms. For both implementations, the baseline is the sequential implementation of the NSM method used in the URDME simulation framework [8], that has been shown to be efficient in comparison to other NSM implementations [8, Suppl. data]. Thus, all implementations share the same code for the sequential logic, but are extended for parallel execution in different ways. The optimism control, as described in Sections 4.1 and 4.2, was made optional and tunable in both implementations. In the following, we refer to the different configurations as follows.

NSM The original sequential NSM simulator.

D-PNSM The Direct PNSM simulator, without optimism control.

D-PNSM[TWE] Identical to D-PNSM, but with approximative adaptive optimism control turned on, as defined in Section 4.1.

R-PNSM The Refined PNSM algorithm, without optimism control.

R-PNSM[DLTWE] Identical to R-PNSM, but with adaptive optimism control using DLTWEs turned on, as defined in Section 4.2.

The DLTWE technique requires detailed state information which is only available in the Refined PNSM algorithm, whereas the approximative TWE technique can directly use aggregated information, and therefore also works for the Direct PNSM algorithm.

To understand how the effort of the simulators is used, a fine-grained, low overhead (approx. 2%) instrumentation of the simulator was implemented, which records the time spent on different activities of interest. We categorize the activities of interest as follows (line numbers refer to Algorithm 1):

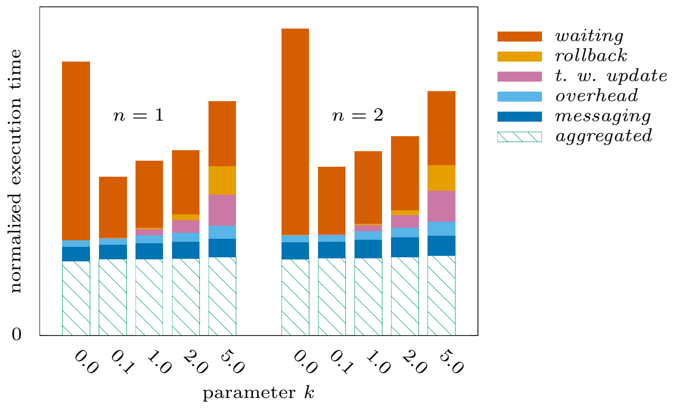


Figure 2: Instrumentation data for the D-PNSM[TWE] algorithm for 32 cores on the sphere[L,d10] model. Time is normalized to the first bar of each group of the parameter n.

aggregated Time spent on evaluating events from the queue with aggregated events. Corresponds to lines 15–24, if e is picked from the aggregated queue.

inter-LP Time spent on evaluating events from the inter-LP queues, lines 15–24, if e is picked from an inter-LP queue.

messaging Time spent on processing events from neighboring LPs, lines 15–24, if e is a received diffusion event.

overhead Time spent on updating the event history and fossil collection, that are directly related to the parallelization overhead. These are not detailed in Algorithm 1.

rollback Time spent in SELECTIVEROLLBACK, line 13, and time spent on re-simulating the time interval reverted by the rollback. The re-simulation cost is naively estimated to be equal to the cost of the rollback.

waiting Time spent on waiting for a neighboring LPs time window estimate, lines 8–10.

time window update Time spent on updating the time window in the D-PNSM[TWE] algorithm, as described in Section 4.1.

Of the above mentioned categories, *aggregated*, *inter-LP* and *messaging* represent useful work. The *rollback* cost is caused by over-optimism, and *waiting* is the cost caused by too much conservatism.

5.2 Benchmarks

We now give a brief description of the benchmarks considered in the evaluation. The benchmarks are constructed to represent a variety of realistic RDME model properties. The topology is varied, which has an impact on the connectivity of LPs, and the reaction to diffusion event ratio (D:R ratio) is varied, which has an impact on the amount of communication between LPs. A more detailed description of the benchmarks is available in our previous study [2].

(1) The *reversible isomerization* benchmark consists of spatial models defined on three different geometries; *sphere*,

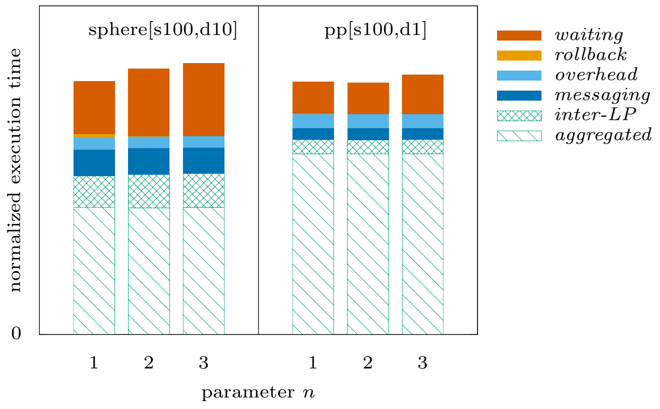


Figure 3: Instrumentation data for the R-PNSM[DLTWE] algorithm for 32 cores on the sphere[L,d10] and the pp[L,d1] models. Time is normalized to the first bar of each group per benchmark.

disc and *rod*. The geometries are discretized into three-dimensional unstructured meshes, leading to a high number of neighbors for each LP, except in the case of the rod model. For each shape, a model is generated in two different sizes of about 14.000 (*small*) and 140.000 (*large*) subvolumes. Each model contains two chemical species, A and B, which are allowed to freely diffuse within the domain with equal diffusion rates. The species may undergo the reversible reaction,

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

with the tunable reaction intensity c. To arrive at a specific D:R ratio, we have determined the parameter in preceding test runs. For each model size, we set the D:R ratio to either 1:1, 10:1, or 100:1, which are the magnitudes found in realistic models, as in [10, 32].

(2) The spatial *predator-prey* (denoted *pp* in the following) model was used as a benchmark in the study by Wang et al. [33]. In contrast to the previous models, the geometry is two-dimensional and the discretization is given by structured meshes, leading to a small number of neighbors for each LP. For the sake of comparison, we present the model at the same configurations as Wang et al., namely at a system size of 40.000 (*small*) and 160.000 (*large*) subvolumes and D:R ratios of 1:1 and 2:1.

We will denote specific model configurations as [s,dy], which means that the system size s is S (*small*) or L (*large*), and the D:R ratio is $y : 1$.

5.3 Experimental Setup

All experiments were run on a 4 socket Intel Sandy Bridge E5–4650 machine. Each processor has 8 cores and 20 MB L3-cache. Hyperthreading was not used, and threads were pinned to cores. The computer runs Linux 3.16.0, and the binaries were compiled using GCC 4.9.2. For the graphs, each data point is the average of three runs. The three-dimensional models were constructed using Comsol Multiphysics 4.3 and converted to computational models using the URDME framework. The two-dimensional structured meshes used in the spatial predator-prey model were constructed using custom Matlab scripts. All the meshes were then partitioned into subdomains using the multilevel k-way partitioning method provided by the Metis library [19]. Metis optimizes the partitioning for minimal number of diffusions crossing subdomain boundaries, while maintaining an equal number of subvolumes in each subdomain.

5.4 Tuning the Optimism Control

In this section we tune the parameters of the D-PNSM[TWE] and R-PNSM[DLTWE] algorithms, as described in detail in Section 4.3. For both algorithms, we tune how many of an LP's received time window estimates are used, denoted as parameter n. For the D-PNSM[TWE], we additionally tune the parameter k in the time window estimate, as described in Section 4.3.

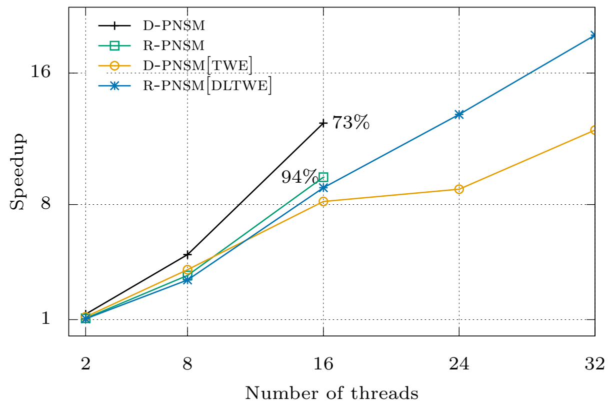
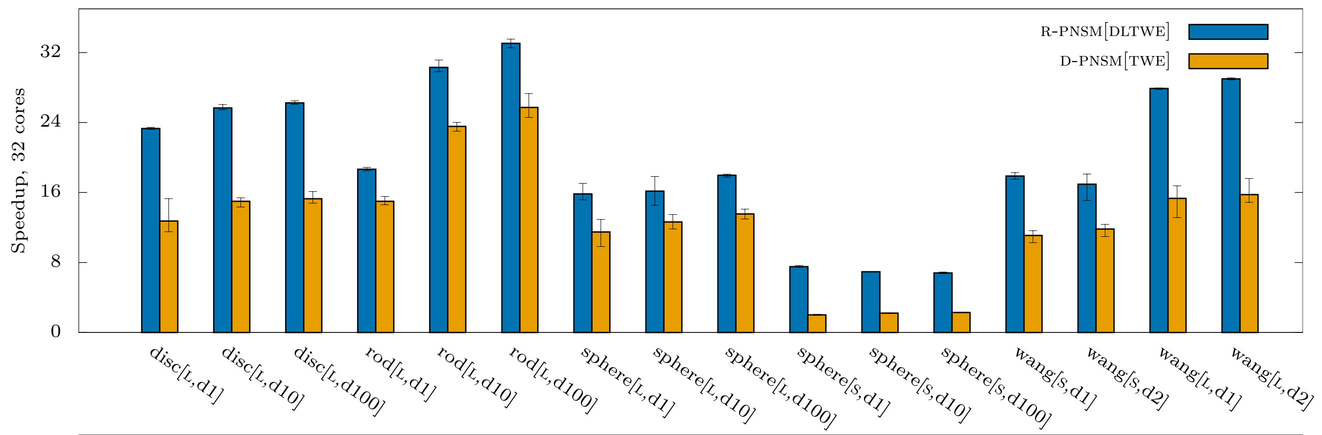


Figure 4: Speedup over NSM for the sphere[L,d10] model. The D-PNSM and R-PNSM did not complete for more than 16 cores due to rollback explosions. For 16 cores, the success rate of the D-PNSM is approx. 73%, and for the R-PNSM it is approx. 94%.

Figure 2 shows the time breakdown of the D-PNSM[TWE] algorithm, for different values of k and n in the sphere[L,d10] model. Each bar shows the time breakdown of a simulation with 32 cores and a combination of values of k and n. We find that the optimal values of k and n are at 0.1 and 1, respectively. The same values were found to be optimal for other benchmarks as well. In the figure, we also observe how the cost of rollbacks increases for bigger k, i.e., when communicating a bigger time window estimate.

Figure 3 shows the effort breakdown of the R-PNSM[DLTWE] algorithm for different values of the parameter n, evaluated on the sphere[L,d10] and the pp[L,d1] models. We see that the best performance is achieved for $n = 1$, which we have also confirmed for several other benchmarks, not shown. Thus, in general, the best performance is achieved by observing the timestamps of future incoming diffusion events from only a single neighbor, viz., the one producing the most rollbacks.

To understand why it suffices to wait on a single LP, we recall that the goal of controlling optimism is to make each LP progress at the same "speed", i.e., no LP should advance the simulation time faster than the slowest LP, roughly expressed. Thus, intuitively, it should suffice that each LP wait for a single slower LP, creating chains of waits-on dependencies, typically rooted at the slowest LP. All dependency chains may not necessarily be connected, in which case they progress at the same speed. Now, increasing n, would add more waits-on relations than necessary, and thus



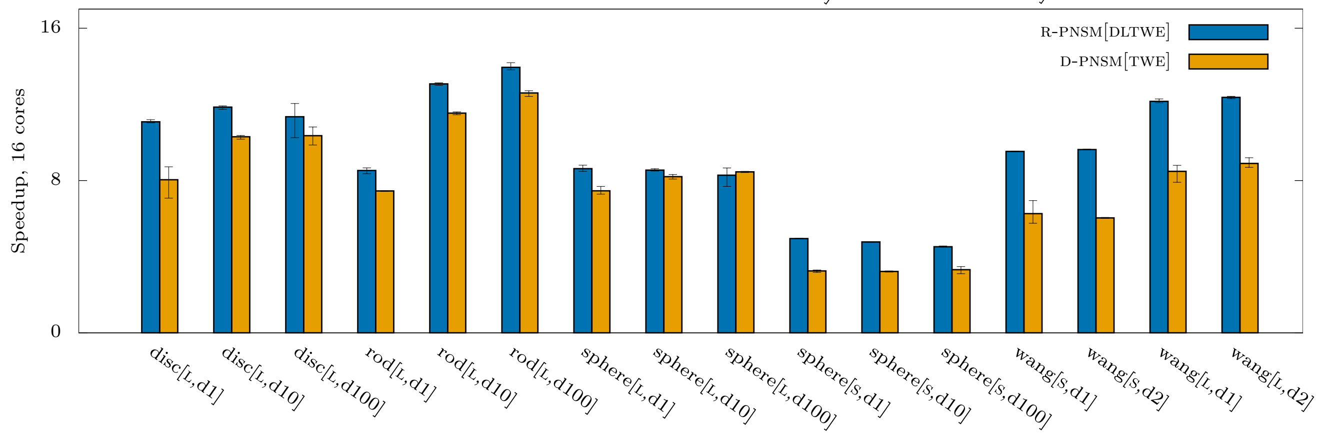


Figure 5: Speedup over NSM, 32 cores (top) and 16 cores (bottom), for all benchmarks.

increase the overhead. Another factor is that increasing n makes the simulation more rigid than necessary. Since the selective rollback technique is used, a straggler may not necessarily generate rollbacks, and hence it is good to allow a certain degree of out of order between LPs.

5.5 Scalability Comparison

In this section we compare the scaling of the two algorithms D-PNSM and R-PNSM, with and without optimism control.

In Figure 4, the speedup curves for the four parallel algorithms, evaluated on the sphere[L,d10] benchmark, are shown.

We would like to point out that the D-PNSM and the R-PNSM algorithms, i.e., the algorithms *without optimism control*, often suffer from rollback explosions, in which case the simulation does not finish within a reasonable amount of time (3 times the expected finishing time), or runs out of memory. In the cases the algorithms finish, the amount of rollbacks is relatively low. Similar behavior has previously been observed by others [23]. We have nevertheless chosen to include the data for the *successful runs only* for both the algorithms, as it is illustrative for the understanding of the performance of the algorithms. For the sphere[L,d10] benchmark, 16 cores, the success probability of the D-PNSM algorithm is 73%, and of the R-PNSM algorithm it is 94%. For 32 cores it is 0% for both the algorithms, clearly showing the necessity of using a mechanism for optimism control.

Now, comparing the speedup for the D-PNSM and the R-PNSM algorithms in Figure 4, we see that the D-PNSM exhibits 34% better performance than the R-PNSM, for 2–16 cores, considering only successful runs. The performance cost of the optimism control for the two algorithms R-PNSM[DLTWE] and D-PNSM[TWE] is also illustrated in the figure. For the R-PNSM[DLTWE] algorithm, the performance loss over the R-PNSM is marginal. This is because the time window calculation, in this case maintaining the inter-LP queues, is already included in the R-PNSM algorithm. For the D-PNSM[TWE], the performance loss over D-PNSM is 25% for 16 cores.

Comparing both algorithms with optimism control on the same benchmark, the D-PNSM[TWE] algorithm has better performance than the R-PNSM[DLTWE] when only a small number of cores are being used. For 8 cores, it is 16% faster. However, for more than 16 cores, the opposite holds, and the R-PNSM[DLTWE] is up to 51% faster. The R-PNSM[DLTWE] algorithm clearly scales better than the D-PNSM[TWE].

The overall performance of the R-PNSM[DLTWE] and the D-PNSM[TWE] algorithms is illustrated in Figure 5. For 32 cores, the R-PNSM[DLTWE] achieves an efficiency ranging between 50–103% on the large models, compared to NSM. Compared to the D-PNSM[TWE], we see a performance improvement in the range of 24–84%. For the smallest model, under several different diffusion ratios, the performance of the R-PNSM[DLTWE] is up to almost 4x better than the D-PNSM[TWE] algorithm. The average speedup of R-PNSM[DLTWE] over NSM for all benchmarks is 20.

On 16 cores, the performance improvement of the R-PNSM[DLTWE] over the D-PNSM[TWE] is in general smaller. The R-PNSM[DLTWE] algorithm has on average 23% better performance. We note that on the sphere[L,d100] benchmark, the D-PNSM[TWE] algorithm actually performs better.

We observe that for both algorithms, the unstructured models are in general more challenging than for example the *predator-prey* models, where the number of neighbors per LP is small. Likewise, the smaller models are more challenging, due to the limited amount of parallelism available. Especially for the smallest model, the performance difference of the D-PNSM[TWE] and the R-PNSM[DLTWE] is particularly accentuated. It also seems that the D-PNSM[TWE] algorithm performs better (in comparison to the R-PNSM[DLTWE]) in highly diffusive models.

5.6 Overhead of the Refined PNSM Algorithm

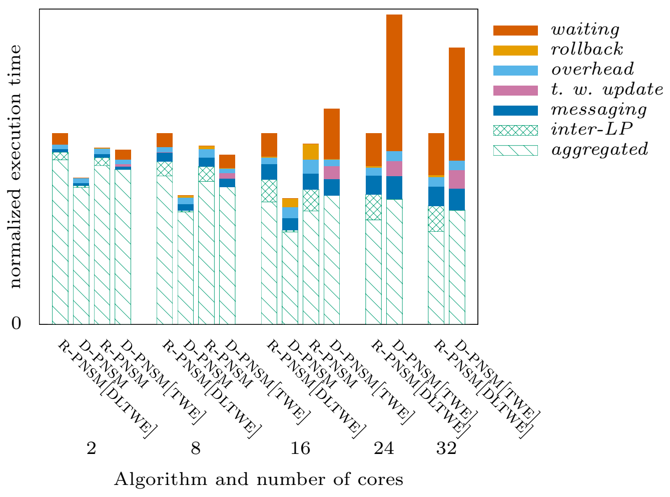


Figure 6: Instrumentation data for all algorithms on the sphere[L,d10] model. Time is normalized to the first bar of each group of cores. We recall from Figure 4, that the D-PNSM and R-PNSM did not complete for more than 16 cores due to rollback explosions. For 16 cores, the success rate of the D-PNSM is approx. 73%, and for the R-PNSM it is approx. 94%.

In this section we investigate the overhead of the R-PNSM algorithm over the sequential NSM and the D-PNSM algorithm, i.e., the cost of the parallelization and the cost of maintaining the inter-LP queues.

First, to estimate the parallelization overhead over NSM, we ran the D-PNSM sequentially (on a single LP) and compared it to the NSM, on the sphere[L,d10] benchmark. Note that, in the sequential case, the R-PNSM algorithm is identical to D-PNSM. The parallelization incurred an overhead of approx. 36%.

Next, in order to better understand the differences in Figure 4, we observe the instrumentation data of the same experiment, shown in Figure 6. The total height of each bar in the figure corresponds to their execution time. The execution time is normalized to that of the R-PNSM[DLTWE] for each number of cores. To estimate the overhead of the R-PNSM algorithm over the D-PNSM, we calculate the difference in the amount of effort spent on all activities except *waiting* and *rollback* for each algorithm, as defined in Section 5.1. We see that the R-PNSM exhibits a significant amount of overhead over the D-PNSM: 38% at 8 cores, and 40% at 16 cores.

As we observed in Figure 4, the D-PNSM[TWE] algorithm is faster than the R-PNSM[DLTWE] for 2 and 8 cores, but the R-PNSM[DLTWE] is faster for more than 16 cores. An explanation is provided: For the R-PNSM[DLTWE] the relative amount of effort not spent on useful work increases by 63% when going from 8 to 16 cores, due to waiting. For the D-PNSM[TWE], the same effort more than doubles. Similar values are found for any increase in the number of cores. Thus, the initial cost of the R-PNSM[DLTWE] is higher, but the rate at which the overhead increases for the R-PNSM[DLTWE] is lower than for the D-PNSM[TWE], making it scale better.

5.7 Comparison to Other Works

In this section, we compare the performance of the Refined PNSM algorithm to that of similar algorithms of other works. We have looked at all to us known relevant papers for comparable experiments; the only two previously published RDME models for which experiments with more than 12 cores have been reported are [33] and [21].

Wang et al. [33] simulate a predator-prey model on a two-dimensional 200×200 structured grid, using their parallel Abstract NSM algorithm. Though it is unknown to us which sequential baseline they use for the comparison, they report a speedup of 11 on 32 cores. For the same benchmark, we achieve a speedup of 17.9. Lin et al. [21] simulate a three-dimensional calcium wave model using the NTW-MT simulator. They report a speedup of 9 on 32 cores. The parameters for the model were not available to us, thus we could not test our simulator on the benchmark.

In our previous work [2], we presented the PAEM algorithm, a parallelization of the AEM algorithm which does not aggregate reactions and diffusions in a subvolume, but maintains the next timestamp of all reactions and diffusions in the event queue. The sequential AEM is significantly slower than NSM, largely due to high memory requirements. Compared to sequential AEM, the PAEM achieved a speedup of 16.4 on 32 cores, for the above benchmark from [33]. The speedup for PAEM over the NSM, however, is only 5.2 on the same benchmark, largely due to its poor memory efficiency.

5.8 Performance Indicators: Degree and Inter-LP Diffusion Ratio

In this section, we investigate the relation between the parallel performance and the model parameters. We investigate the impact of two performance indicators, namely

Degree The graph degree of the communication network between LPs.

Inter-LP Diffusion ratio The number of inter-LP diffusions over the total number of diffusions.

In Figure 7 we compare the benchmark results of three models on 32 cores. Only one of the performance indicators differ between each model. The *rod-sphere* benchmark is a modified version of the rod[L,d10] benchmark, with an inter-LP diffusion ratio that is equal to that of the sphere[L,d10] benchmark. All the three models have the same number of subvolumes, and the benchmarks take the same time to complete for the sequential simulator. We also want to ensure

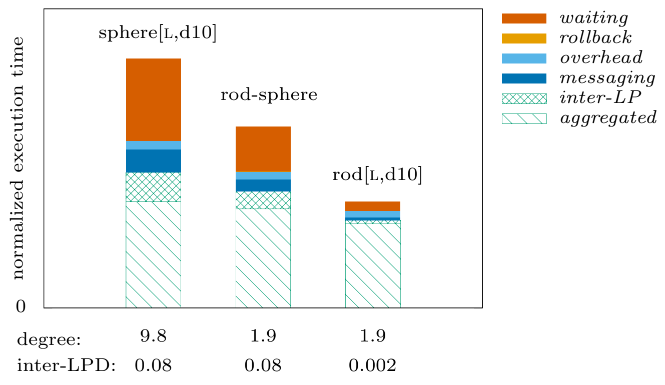


Figure 7: Instrumentation data of R-PNSM[DLTWE] using all DLTWEs, on the sphere[L,d10], the rod-sphere, and the rod[L,d10] models, 32 cores.

that the same number of inter-LP diffusions is used when deciding if an LP should block. Thus, we set the algorithm to use all the DLTWEs for this benchmark, and not only the n most important DLTWEs, as described in Section 5.4.

Comparing the sphere, with a degree of 9.8 to the rod-sphere, with degree 2, we see a performance difference of 38%. As expected, more time is spent on messaging and processing of inter-LP events for the sphere model. Additionally, the waiting time is increased. Comparing the rod-sphere, with an inter-LP diffusion ratio of 0.08, to the rod, with a ratio of 0.002, we see a performance difference of 71%. This gives some insight of the performance results in Section 5.5, where the large models in general had better performance than the small models, as they exhibit a smaller inter-LP diffusion ratio. Likewise it explains why the rod and the predator-prey benchmarks exhibited better performance than the other benchmarks, as they have a lower degree.

p

6. CONCLUSION

We have presented a new efficient approach to synchronization in optimistic PDES for spatial stochastic simulation of reaction-diffusion models, and used it to develop a parallel simulator, called Refined PNSM. A main contribution is to show that by refining the representation of the model state in order to expose explicit times for future diffusions between LPs, we can improve the accuracy of the optimism control, thereby significantly reducing the amount of rollbacks as well as maintaining the blocking at a modest level. Even though the refinement incurs a substantial overhead, experimental evaluation shows that the resulting parallel efficiency significantly outweighs the overhead.

We have also showed that our resulting simulator is superior in parallel performance to existing simulators for comparable models that have been reported in the literature, for cases where it has been possible to obtain data on simulated models.

We expect that the general strategy for the design of our parallelization also can be applied to simulators for other classes of models, where the partitions of different LPs exhibit tight and frequent interaction.

7. ACKNOWLEDGMENTS

We would like to thank the reviewers for their helpful comments.

This work was supported in part by the Swedish Research Council within the UPMARC Linnaeus centre of Excellence.

8. REFERENCES

[1] P. Bauer and S. Engblom. Sensitivity estimation and inverse problems in spatial stochastic models of chemical kinetics. volume 103 of *LNCSE*, pages 519–527. Springer, 2015.

[2] P. Bauer, J. Lindén, S. Engblom, and B. Jonsson. Efficient inter-process synchronization for parallel discrete event simulation on multicores. In *Proc. SIGSIM-PADS*, pages 183–194. ACM, 2015.

[3] C. Carothers, D. Bauer, and S. Pearce. ROSS: a high-performance, low memory, modular time warp system. In *Proc. 14th Workshop on Parallel and Distributed Simulation*, pages 53–60. IEEE, 2000.

[4] L. Chen, Y. Lu, Y. Yao, S. Peng, and L. Wu. A well-balanced time warp system on multi-core environments. In *Proc. 25th Workshop on Parallel and Distributed Simulation*, pages 1–9. IEEE, 2011.

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

[6] E. Deelman and B. K. Szymanski. Breadth-first rollback in spatially explicit simulations. In *Proc. 11th Workshop on Parallel and Distributed Simulation*, pages 124–131, 1997.

[7] L. Dematté and T. Mazza. On parallel stochastic simulation of diffusive systems. volume 5307 of *LNCS*, pages 191–210. Springer, 2008.

[8] B. Drawert, S. Engblom, and A. Hellander. URDME: a modular framework for stochastic simulation of reaction-transport processes in complex geometries. *BMC Syst. Biol.*, 6(1):76, 2012.

[9] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *Syst. biol.*, 1(2):230–236, 2004.

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

[11] A. Ferscha. Probabilistic adaptive direct optimism control in time warp. In *Proc. 9th Works. Par. Distr. Sim.*, pages 120–129. IEEE, 1995.

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

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

[14] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104(9):1876–1889, 2000.

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

[16] 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.

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

[18] 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 Perf. Eval. Rev.*, 35:22–31, 2008.

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

[20] M. J. Lawson, B. Drawert, M. Khammash, L. Petzold, and T.-M. Yi. Spatial Stochastic Dynamics Enable Robust Cell Polarization. *PLoS Comp. Biol.*, 9(7), July 2013.

[21] Z. Lin, C. Tropper, M. N. Ishlam Patoary, R. A. McDougal, W. W. Lytton, and M. L. Hines. NTW-MT: A Multi-threaded Simulator for Reaction Diffusion Simulations in NEURON. In *Proc. SIGSIM-PADS*, pages 157–167. ACM, 2015.

[22] J. Liu and D. Nicol. Lookahead revisited in wireless network simulations. In *Proc. 16th Workshop on Parallel and Distributed Simulation*, pages 79–88. IEEE, 2002.

[23] B. Lubachevsky, A. Schwartz, and A. Weiss. An analysis of rollback-based simulation. *ACM Trans. Model. Comput. Simul.*, 1(2):154–193, 1991.

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

[25] N. Marziale, F. Nobilia, A. Pellegrini, and F. Quaglia. Granular time warp objects. In *Proc. SIGSIM-PADS*, pages 57–68. ACM, 2016.

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

[27] A. Pellegrini, R. Vitali, S. Peluso, and F. Quaglia. Transparent and efficient shared-state management for optimistic simulations on multi-core machines. In *Proc. 20th Int. Symp. on Modeling, Analysis Simulation of Computer and Telecom. Systems*, pages 134–141. IEEE, 2012.

[28] P. L. Reiher, F. Wieland, and D. Jefferson. Limitation of optimism in the time warp operating system. In *Proc. 21st Winter Simulation Conference*, pages 765–770. ACM, 1989.

[29] L. M. Sokol, J. B. Weissman, and P. A. Mutchler. MTW: An Empirical Performance Study. In *Proc. 23rd Conference on Winter Simulation*, pages 557–563. IEEE Computer Society, 1991.

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

[31] J. S. Steinman. Breathing time warp. In *Proc. 7th Workshop on Parallel and Distributed Simulation*, pages 109–118. ACM, 1993.

[32] M. Sturrock, A. Hellander, A. Matzavinos, and M. A. J. Chaplain. Spatial stochastic modelling of the Hes1 gene regulatory network: intrinsic noise can explain heterogeneity in embryonic stem cell differentiation. *J. R. Soc. Interface*, 10(80), 2013.

[33] 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, 2011.

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