Exposing Inter-Process Information for Efficient PDES of Spatial Stochastic Systems on Multicores

JONATAN LINDÉN, Uppsala University, Sweden
PAVOL BAUER, German Center for Neurodegenerative Diseases, Germany
STEFAN ENGBLOM, Uppsala University, Sweden
BENGT JONSSON, Uppsala University, Sweden

We present a new approach for efficient process synchronization in parallel discrete event simulation on multicore computers. We aim specifically at simulation of spatially extended stochastic system models where time intervals between successive inter-process events are highly variable and without lower bounds: this includes models governed by the mesoscopic Reaction-Diffusion Master Equation (RDME). A central part of our approach is a mechanism for optimism control, in which each process disseminates accurate information about timestamps of its future outgoing interprocess events to its neighbours. This information gives each process a precise basis for deciding when to pause local processing in order to reduce the risk of expensive rollbacks caused by future "delayed" incoming events. We apply our approach to a natural parallelization of the Next Subvolume Method (NSM) for simulating systems obeying RDME. Since this natural parallelization does not expose accurate timestamps of future interprocess events, we restructure it to expose such information, resulting in a simulation algorithm called *Refined Parallel NSM* (Refined PNSM). We have implemented Refined PNSM in a parallel simulator for spatial extended Markovian processes. On 32 cores, it achieves an efficiency ranging between 43–95% for large models, and on average 37% for small models, compared to an efficient sequential simulation without any code for parallelization. It is shown that the gain of restructuring the naive parallelization into Refined PNSM more than outweighs its overhead. We also show that our resulting simulator is superior in performance to existing simulators on multicores for comparable models.

Additional Key Words and Phrases: Parallel Discrete-Event Simulation; PDES; Optimism control; Multicore; Spatial Stochastic Simulation

1 INTRODUCTION

Discrete Event Simulation (DES) is an important tool in a wide-ranging area of applications, e.g., in design and manufacturing, healthcare and epidemics, modeling of chemical and biological systems, 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 [16, 22, 30, 33]. In the last decade, a driving factor has been the advent of parallel computing platforms, such as GPUs and multicores [17]. Multicores allow fine-grained low-latency communication between processing elements, which is essential for efficient parallel simulation of many classes of models. This has been exploited in new synchronization techniques (e.g., [7, 35, 44]). Still, achieving good performance and speedup on multicores 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

Authors' addresses: Jonatan Lindén, Uppsala University, Dept. of Information Technology, Uppsala, Sweden, linden.jonatan@gmail.com; Pavol Bauer, German Center for Neurodegenerative Diseases, Neural Networks, Bonn, Germany, bauer.pa@gmail.com; Stefan Engblom, Uppsala University, Dept. of Information Technology, Uppsala, Sweden, stefane@it.uu.se; Bengt Jonsson, Uppsala University, Dept. of Information Technology, Uppsala, Sweden, bengt@it.uu.se.

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 future 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 allow an LP to process an event only when it is guaranteed that no straggler will later arrive [33], possibly causing significant performance loss by excessive synchronization and blocking of local LP execution; *optimistic* approaches allow stragglers by invoking suitable corrective action (rollback) [22, 30], 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 [8, 21]).

The advent of conservative and optimistic approaches can be regarded as resulting from a lack of accurate information. If each LP could know the timestamps and causal dependencies of future incoming events from other LPs, it could optimally advance its local simulation as far as is possible without causing stragglers. In the absence of precise information about future incoming events, each LP must continuously decide whether to continue local processing, risking future rollback costs, or to pause local processing, risking lost performance. Many techniques for controlling optimism can be seen as striking a trade-off between these risks [39]. However, they cannot change the basic fact that the achievable efficiency is fundamentally limited by the accuracy of available information on future inter-LP events.

In this paper, we show that a mechanism by which the LPs can obtain precise information about timestamps of future incoming inter-LP events is a crucial building block for the design of efficient PDES algorithms on multicores. We substantiate this claim 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. This information only allows it to communicate rather crude estimates of timestamps of future inter-LP events to its neighbours. We thereafter refine this natural parallelization algorithm, by restructuring each LP's simulation algorithm in a way that makes available more precise information about future inter-LP events. This information is disseminated using the technique of *Dynamic Local Time Window Estimates (DLTWE)*, introduced in our previous work [3],

Our parallelization is performed in the context of spatial simulation of models governed by the mesoscopic Reaction-Diffusion Master Equation (RDME) [18]. 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) [12]. 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 [14], regulatory processes relevant for differentiation of stem cells [41], or the polarization of yeast cells [25].

A natural parallelization of NSM, here called *Direct Parallel NSM* (Direct PNSM), has been proposed in, e.g., [10, 23]. 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 exposes precise information about timestamps of future outgoing inter-LP diffusion events, which can then be disseminated to neighboring LPs using the *Dynamic Local Time Window Estimates* (DLTWE) technique [3]. 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 [3]. 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 is more efficient in estimating the effects of perturbations in kinetic parameters thanks to a tighter coupling of sample trajectories (see [1] for an in-depth discussion and benchmarks). This tighter coupling comes at the cost of an increased event queue, thus causing sequential AEM to be significantly slower than sequential NSM. This disadvantage also applies to the parallelization of AEM which therefore 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 [3]. More details on the comparison with parallel AEM is found in Section 5.8.

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 both unstructured (tetrahedral) and structured (cartesian) meshes. On a 32-core machine, consisting of 4 sockets with 8 cores earch, Refined PNSM achieves an efficiency ranging between 43–95% for large models, and in average 37% 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 22–82% 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. It follows that our refinement of Direct PNSM increases the parallel efficiency to an extent that significantly outweighs its overhead.

We also compare our resulting simulator, based on Refined PNSM, to other existing parallel simulators on multicores 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. [43], we achieve a performance speedup on 32 cores which is approx. 44% 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 on multicores, which is superior in performance to other multicore simulators reported in the literature, and
- experimental evidence that, on multicores, 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. We remark that it is beyond our scope to consider simulators that run on clusters.

An implementation of the Refined PNSM solver is scheduled for release with the upcoming version 1.4 of the URDME simulation framework [11].

Organization of Paper. The next section reviews previous related research. 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.

This paper is a revised and extended version of [27].

2 RELATED WORK

Numerous methods for parallelization of discrete event simulation (PDES) have been proposed. Here we will only review works that are closest to our contributions; in general, we refer to the comprehensive surveys in [8, 16, 21].

Approaches to parallelization in PDES are coarsely classified into *conservative* [33] and *optimistic* [22, 30]. Optimistic approaches [22, 30] 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. The suitability of a technique depends on the employed computing platform; different techniques may perform well for distributed systems, e.g., communicating over MPI, as compared to shared memory multicores. Several techniques let LPs regulate their event processing rate in response to various statistics that are locally available to that LP, such as the frequency of rollbacks [36], or the expected timestamps of future incoming events as estimated from statistics of past incoming events [15]. Another idea is to employ moving constant size *time windows* that bound how far each LP can advance its local time (e.g., [28, 38, 40]). Such time windows are computed using model-specific *lookahead*, e.g., derived from known minimum communication delays. Our DLTWE technique also aims to provide a bound for how far LPs can safely advance local time. However, in our setting there is no statically available lookahead since inter-event times have no lower bound. Instead, time windows are computed and updated dynamically, enabling more accurate optimism control, even in situations where there are no minimum static time windows. It should be mentioned that while DLTWEs are rather accurate bounds for safe advancement of local LP times, they are not completely safe, since they may sometimes be decreased when new inter-process events are scheduled on short notice.

A further development of these approaches is the class of "near-perfect" state information (NPSI) protocols, including the *elastic time algorithm* [39]. Here, the time window is based on the global

virtual time (GVT) and on information about future messages to neighboring LPs, which are computed and communicated over a special high-speed network. Our optimism control can be seen as a refinement of ththe elastic time approach, where near-accurate information on future inter-LP messages is disseminated and used by neighboring LPs. Also, our realization of this technique exploits the low inter-process latency offered by modern multicores, removing the need for a dedicated high-speed network.

PDES on multicores has 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., [7, 35]). This adds cost for synchronizing data accesses across cores. Marziale et al. [31] tries to remedy the cost of inter-core accesses by grouping domains of a certain granularity to one single LP. Lin et al. [26] employ a technique called Multi-Level Queuing, in order to minimize the contention among threads in a multicore RDME simulator. Our implementation partitions the simulation domain in a way that tries to minimize the interaction between subdomain. Although this interaction is still significant, we suffer much less from contention between cores. There are several other optimizations for multicores that we do not consider, which could be inherited also into our framework, including NUMA-optimizations presented by Wang et al. [44] implemented for the general optimistic simulator ROSS [6], and optimizations for memory access latency by Pellegrini and Quaglia [34].

Parallel simulation of RDME models using the Next Subvolume Method was previously addressed by [10, 23, 26, 43]. The simulators are implemented in MPI, where each LP simulates a subvolume [26, 43] or a subdomain [10, 23]. We observe, as above, that the low-latency intercore communication offered by shared memory multicores gives us more efficient synchronization between processing elements. Dematté and Mazza [10] first proposed that optimistic PDES is favorable for solution of RDME models. Control of optimism was realized by a static time window [23] or Breathing Time Warp [43]. In Section 5.8, we compare our implementation to measurements reported by Wang et al. [43] and Lin et al. [26].

3 SPATIAL STOCHASTIC SIMULATION

The Reaction-Diffusion Master Equation (RDME) [18] 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 [19], numerous such sampling methods have been proposed. For spatial models, the most commonly used exact sampling method is the Next Subvolume Method [12] (NSM).

The NSM algorithm is a form of 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 depend 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 [5].

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.

3.1 Considerations for Faithfulness to Physical Model

In order to complete the description of the RDME with some context in which it arises, we here briefly review some aspects of the physical modeling involved in deriving a sufficiently accurate description in terms of a RDME from a not yet discretized physical model. The purpose is to gain an understanding into how the spatial discretization, in particular the choice of subvolume- and subdomain sizes, affects the accuracy of the RDME formulation. Thus, conditions for how the discretization can be performed come both from the physical model as well as from what is computationally tractable in a parallel implementation. General aspects of the RDME are discussed in [18, Ch. 8] and [42, Ch. XIV]; the present discussion draws upon [13, Ch. 2.2]. It is not necessary to digest the material of this section for appreciating the results in the rest of the paper.

The traditional well-stirred master equation covering the reactive kinetics in a single subvolume is derived from the physical premises that the molecules move around freely in vacuum and react when colliding in [20]. *Diffusion-controlled* kinetics [4], which is what the RDME tries to model, is different as diffusing by Brownian motion in a liquid means that the mean free path, i.e., the distance traveled between particle collisions, is at most on the order of the diameter of the solvent molecules, which is typically much smaller than that of the tracer particle itself. The mathematical description of Brownian motion is the *Itô diffusion*,

$$d\xi = \sigma dW(t), \tag{1}$$

where $\xi = [\xi_1; \xi_2; \xi_3]$ is the tracer particle coordinate, σ the diffusion constant, and where $W(t)$ is a three-dimensional Wiener process (which can be regarded as a continuous analogue of a random walk). The solution to (1) is a 3D Gaussian distribution and one can show that the *mean first exit time* from the ball $\|\xi\| < r$ is given by

$$\tau = \frac{r^2}{3\sigma^2}. \tag{2}$$

The fractional nature of the movement is evident since by (2), $r/\tau \sim \tau^{-1/2}$ and the limit $\tau \to 0$ (i.e. the molecule's "speed") makes no sense.

Under diffusion the molecule moves around irregularly in a space filling fashion, thus slowly searching through the entire volume. In this way a molecule traveling a distance on the order of its own radius ρ searches through a volume proportional to ρ^3. On the average, by (2) this takes time proportional to ρ^2/σ^2. A large number $N = \sigma^2/\rho^2\Delta t$ of such translocations therefore searches through a total volume of about $\sigma^2\rho\Delta t$. Although this is to be regarded as a crude estimate since the regions will partially overlap, when Δt is sufficiently small the linear scaling with Δt will still be valid.

This observation opens up for a rate model in terms of a continuous-time Markov chain because we can form a reaction probability which is proportional to Δt and retrieve the Markov chain in the limit $\Delta t \to 0$. However, this model is only valid in a *local* volume of radius less than about $\sigma \Delta t^{1/2}$ with Δt a characteristic time scale (e.g., average life time of the molecules). Beyond this volume the diffusion is too slow to consider the system as well-stirred and unresolved spatial effects may build up.

As the kinetics can no longer be regarded as well-stirred in the whole volume, a reasonable idea is to subdivide the domain V into smaller computational subvolumes V_j. This is done such that their individual volume $|V_j|$ is sufficiently small to make them behave as practically well-stirred by the presence of diffusion. It is clear from (2) that the *diffusion rate* per molecule in subvolume V_k is proportional to σ^2/h_k^2, where h_k is a measure of the length scale of subvolume V_k.

By construction, the RDME is only a valid model under certain requirements. Denote the minimum average survival time of the molecular species by τ_Δ. Then we should have that

$$\rho^2 \ll h^2 \ll \sigma^2 \tau_\Delta, \tag{3}$$

where, as before, the molecular radius is denoted by ρ and where h is a suitable measure of the length of each subvolume. The upper bound guarantees that the mixing in between reaction events by diffusion is sufficiently fast that each subvolume can be regarded as well-stirred. The lower bound on the subvolume size guarantees that molecules and reaction events can be properly localized within the subvolumes. Importantly, it follows that for a typical discretization satisfying (3), the total diffusion intensity will dominate that of the reactions.

Consider now a spherical region of space of radius H, discretized into smaller subdomains of radii about h. In total there are $M_V \sim (H/h)^3$ smaller such subvolumes and to set the scale we assume a reaction intensity W_r in each. By the above discussion there is also a diffusion intensity W_d in each subvolume, and W_d is fairly much greater than W_r in order for the modeling to be valid.

Since $W_d \propto 1/h^2$, we have that the total event intensity $\sim M_V h^{-2} = H^3 h^{-5}$. Further, there are $M_A \sim (H/h)^2$ subvolumes at the boundary, so the total diffusion intensity *out of* the spherical region is $\sim M_A h^{-2} = H^2 h^{-4}$. It follows that the ratio of private to external events scales as $\sim Hh^{-1}$, i.e., it increases with a larger private domain H and a finer discretization h. In other words, for a large enough finely discretized thread-private domain, there will be enough internal events to hide the cost associated with the specialized treatment at the boundary.

To conclude, the simulation work for a given model increases as h^{-5}, which is quite fast, more than can be compensated for by parallelization. Thus, the parameter h should not be chosen smaller than necessary for simulation accuracy by (3). Once h has been chosen, the observation that private to external events scales as $\sim Hh^{-1}$ implies the existence of a size H of subdomains that can reasonably be allocated to LPs when parallelizing.

4 PARALLELIZATION

In this section, we present our parallelization of the 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 [22]. 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* [3]. It is an adaptation of the breadth-first rollback mechanism [9], 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 cannot know the time of a specific inter-LP diffusion event in advance.

As the model dynamics are 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, explained in detail in Section 4.3. The estimates are communicated to neighbors and updated during each simulator loop execution. We refer to this best effort technique as *Time Window Estimates* (TWE) in the evaluation.

4.2 Refined PNSM

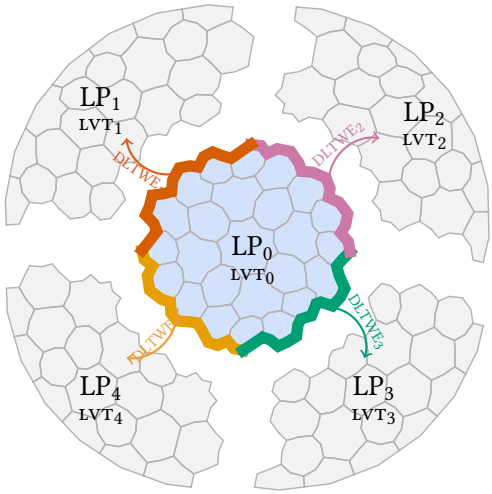


Fig. 1. Schematic drawing of the Refined PNSM algorithm. The domain is divided into several LPs (grey and blue areas), each operating at a local simulation time LVT_i. Each LP contains a number of subvolumes (solid lines), within which a set of reactions and diffusions take place. Looking at LP_0, all 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, orange and pink overlay) are stored in inter-LP queues, one per neighbor. The inter-LP event queues contain explicit next occurrence times of each diffusion, 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 $\text{LVT}_i < \text{DLTWE}_i$, otherwise block the execution in order to minimize the risk of rollbacks.

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 to the Direct PNSM. 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 near-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 *a)* one event queue containing the aggregated subvolume events, representing reactions and intra-LP diffusions, and *b)* 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. The top entry can change in two ways: either *a)* the corresponding diffusion is performed, bringing the next entry to the top, or *b)* the population in the corresponding voxel changes in such a way that the rate of the top entry changes, thereby changing its time of occurrence. This is essentially the Dynamic Local Time Window Estimate (DLTWE) technique, introduced in our previous work [3].

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 mechanism for communicating

timestamps of future inter-LP events in Direct PNSM, they accurately predict the times at which the inter-LP events will occur. However, rollbacks cannot be completely ruled out, since the estimates are updated whenever the 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. This corresponds to case *b)* in the previous paragraph. 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, we consider two forms of tuning.

The first form of 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. In fact, for n being equal or greater than the number of neighbors of any LP, Refined PNSM approximates a 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 caused the highest amount of rollbacks for the LP during the most recent time window of the the simulation. An exception is made for LPs with only a few neighbors, for which the DLTWEs of all neighbors are used.

The second form of tuning can be performed only in the Direct PNSM, and consists in scaling the expected next diffusion time 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.

```
1 structure LP_i:
2     m                          ▷ Number of neighbors
3     n                          ▷ Optimism control parameter
4     EventQueue[0 ... m]        ▷ Time-sorted queues of events
5     SubvolumeState[1 ... n_i]  ▷ State of subdomain
6     History                    ▷ History of past events
7     Channel[1 ... m]           ▷ Channels of incoming messages
8     Dltwe_{1...m}              ▷ Incoming DLTWEs, one per neighbor
```

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

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

```
1  while true do
       ▷ First phase: find the next event to process
       ▷ Peek at top of each message channel:
2      e_msg ← earliest message in { RetrieveMsg(Channel[j]) | 1 ≤ j ≤ m}
3      e_local ← earliest event in {peek(EventQueue[j]) | 0 ≤ j ≤ m}
4      if e_msg.time ≤ e_local.time then                ▷ If e_msg precedes any local event
5          e ← pop e_msg from its message channel       ▷ Next event e is from a neighbor
6      else
7          BadNbrs ← {indices of the n neighbors having caused the most rollbacks}
8          while ∃j ∈ BadNbrs s.t.Dltwe_j < e_local.time do  ▷ Next event's time > selected DLTWEs
9              if (∃m ∈ Channel[j]) ∨ (∃k, ∃e_m ∈ Channel[k]s.t.k ≠ j ∧ e_m.time < Dltwe_j) then
10                 restart main loop
11         e ← pop e_local from event queue             ▷ The next event to be processed is local
       ▷ Second phase: process selected event
12     if e.time < SubvolumeState[e.dest].time ∨ e is an anti-message then
13         SelectiveRollback(e) ▷ Rollback if e is a straggler (possibly local) or an anti-message
14         if e is an anti-message then restart main loop
15     add e to History
16     update SubvolumeState[e.subvol]
17     update timestamps of affected future reactions/diffusions in EventQueue[0 . . . m]
18     if e is a diffusion then
19         if e.dest is local then
20             update SubvolumeState[e.dest]
21         else
22             send diffusion message to LP_j where e.dest ∈ LP_j.SubvolumeState
23     for each neighbor LP_j do
24         LP_j.Dltwe_i ← time of peek (EventQueue[j])   ▷ Update the DLTWEs of neighbors
```

- *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,
- $Channel[1 \ldots m]$ is an array of incoming message channels, one for each neighbor, and
- $Dltwe_{1\ldots m}$ 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 [3], 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 [3].

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.

4.5 Faithfulness of Parallelization

An important step in the development of a parallel simulator for stochastic systems is to ensure that the algorithm is implemented correctly. A strong evidence of correctness is the ability to generate exactly the same solution as a sequential simulator, for a given model. Such an ability is not only a strong indicator of correctness, but also facilitates the finding of programming errors throughout the implementation. We identify two properties of our parallel simulator that allow for the generation of solutions that agree with those generated by the sequential simulator.

The first property is the use of the *same streams of random numbers* in the sequential as well as the parallel simulation. This can be achieved by assigning unique generator seeds to the pseudorandom number generators (PRNGs) used in the sequential as well as parallel simulator, for each subvolume in the model.

The second property is that the state of the PRNGs are reverted during rollbacks. Many PRNGs are reversible, e.g., linear congruential generators such as the Posix standard's `drand48()` and similar (our case), or more complex ones, such as the Mersenne twister [32]. For each entry in the event history, it is known how many pseudorandom numbers that were used, thus it is simply a matter of reverting the corresponding subvolume's or subvolumes' PRNG(s) the same number of steps. Hence, the state of a subvolume, including the random number generator, is transparent to rollbacks. Reverting the random number generator is also important for the reason that sampling bias is otherwise introduced into the parallel simulation. Such bias is the result of the systematic rejection of events that have been generated based on random numbers that are close to the maximum range of the generator (e.g., uniformly distributed random numbers close to 1). These are, for example, diffusion events that will occur further away from the local simulation time, and thus have a higher probability to be rolled back than events that occur shortly after the local simulation time.

Both these properties have been implemented in the Direct PNSM simulator, and have been used to verify that it computes the same trajectories as the corresponding sequential simulation

algorithm. A trajectory is in our case a sequence of model states at predefined instants of virtual time (i.e., the state at a subvolume after the processing of the latest event with a timestamp smaller than the predefined instant). For the Refined PNSM, there is no corresponding sequential implementation, since its discretization does not make sense in a sequential setting. However, Refined PNSM is implemented on the code base of Direct PNSM, indicating that part of the code has been validated in this way. As an alternative technique, correctness of parallel simulators of stochastic systems may also be verified by statistical tests between the sequential and parallel solutions, as shown by Wang et al. [43]. However, in this case it is very time-consuming, if not impossible, to assess whether a change in the parallel simulator has resulted in an error.

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 does load imbalance affect the performance?* (Section 5.7)
- *How well does the Refined PNSM compare to other works?* (Section 5.8)
- *Which model parameters affect the performance of the Refined PNSM algorithm?* (Section 5.9)

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 [11], that has been shown to be efficient in comparison to other NSM implementations [11, 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 algorithm, with optimism control turned off.
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, with optimism control turned off.
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):

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 computing the time window estimates 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.

(1) The *reversible isomerization* benchmark consists of spatial models defined on three different geometries; *sphere*, *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. Additionally, a configuration of the large sphere model with a lower population density has been evaluated, denoted *sparse*. The motivation for the latter benchmark is that the RDME is typically used when the simulated population is low, as described in Section 3. 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,

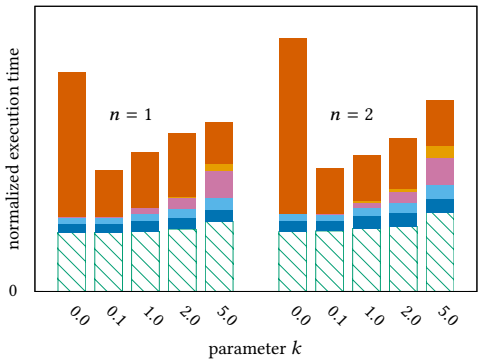
$$A \xrightarrow{c} B, \qquad B \xrightarrow{c} A,$$

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 [14, 41].

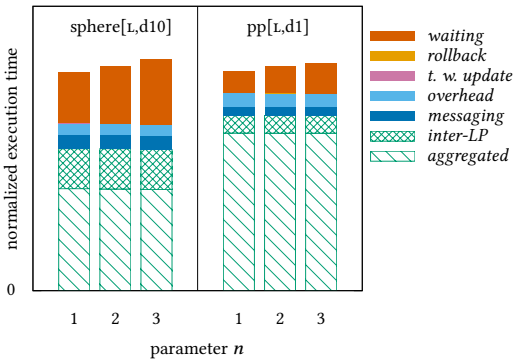
(2) The spatial *predator-prey* (denoted *pp* in the following) model was used as a benchmark in the study by Wang et al. [43]. The benchmark is the spatial extension of the Lotka-Volterra model, proposed by Schinazi [37]. The system contains three species, A, B, and C, where the initial copy number for each is set to 1000 per subvolume. The model reads

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

We have used the model parameters proposed by Wang et al. [43]. The diffusion rates of species B and C are $d_B^1 = 2.5$ and $d_C^1 = 5$, while $d_A = 0$ in all cases. 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 per LP. The model has been run at two sizes, viz., 40.000 (*small*) and 160.000 (*large*) subvolumes, and at a D:R ratio of 1:1.



(a) D-PNSM[TWE] on the sphere[L,d10] model.



(b) R-PNSM[DLTWE] on the sphere[L,d10] and the pp[L,d1] models.

Fig. 2. Instrumentation data for the tuning of the D-PNSM[TWE] and R-PNSM[DLTWE] algorithms for 32 cores. Time is normalized to the first bar of each group per benchmark.

(3) A model of the *Min-protein* system in the *E. coli* bacterium [14]. The system consists of five different species, whose interactions are described by five reactions. The model is challenging, both in that the species oscillate through the length of the bacteria, and that two of the species are limited to the membrane of the bacteria. The reaction- and diffusion-rates of the model can be found in [14], and the model is also available for download in the current release of URDME [2]. We evaluate three models corresponding to three different lengths of the bacterium, viz., 4 µm with ∼9.000 subvolumes, 10 µm with ∼24.000 subvolumes, and 15 µm with ∼35.000 subvolumes, roughly corresponding to wild-type *E. coli*, and two different lengths of a mutant *E. coli*, all discussed in [14].

For the reversible isomerization and the predator-prey benchmarks, we will denote specific model configurations as [s, dy], where the first parameter, the system size s, is either S (*small*) or L (*large*), and the second parameter is the D:R ratio, interpreted as y : 1. For the low population density sphere model, we have added the keyword "sparse" to the parameter list. For the mincde benchmark, the bacterium length s identifies the benchmark configuration, e.g., [s µm].

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 4.9.0, and the binaries were compiled using GCC 6.3.0. 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 [24]. 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

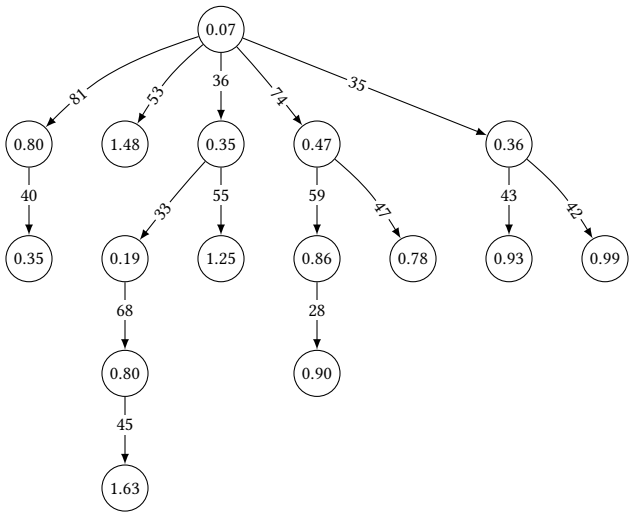


Fig. 3. Graph representing all LPs and the channels having caused the most rollbacks, for the D-PNSM algorithm during a simulation run of the sphere[L, d10] benchmark, with 16 threads. Each node represents an LP, and each edge represents a channel. Each LP is annotated with the total amount of time it has spent on processing rollbacks, in seconds. For each LP, the channel having caused the most rollbacks has been added as an edge from the sending LP to this LP, annotated with the percentage of the LP's rollbacks it has caused. Only for the slowest LP, such an edge has been omitted.

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 2(a) 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.

The instrumentation data of the R-PNSM[DLTWE] algorithm is shown in Figure 2(b), 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 that has caused the most rollbacks.

The observation that the best performance is obtained when each LP waits on the DLTWE of only one other LP can, we believe, be explained as follows. An important goal of optimism control is to make each LP progress at approximately the same speed, i.e., no LP should advance its simulation time significantly further than the slowest LP. Intuitively, the neighbors of the slowest LP have to wait the most for the slowest LP. For any LP, most of the time, an LP's speed will directly or indirectly be constrained by the slowest LP. Controlling the simulation speed of an LP more strictly, by increasing n, would only increase the overhead of the simulation, since it is still the slowest LP that limits the speed. Furthermore, increasing n can make the simulation more rigid than necessary by reducing the freedom that is necessary to accommodate modest temporary variations in simulation speed. It should be noted that with our selective rollback technique, a straggler need

not cause a rollback; hence our simulator benefits from allowing modest differences in simulation time between LPs. It should further be noted that it is quite difficult to predict statically, in advance, which neighbour will cause the most rollbacks.

To investigate how the waiting time, and the rollbacks, are distributed among the LPs, measurements were done for the D-PNSM algorithm, during a simulation run of the sphere[L,d10] benchmark, on 16 threads. For each LP, the total amount of time spent on processing rollbacks was recorded. For each channel, the number of rollbacks caused by stragglers received by the channel was recorded. The resulting graph is shown in Figure 3. The nodes in the graph represent the LPs, annotated with the total amount of time each LP spent on processing rollbacks. For each LP, one incoming edge, representing a channel, was added to the graph, viz., the channel having caused the most rollbacks to this LP (only for the slowest LP, such an edge was omitted). Each edge is annotated with the percentage of the number of rollbacks that were attributable to the corresponding channel. We see that each LP's rollbacks come from the same direction; in the end, the speed of every LP is limited by the slowest LP. The structure of rollbacks would also explain why increasing the parameter n does not improve performance: observing the DLTWE of the neighbor producing the second most rollbacks would, as can be seen in the graph, also be constrained, to the greatest part, by the slowest LP. Thus, the second DLTWE would only add overhead, and would not lead to a better control of the simulation speed.

5.5 Scalability Comparison

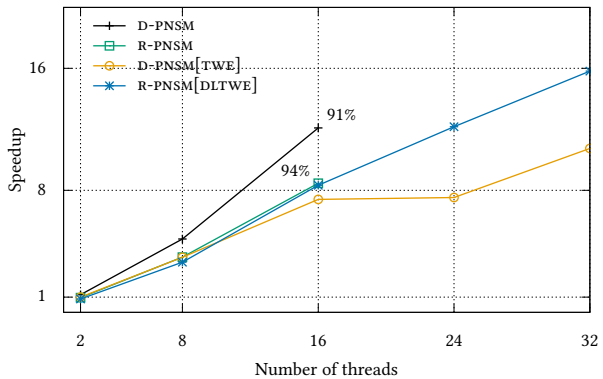


Fig. 4. Speedup over NSM for the sphere[L,d10] model. The D-PNSM and R-PNSM did not complete for more than 16 cores due to rollback explosions. For 16 cores, the success rate of the D-PNSM is approx. 91%, and for the R-PNSM it is approx. 94%.

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 [29]. 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 91%, and of the R-PNSM algorithm it is 94%. For 32 cores it is 33% for the D-PNSM algorithm, and 0% for the R-PNSM algorithm, 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 up to 40% 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 cost of the optimism control is marginal as can be seen by comparison with the performance of the R-PNSM algorithm. This is because the basis for 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 cost of the optimism control is 63% for 16 cores, as seen by comparison with the performance of the D-PNSM algorithm.

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 9% faster. However, for 16 cores and more, the opposite holds, and the R-PNSM[DLTWE] is up to 42% 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. The following performance comparisons are only based on benchmarks that both algorithms handle. For these, for 32 cores, the R-PNSM[DLTWE] achieves an efficiency ranging between 43–95% on the large models, compared to NSM. Compared to the D-PNSM[TWE], we see a performance improvement in the range of 22–82%. For the smallest model, under several different diffusion ratios, the performance of the R-PNSM[DLTWE] is up to about 4x better than the D-PNSM[TWE] algorithm. The average speedup of R-PNSM[DLTWE] over NSM for all benchmarks is 17.

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 observe that for both algorithms, the two-dimensional structured mesh model of the *predator-prey* model is less challenging than the other models, since the number of neighbors per LP is small. We also observe that the smaller models are more challenging, due to the limited amount of parallelism available. Especially for the small sphere models, the performance difference of the D-PNSM[TWE] and the R-PNSM[DLTWE] is particularly accentuated. It also seems that both the R-PNSM[DLTWE] and the D-PNSM[TWE] algorithm performs better in highly diffusive models, which was hinted at in Section 3.1.

We note that the D-PNSM[TWE] algorithm did not handle models with smaller population density, viz., the mincde and the sparse sphere models. We believe this to be due to the estimates becoming increasingly unreliable as the copy number of the boundary subvolumes decrease, causing a dead-lock.

5.6 Overhead of the Refined PNSM

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%.

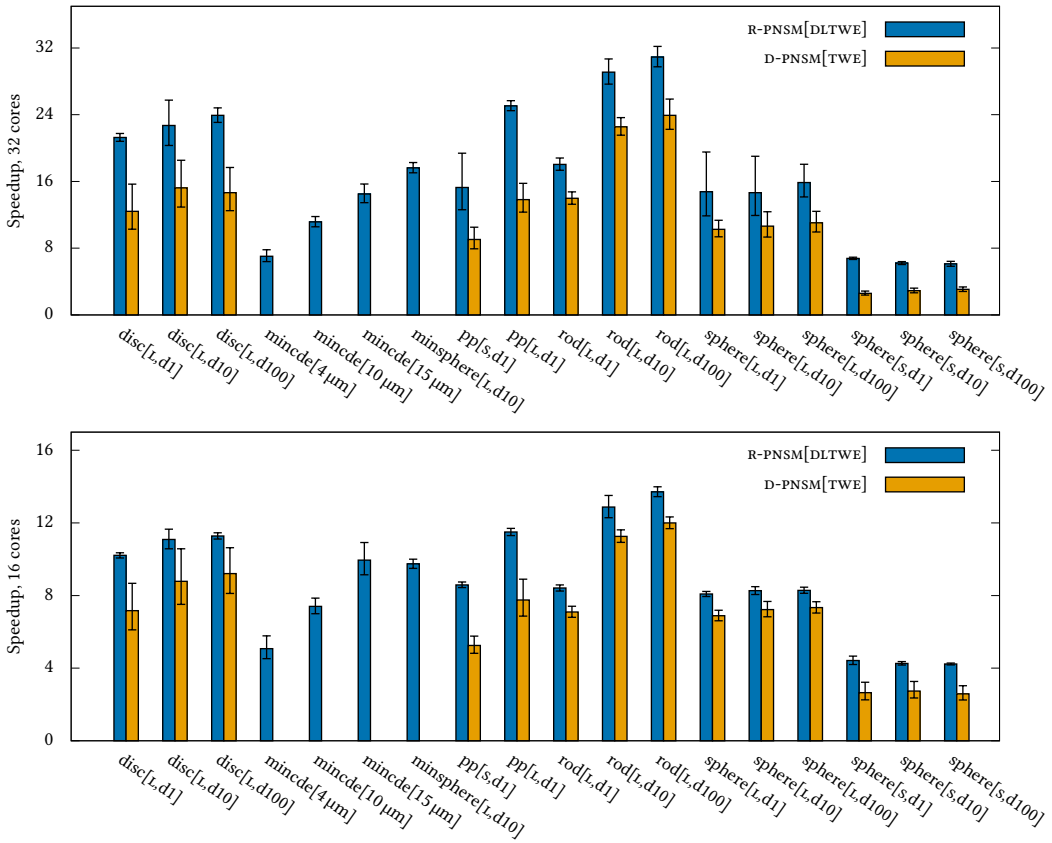


Fig. 5. Speedup over NSM, 32 cores (top) and 16 cores (bottom), for all benchmarks. Error bars denote 95% confidence interval over 10 runs.

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 16 cores and more. An explanation is provided by Figure 6: For the R-PNSM[DLTWE] the relative amount of effort not spent on useful work increases by 41% 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.

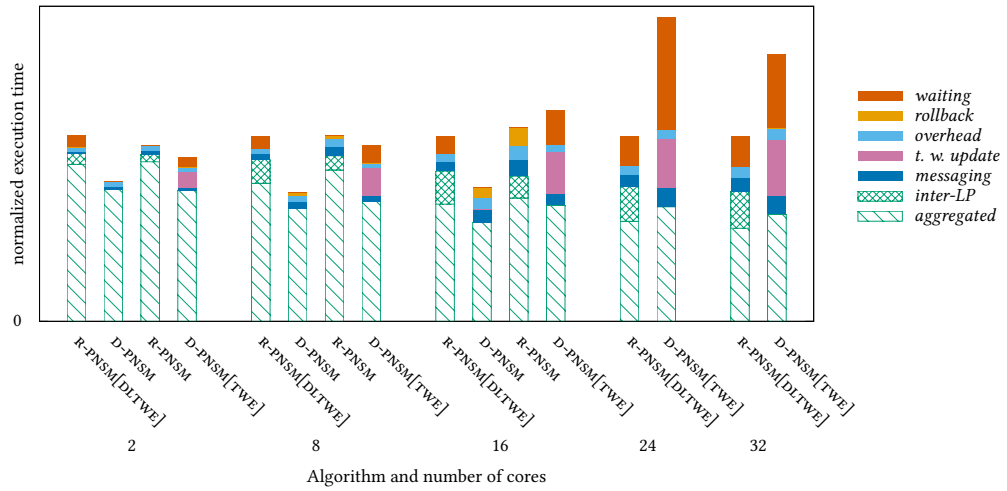


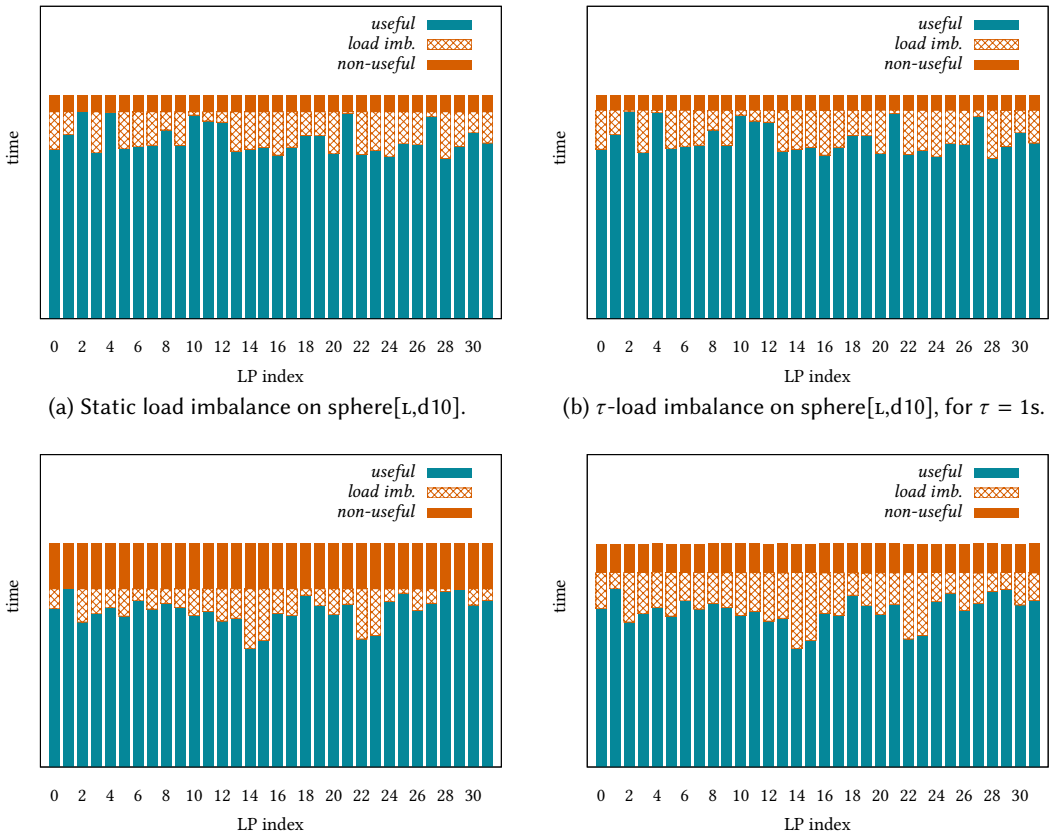
Fig. 6. Instrumentation data for all algorithms on the sphere[L,d10] model. Time is normalized to the first bar of each group of cores. We recall from Figure 4, that the D-PNSM and R-PNSM did not complete for more than 16 cores due to rollback explosions. For 16 cores, the success rate of the D-PNSM is approx. 91%, and for the R-PNSM it is approx. 94%.

5.7 Load Imbalance

In this section, we try to characterize the waiting time that cannot be prevented by optimism control. Since our implementation does not perform dynamic load balancing, it is important that the simulation model, i.e., its load, is evenly partitioned onto the LPs (which are mapped one-to-one on processor cores). Otherwise, LPs that are given less simulation work than others, will spend some of their time on waiting and processing rollbacks, hereafter denoted *non-useful* work. However, many models exhibit a load that changes dynamically, in which case the load imbalance is unavoidable, no matter whether optimism control is used or not.

To capture both the static and the dynamic property of load imbalance, we define the τ-*load imbalance* of a simulation run as the amount of time per LP spent on non-useful work exceeding that of the slowest LP, during each interval of length τ for some time step τ. In particular, as $\tau \to 0$, all time spent on waiting and on processing rollbacks will be characterized as τ-load imbalance, since for small enough intervals, there will always be some LP that does not have any time spent on non-useful work. Given a simulation of length t, for $\tau = t$, we denote τ-load imbalance as *static load imbalance*. Thus, for a simulation method that does not perform any dynamic load balancing, one necessary but not sufficient condition for good performance is that the static load imbalance is close to zero. For a simulation where load balancing is done at a step size of T, a similar condition would be that the τ-load imbalance is close to zero for $\tau = T$. When $\tau \to 0$, given a perfect dynamic load-balancing procedure, all non-useful work can be removed.

In the case of static load imbalance, it represents an upper bound on the amount of non-useful work that can be prevented by a suitable optimism control. In the case of τ-load imbalance, it represents an upper bound on the amount of non-useful work that can be prevented with a load-balancing mechanism with a minimum step size of τ. In Figure 7(a) we see a simulation run of the sphere[L,d10] model on 32 threads. Each bar represents the time allocation of an individual LP, divided into useful work, non-useful work, and the proportion of the non-useful work that is classified as static load imbalance. In the figure, LP 2 has the least amount of non-useful work, i.e., it



(a) Static load imbalance on sphere[L,d10].

(b) τ-load imbalance on sphere[L,d10], for $\tau = 1$s.

(c) Static load imbalance on mincde[10 µm].

(d) τ-load imbalance on mincde[10 µm], for $\tau = 1$s.

Fig. 7. Instrumentation data for each thread of the R-PNSM[DLTWE] algorithm on the sphere[L,d10] benchmark in Figures (a) and (b), and mincde[10 µm] in Figures (c) and (d). 32 threads.

is the slowest LP. Thus, for each LP, all non-useful work exceeding that of LP 2, is classified as static load imbalance, and could possibly have been removed by improving the initial partitioning of the between LPs. The rest of the non-useful work is then comprised of dynamic load imbalance and imperfections in the optimism control protocol, and represent an upper bound on the improvement that can be achieved with a better optimism control. For the sphere[L,d10] model, we could have achieved at most 7% better performance by improving the optimism control. By improving the initial partition, we could have achieved at most 12% better performance. In Figure 7(b), we see the same data, except that we show the part of the waiting time that can be classified as τ-load imbalance, for a time step of $\tau = 1$s. We see that the difference between static load imbalance and τ-load imbalance is only marginally different, i.e., the load is very uniform over time, which we already know by the definition of the model, in Section 5.2. On the other hand, in Figures 7(c) and 7(d), we have applied the same load imbalance measures, but on the mincde[10 µm] benchmark, which exhibits considerable load variations. Here we see that the difference between the static load imbalance and the dynamic load imbalance measure is bigger, the amount of non-useful

work classified as load imbalance increases by 70%. Thus, for the kind of load that mincde[10 μm] represents, to improve performance some kind of load balancing would be a suitable method.

Finally, in Figure 8(a), the static load imbalance measure is displayed on the instrumentation data for the sphere[L,d10] model. Now we see that there is very small room for performance improvement when it comes to reducing the waiting time by modifying the optimism control.

5.8 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 [43] and [26].

Wang et al. [43] simulate a predator-prey model on a two-dimensional 200×200 structured grid, using their parallel Abstract NSM algorithm, corresponding to our pp[s,d1] model. 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 15.8. Lin et al. [26] 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 [3], 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 [43]. The speedup for PAEM over the NSM, however, is only 5.2 on the same benchmark, largely due to its poor memory efficiency.

5.9 Performance Indicators: Inter-LP Diffusion Ratio and Degree

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

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

Degree The graph degree of the communication network between LPs.

In Figure 8(b) 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 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 into 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

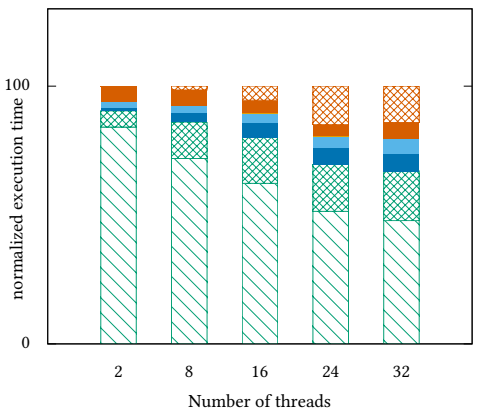
diffusion ratio. Likewise it explains why the rod and the predator-prey benchmarks exhibited better performance than the other benchmarks, as they have a lower degree.

6 CONCLUSION

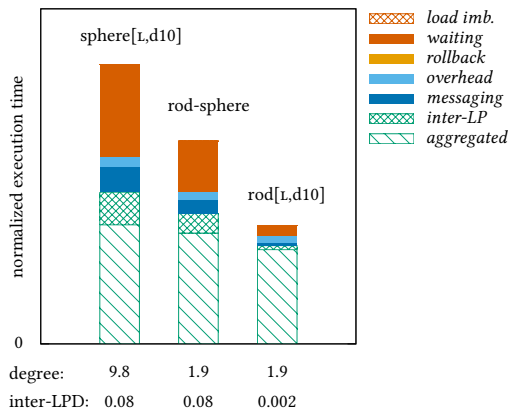
We have presented a new efficient approach to synchronization in optimistic PDES for spatial stochastic simulation of reaction-diffusion models on multicores, and used it to develop a parallel simulator, called Refined PNSM. The Refined PNSM simulator builds on a natural parallelization of the Next Subvolume Method, which is restructured in order to expose near-accurate timestamps of future interprocess events. By disseminating these timestampes using the DLTWE technique, 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 restructuring incurs a substantial overhead, experimental evaluation shows that the resulting parallel efficiency significantly outweighs the overhead.

We also show that an optimization, in which each LP only uses a single (in some cases two) incoming time estimates to control local processing, improves the optimism control by providing more flexibility to adapt to temporary fluctuations, without signifantly increasing the number of rollbacks. 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 relevant data.

More generally, our work shows how the fine-grained and low-latency communication offered by multicores can be exploited to perform efficient parallelization of simulation, even for models that are "difficult" in the sense that inter-process ineraction is fine-grained, unpredictable, and without lower bounds.



(a) Run on the sphere[L,d10] benchmark, 2 to 32 threads. The portion of the waiting time that can be attributed to static load imbalance according to our model is shown in the figure. Time is normalized per bar.



(b) Run using all DLTWEs on the sphere[L,d10], the rod-sphere, and the rod[L,d10] models, 32 cores. Time is normalized to the leftmost bar.

Fig. 8. Instrumentation data of the R-PNSM[DLTWE] algorithm.

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.

REFERENCES

[1] P. Bauer and S. Engblom. 2015. Sensitivity Estimation and Inverse Problems in Spatial Stochastic Models of Chemical Kinetics. LNCSE, Vol. 103. Springer, 519–527.
[2] P. Bauer and S. Engblom. 2017. *The URDME Manual version 1.3.* Technical Report 2017-003. Dept of Information Technology, Uppsala University. Available at http://arxiv.org/abs/0902.2912.
[3] P. Bauer, J. Lindén, S. Engblom, and B. Jonsson. 2015. Efficient Inter-Process Synchronization for Parallel Discrete Event Simulation on Multicores. In *Proc. SIGSIM-PADS.* ACM, 183–194.
[4] O. G. Berg and P. H. von Hippel. 1985. Diffusion-Controlled Macromolecular Interactions. *Ann. Rev. Biophys. Chem.* 14, 1 (1985), 131–160.
[5] M. A. Gibson J. Bruck. 2000. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *J. Phys. Chem. A* 104, 9 (2000), 1876–1889.
[6] C. D. Carothers, D. Bauer, and S. Pearce. 2000. ROSS: a high-performance, low memory, modular time warp system. In *Proc. 14th Workshop on Parallel and Distributed Simulation.* IEEE, 53–60.
[7] L.-L. Chen, Y.-S. Lu, Y. P. Yao, S. L. Peng, and L.-D. Wu. 2011. A Well-Balanced Time Warp System on Multi-Core Environments. In *Proc. 25th Workshop on Parallel and Distributed Simulation.* IEEE, 1–9.
[8] S. R. Das. 2000. Adaptive Protocols for Parallel Discrete Event Simulation. *J. Oper. Res. Soc.* 51, 4 (2000), 385–394.
[9] E. Deelman and B. K. Szymanski. 1997. Breadth-First Rollback in Spatially Explicit Simulations. In *Proc. 11th Workshop on Parallel and Distributed Simulation.* IEEE, 124–131.
[10] L. Dematté and T. Mazza. 2008. On Parallel Stochastic Simulation of Diffusive Systems. In *Proc. 6th Int'l Conference on Computational Methods in Systems Biology (CMSB '08).* Springer-Verlag, Berlin, Heidelberg, 191–210.
[11] B. Drawert, S. Engblom, and A. Hellander. 2012. URDME: a modular framework for stochastic simulation of reaction-transport processes in complex geometries. *BMC Syst. Biol.* 6, 76 (2012), 1–17.
[12] J. Elf and M. Ehrenberg. 2004. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *Syst. biol.* 1, 2 (2004), 230–236.
[13] S. Engblom. 2008. *Numerical Solution Methods in Stochastic Chemical Kinetics.* Ph.D. Dissertation. Uppsala University.
[14] D. Fange and J. Elf. 2006. Noise-Induced Min Phenotypes in E. coli. *PLoS Comput. Biol.* 2, 6 (2006), e80.
[15] A. Ferscha. 1995. Probabilistic Adaptive Direct Optimism Control in Time Warp. In *Proc. 9th Workshop on Parallel and Distributed Simulation.* IEEE, 120–129.
[16] R. M. Fujimoto. 1990. Parallel Discrete Event Simulation. *Comm. of the ACM* 33, 10 (1990), 30–53.
[17] R. M. Fujimoto. 2016. Research Challenges in Parallel and Distributed Simulation. *ACM Trans. Model. Comput. Simul.* 26, 4 (2016), 22:1–22:29.
[18] C. W. Gardiner. 2007. *Handbook of stochastic methods for physics, chemistry, and the natural sciences.* Springer.
[19] D. T. Gillespie. 1977. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* 81, 25 (1977), 2340–2361.
[20] D. T. Gillespie. 1992. A rigorous derivation of the chemical master equation. *Phys. A* 188 (1992), 404–425.
[21] S. Jafer, Q. Liu, and G. A. Wainer. 2013. Synchronization methods in parallel and distributed discrete-event simulation. *Simul. Model. Pract. Th.* 30 (2013), 54–73.
[22] D. R. Jefferson. 1985. Virtual Time. *ACM Trans. Program. Lang. Syst.* 7, 3 (1985), 404–425.
[23] M. Jeschke, R. Ewald, A. Park, R. Fujimoto, and A. M. Uhrmacher. 2008. A Parallel and Distributed Discrete Event Approach for Spatial Cell-biological Simulations. *SIGMETRICS Perf. Eval. Rev.* 35 (2008), 22–31.
[24] G. Karypis and V. Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392.
[25] M. J. Lawson, B. Drawert, M. Khammash, L. Petzold, and T. M. Yi. 2013. Spatial Stochastic Dynamics Enable Robust Cell Polarization. *PLoS Comp. Biol.* 9, 7 (2013), e1003139.
[26] Z. Lin, C. Tropper, R. A. McDougal, M. N. I. Patoary, W. W. Lytton, Y. Yao, and M. L. Hines. 2017. Multithreaded Stochastic PDES for Reactions and Diffusions in Neurons. *ACM Trans. Model. Comput. Simul.* 27, 2 (2017), 7:1–7:27.
[27] J. Lindén, P. Bauer, S. Engblom, and B. Jonsson. 2017. Exposing Inter-Process Information for Efficient Parallel Discrete Event Simulation of Spatial Stochastic Systems. In *Proc. SIGSIM-PADS.* ACM, 53–64.
[28] J. Liu and D. Nicol. 2002. Lookahead Revisited in Wireless Network Simulations. In *Proc. 16th Workshop on Parallel and Distributed Simulation.* IEEE, 79–88.
[29] B. Lubachevsky, A. Schwartz, and A. Weiss. 1991. An Analysis of Rollback-based Simulation. *ACM Trans. Model. Comput. Simul.* 1, 2 (1991), 154–193.

[30] B. D. Lubachevsky. 1988. Efficient parallel simulations of dynamic Ising spin systems. *J. Comput. Phys.* 75, 1 (1988), 103–122.
[31] N. Marziale, F. Nobilia, A. Pellegrini, and F. Quaglia. 2016. Granular Time Warp Objects. In *Proc. SIGSIM-PADS*. ACM, 57–68.
[32] M. Matsumoto and T. Nishimura. 1998. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator. *ACM Trans. Model. Comput. Simul.* 8, 1 (Jan. 1998), 3–30.
[33] J. Misra. 1986. Distributed Discrete-Event Simulation. *ACM Comput. Surv.* 18, 1 (1986), 39–65.
[34] A. Pellegrini and F. Quaglia. 2015. NUMA Time Warp. In *Proc. SIGSIM-PADS*. ACM, 59–70.
[35] A. Pellegrini, R. Vitali, S. Peluso, and F. Quaglia. 2012. Transparent and Efficient Shared-State Management for Optimistic Simulations on Multi-core Machines. In *Proc. MASCOTS*. IEEE, 134–141.
[36] P. L. Reiher, F. Wieland, and D. Jefferson. 1989. Limitation of Optimism in the Time Warp Operating System. In *Proc. 21st Winter Simulation Conference*. ACM, 765–770.
[37] R. B. Schinazi. 1997. Predator-prey and host-parasite spatial stochastic models. *Ann. Appl. Probab.* 7, 1 (1997), 1–9.
[38] L. M. Sokol, J. B. Weissman, and P. A. Mutchler. 1991. MTW: An Empirical Performance Study. In *Proc. 23rd Winter Simulation Conference*. IEEE, 557–563.
[39] S. Srinivasan and P. F. Reynolds Jr. 1998. Elastic Time. *ACM Trans. Model. Comput. Simul.* 8, 2 (1998), 103–139.
[40] J. S. Steinman. 1993. Breathing Time Warp. In *Proc. 7th Workshop on Parallel and Distributed Simulation*. ACM, 109–118.
[41] M. Sturrock, A. Hellander, A. Matzavinos, and M. A. J. Chaplain. 2013. 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), 20120988.
[42] N. G. van Kampen. 2004. *Stochastic Processes in Physics and Chemistry* (2nd ed.). Elsevier.
[43] B. Wang, B. Hou, F. Xing, and Y. Yao. 2011. Abstract Next Subvolume Method: A logical process-based approach for spatial stochastic simulation of chemical reactions. *Comput. Biol. Chem.* 35, 3 (2011), 193–198.
[44] J. Wang, D. Jagtap, N. B. Abu-Ghazaleh, and D. Ponomarev. 2014. Parallel Discrete Event Simulation for Multi-Core Systems: Analysis and Optimization. *IEEE Trans. Par. Distr. Syst.* 25, 6 (2014), 1574–1584.