On Parallel Stochastic Simulation of Diffusive Systems

Lorenzo Dematté[1,2] and Tommaso Mazza[1]

[1] The Microsoft Research - University of Trento
Centre for Computational and Systems Biology
Piazza Manci, 17, 38100, Povo (TN), Italy
{dematte,mazza}@cosbi.eu
[2] Department of Information Engineering and Computer Science (DISI),
University of Trento

Abstract. The parallel simulation of biochemical reactions is a very interesting problem: biochemical systems are inherently parallel, yet the majority of the algorithms to simulate them, including the well-known and widespread Gillespie SSA, are strictly sequential. Here we investigate, in a general way, how to characterize the simulation of biochemical systems in terms of Discrete Event Simulation. We dissect their inherent parallelism in order both to exploit the work done in this area and to speed-up their simulation. We study the peculiar characteristics of discrete biological simulations in order to select the parallelization technique which provides the greater benefits, as well as to touch its limits. We then focus on reaction-diffusion systems: we design and implement an efficient parallel algorithm for simulating such systems that include both reactions between entities and movements throughout the space.

Keywords: Parallel and distributed simulation, reaction-diffusion systems, Gillespie SSA.

1 Introduction

In computational biology, the interest on multi-processor computing is growing over the years, even if ubiquitous and parallel computing require deep knowledge both on the bio-reality and on the tools in charge of handling and interpreting it. Indeed, the correct parallel computation of whatever problem must take into account four milestones: (i) the best computational splitting policy; (ii) how to handle synchronization among the computational workers, (iii) the more suitable hardware architecture and software packages to use and (iv) the nature of the inherent parallelism.

There are problems naturally parallelizable and others purely serial. According to the case, the additional computing power afforded by new machines can be used to advantage of one or of the other. To enhance the efficiency of Monte Carlo simulations, Single Replication in Parallel (SRIP) and Multiple Replications in Parallel (MRIP) computational paradigms have been widely contemplated in the past and deemed to be appropriate.

M. Heiner and A.M. Uhrmacher (Eds.): CMSB 2008, LNBI 5307, pp. 191–210, 2008.

Single Replication in Parallel. The SRIP approach is based on the decomposition of a stochastic trajectory into logical processes, running on different processors and communicating by means of message passing protocols [1]. For naturally divisible problems, it shows elevated performances in speed-up and scale-up benchmarks. Significant drawbacks originate from the necessity for warranty of synchronism.

Multiple Replications in Parallel. The MRIP method speeds up simulation by launching independent replications on multiple computers and using different random seeds in such a way the processes result approximatively uncorrelated [2]. In contrast to SRIP, MRIP can be easily applicable to any system, independent of the inherent system parallelism. However, the fact that a single replication cannot be executed on a unique processor and that outputs (or pieces of them) almost deterministic are identical when replicated, make the use of MRIP approaches sometimes inappropriate [3]. The MRIP and SRIP approaches are not exclusive, i.e., it is possible to use MRIP and SRIP in the same simulation program.

In biology, whereas the MRIP policy, well understood and investigated for a long time [2], [4], [5], [3], [6], [7], [8], [9], [10], [11], finds straightforward application to real case-studies [12], [13], the SRIP policy has a rather vague characterization. SRIP methods can be further divided into two opposite subcategories which include: (a) methods that exploit *data-parallelism* (or *loop-level parallelism*), namely that exemplify simulation of interacting particles on a finite grid in which individual processors are in charge of simulating the state of each site [14]; (b) methods that exploit *task-parallelism* (or *functional parallelism*), namely that divide the computation of a realization into a set of sub-computations among cooperative processors by computational dependency criteria [1], [15]. To date, the research in distributed-parallel processing has successfully solved many related problems; however, it has not led yet to a portable and efficient tool for distributing stochastic simulation in the field of computational biology. We aim to move the attention of the reader toward our target by going through the theoretical bases and strategic decisions which configure our insight.

In particular, the next section will introduce the Gillespie Stochastic Simulation Algorithm (SSA), the most known and the *de-facto* standard for the simulation of biochemical systems at microscopic and mesoscopic level and a possible extension for simulating bigger systems where spatiality and diffusion are important variables. Next, we will briefly introduce the category of computer-simulation systems known as DES and the work done on these systems in the light of parallel and distributed computing. We will show how the SSA can be reformulated in term of a DES system, and we will show the characteristics the algorithm assumes when it runs in a parallel environment. Section 4 will present how these concepts were used in the designing and implementation of a reaction-diffusion simulator that runs on HPC clusters, and Sections 5 and 6 will close the paper with an example and considerations about future work.

2 The Gillespie SSA

The stochastic approach to chemical kinetics was first employed by Delbruck in the '40s. The basic assumptions of this approach are that a chemical reaction occurs when two (or more) molecules of the right type collide in an appropriate way, and that such collisions are *random* in a system of molecules in thermal equilibrium. Whenever two molecules come into a certain proximity, they can react with some probability: collisions are frequent, but those with the proper orientation and energy, that is the collisions that allow molecules to react together, are infrequent. In [16] and [17], Gillespie introduced the additional assumption that the system is in thermal equilibrium. This assumption means that the considered system is a well-stirred mixture of molecules, where the number of non-reactive collisions is much higher than the number of chemical reactions. It makes possible to state that the molecules are randomly and uniformly distributed at all times. The derived stochastic method becomes computationally lighter than the classical methods in charge of predicting collisions by estimating the collision volume of each particle.

The so called *Stochastic Simulation Algorithm* (SSA) models a general biological system as a set of pairs *(entity type, quantity)* and a set of possible interactions between the entities. In the case of biochemical models, *entities* are molecules and *interactions* are coupled chemical reactions. Therefore, we can reduce the necessary parameters for describing a system to:

- the *entities*, usually referred to as *species*, present in the system $S_1, ..., S_N$;
- the number and type of *interactions*, called *reaction channels*, through which the molecules interact $R_1, ..., R_M$;
- the state vector $\mathbf{X}(t)$ of the system at time t, where $X_i(t)$ is the number of molecules of species S_i present at time t.

The state vector $\mathbf{X}(t)$ is a vector of random variables, that does not take account of the position and velocity of the single molecules. For each reaction channel R_j, a function a_j, called *propensity function* for R_j, is defined as:

$$a_\mu = h_\mu c_\mu, \text{ for } \mu = 1, \ldots, M \tag{1}$$

such that h_μ is the number of distinct reactant combinations for reaction R_μ and c_μ is a constant depending on physical properties of the reactants. The c_μ constant is usually called *base rate*, or simply *rate* of an action, while the value of the function a_μ is called the *actual rate*.

Gillespie derived a physical correct *Chemical Master Equation* (CME) from the above representation of biochemical interactions. Intuitively, this equation shows the stochastic evolution of the system over time, which is indeed a Markov process. Gillespie also presented in [17] an exact procedure, called *exact stochastic simulation*, to numerically simulate the stochastic time evolution of a biochemical system, thus generating one single trajectory. The procedure is based on the *reaction probability density function* $P(\tau, \mu)$, which specifies the probability that the next reaction is an R_μ reaction and that it occurs at time τ:

$$P(\tau, \mu) = \begin{cases} a_\mu \exp(-a_0 \tau) & \text{if } 0 \leq \tau < \infty \text{ and } \mu = 1, \ldots, M \\ 0 & \text{otherwise} \end{cases}$$

where a_μ is the *propensity function* and a_0 is the sum of a_μ, $\mu = 1, \ldots, M$.

The *reaction probability density function* is used in a stochastic framework to compute the probability of an action to occur. The way of computing the combinations h_μ and, consequently the *actual rate* a_μ, varies with the different kind of reactions. In the case of first-order reactions, h_μ is equal to the number of entities (the *cardinality*) of the one reactant, while in the case of second-order reactions, h_μ corresponds to the number of all possible interactions that can take place among the reactants.

2.1 Simulation of Reactive-Diffusive Systems

When studying a single localized pathway, the macroscopic description of its kinetics usually suffices. On the other end, many biological processes are not local and, often, they take place in an inhomogeneous medium, the *cytosol*, where spatially localized fluctuations of inorganic catalysts and intracellular diffusion can play an important role. When dealing with such processes, it is mandatory to explicitly consider the cell geometry and, in general, spatial conformations and diffusion processes.

Several algorithms for the simulation of reactive-diffusive systems exist; each of them uses a different abstraction that gives a different level of detail which influences both the accuracy of the simulation and its execution speed. Chemical and biochemical reactions can be simulated in a very precise and detailed way using *molecular dynamics* [18], a form of computer simulation where atoms are allowed to interact under known physics laws. In these simulations, details about the chemical reactions, like formation and bonds breaking between single atoms, are explicitly simulated as well as the position and energy of every atom in the system. These methods have been applied to a wide range of problems of chemical and biological interest, such as chemical reactions in solution and enzymes and solvent effects on electronic excited states.

Other methods, like the one used by Bray et al. in *Smoldyn* [19], operate at a coarser level of detail, where molecules have an identity and an exact position in a continuous space, but no volume, shape or inertia. Moreover, every molecule of interest is represented as an individual point, while those that are not of interest (water, non-reactive molecules, etc.) are not represented. Molecules move at rates specified by a *diffusion coefficient* and diffuse in random directions and distances calculated by means of the *Fick's* second law. Bimolecular reactions take account of the spatial relations; a bimolecular reaction occurs if two reactants approach each other within a *binding radius*, a radius that is different (typically smaller) than the physical radius of the molecules, and that depends on the diffusion coefficients and on the reaction rate constant. Simulated space is continuous; on the other hand, simulated time is discrete as reactions, computation of movements and update of the position are done at fixed time steps.

2.2 Reaction-Diffusion with the Gillespie Method

The Gillespie algorithm, introduced in Sec. 2, allows to simulate chemical reactions in an efficient way. Every collision that leads to a reaction is explicitly simulated, but collisions that do not lead to a reaction are not. The stochastic behaviour of the chemical system is preserved, as molecules are still represented as discrete quantities, but information on a single molecule, and with it any positional information, are lost. Moreover, the assumptions made by Gillespie explicitly rule out diffusion from the system: since the solution is in thermal equilibrium, it is assumed that diffusion is instantaneous so that each molecule has the same probability of reacting with every other molecule in the system. The algorithm works well locally, but cannot be used to represent complex pathways that span over a considerable extension of reactions taking place in an inhomogeneous medium.

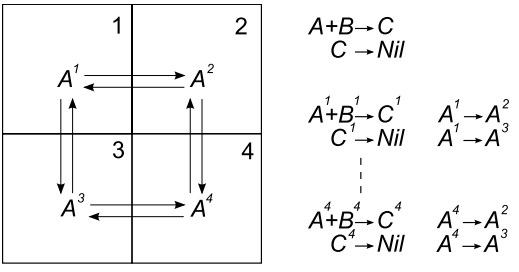


Fig. 1. The extension to the Gillespie SSA proposed by Bernstein. On the left, a discretization of the space into four cells. On the right, the species and the reactions added to the system in order to deal with diffusion.

A proposed extension is the *discretization* of the space by subdivision into logical sub-volumes, often referred to as *cells*. The dimension of a cell is chosen to be small enough for the sub-space to be homogeneous and for the enclosed entities to have almost instantaneous diffusion, so that the assumptions made by the Gillespie algorithm are valid inside a single cell; furthermore, spatial information is added to the system by duplicating every species S. New species with the same characteristics of S and with an index identifying its position on the grid are added to the system $(S_1, S_2, ..., S_n)$; diffusion is represented by first-order reactions among species. This method, proposed by Bernstein [20], is depicted in Fig. 1.

The advantage of this approach is that the algorithm in charge of simulating the reaction-diffusion system does not change; it is possible to add more species to model the molecules in different compartments and add reactions to "diffuse" between adjacent compartments, and then to use the existing tools and algorithms to simulate the modified system.

An efficient implementation for simulating reactive-diffusive systems by using spatial structures is used in the *next subvolume method* (Elf et al. [21]). The

underlying theory is the same utilized by Bernstein, as both are based on the exact realizations of the Markov process described by the Reaction Diffusion Master Equation. The algorithm uses three data structures: (i) a *connectivity matrix*, (ii) an *event queue* and (iii) a *configuration matrix*, used to naturally partition reactions into sub-volumes. Instead of mapping movements of entities using different species, the *direct method* [17] is used on each sub-volume to compute the time for the next event, i.e. a chemical reaction or a diffusion event. Then, the *next reaction method* [22] is used to identify the sub-volume where the first event will occur. The event is simulated, then the reaction and diffusion times in the volume (or volumes, in case of diffusion) are updated using the *direct method* again.

3 Discrete Event Simulation (DES)

In DES, the life of a *system* is modelled as a sequence of timed events. With this approach, a system is set up by a collection of *processes* $P = \{p_1, p_2, \ldots\}$ and of *activities* or *events* $E = \{e_1, e_2, \ldots\}$. A *process* is fully characterized by a finite set of *states* $S = \{s, s', \ldots\}$. At any given time, each process has exactly one *active state*. Each state s has a set of *actions* $A_s = \{\alpha_s, \alpha'_s, \ldots\}$ that can be performed when the process is in that state; the aim of an action is to change the current active state. *Activities* or *events* are sets of actions that are executed together to transform the state of the system. Here, we refer to the state of a system z as the collection of all the active states of the processes in the system. A *run* is thus meant as a sequence of interleaved system states and events: $r : z_0|e_0 \rightarrow z_1|e_1 \rightarrow z_2|e_2 \ldots z_{(u-1)}|e_{(u-1)} \rightarrow z_u$. As opposed to continuous simulation, in discrete event simulation, state changes of the simulated system are assumed to happen at discrete points of the virtual time and are thus controlled by uncontinuous functions, resulting in a succession of *events*.

DES can be used to simulate stochastic processes. In a stochastic process, each state is partially but not fully determined by the previous one. Typically, a stochastic process can have one or more deterministic *arguments*[1] and their values range over an index collection of non-deterministic random variables X_i with certain probability distributions. Such functions are equally known as *realisations* or *simple paths*. The view of a stochastic process as an indexed collection of random variables is the most common one. The *events* to be executed are bound to the set of random variables X_i, that determine which event will be executed and when. A simulation executes events in nondecreasing time-stamp order so that the virtual time (the time-stamp of the last executed event) never decreases.

Indeed, the occurrence of an event typically causes four actions: (a) progression of the virtual time to the *timestamp* of the simulated event; (b) changes of the state of the simulated system; (c) scheduling of new events and (d) descheduling of other events. Thus the basic data structure of a DES program consists of: (i) a virtual simulation clock; (ii) a timestamp ordered list of pending events and (iii) the state variables.

[1] We consider the *time* as always present among the arguments.

3.1 Parallel and Distributed Discrete Event Simulation (PDES and DDES)

In this summary, we deal with *parallelism at model function level*. In particular, we focus on methods which make intensive use of multiprocessors architectures for DES and which can be classified in between the following two classes: *parallel discrete event simulation* (PDES) and *distributed discrete event simulation* (DDES).

In PDES and DDES, a simulation model is partitioned into regions or domains[2]. Each region is simulated by a so-called *logical process* (LP). Each LP consists of [23]: (i) a spatial region R_i of the simulated system; (ii) a simulation engine SE_i executing the events belonging to the region R_i and (iii) a communication interface, enabling LPs to send messages to and receive messages from other LPs.

LPs are mapped onto distinct processors with (as an assumption) no common memory. Thus, every LP can only access a subset of the state variables $S_i \subset S$, disjoint to the state variables assigned to the other LPs. The simulation engine SE_i of each LP processes two kinds of events: *internal* events which have no direct causal impact on the state variables held in the other LPs and *external* events that can change the state variables in one or more other LPs. If an external event is processed, the LP holding the state variables that are to be changed is informed through a message sent by the LP. The message routing between the LPs is done by a communication system, connecting the LPs. Incoming messages are stored in input queues, one for each sending process.

Due to different virtual time progression within the various LPs, the causality principle is hard to be guaranteed and special considerations have to be made to obtain the same simulation results from DDES as from sequential DES. The two most commonly used synchronization protocols in DDES are: (i) The *conservative* (or Chandy-Misra) synchronization protocol developed by Chandy and Misra [24], [25] and (ii) the *optimistic* (or time warping) simulation protocol based on the virtual time paradigm proposed by Jefferson [26].

3.2 Conservative vs. Optimistic

The basic idea of the conservative protocol is to absolutely avoid the occurrence of causality violations. It is granted by strictly freezing the computation of an event e with virtual time (VT) t_e until when no messages with VT lower than t_e will be received. Under the assumption of FIFO message transport, this is achieved by only simulating an event if its VT is lower than the minimum of the timestamps of all events in all input queues.

An obvious problem arising in conservative simulation is the possibility of deadlocks [27]. Some deadlock resolution schemes have been developed during the last years. Among them, the more interesting are: [24] which avoids deadlock by the use of NULL-messages and [25] which detects and recovers deadlock, in advance. Some optimization protocols are discussed in [28], [29] (Null-messages

[2] For the purposes of this paper, only spatial decomposition is considered; however, the concepts illustrated here are also suitable for decompositions into general domains.

approach), in [30] (NULL-messages on request), in [31], [32] (lookahead computation), and in [33], [34] (local deadlock detection).

In contrast to the conservative protocol, there is no blocking mechanism in the optimistic one. An event is simulated even if it is not safe to process. Thus, causality errors are allowed to occur, but are later detected and solved. To guarantee causality, a mechanism called *time warp* or *rollback* has been designed. Time warp is optimistic in the sense that each processor P_0 executes events in timestamp order under the optimistic assumption that causality is not being violated. At any point, however, P_0 may receive a straggler event E, that should have been executed before the last several events already executed by P_0. In this case, it rolls back to a check pointed system state that corresponds to a time-stamp which is a global minimum among all VT (global virtual time) and less than the straggler's time-stamp. Processor P_0 resumes its execution from this point, and P_0 processes E, in the right time-stamp order.

A successful optimistic DDES minimizes the runtime costs of (i) state-saving system state, (ii) rollback, (iii) global virtual time (gvt) computation, and (iv) interprocessor communication.

3.3 Characterization of the Gillespie SSA as a PDES Algorithm

From a computational point of view, a biochemical system designed to be simulated with SSA can be seen as a collection of interacting processes, where each process can be in a different state among a set of discrete states. In this view, *biochemical species* are treated as *processes* that are able to perform a set of actions, changing their state in response to an external or internal action; *reactions* can be codified as *events* that are composed of a number of complementary actions, so that the execution of a reaction results in a simulation event that executes two (in the case of mono-molecular reactions) or more (in the case of bi-molecular reactions) actions in two or more processes (see Fig. 2).

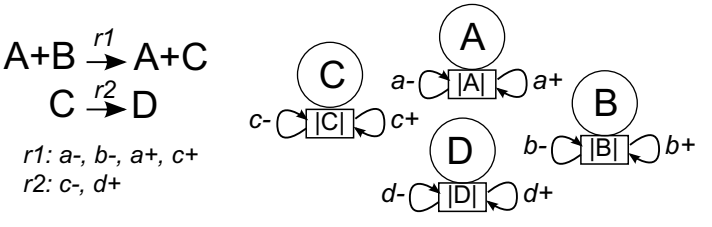


Fig. 2. A set of species and a set of reaction (left) represented as a set of processes and events (right). Each event is composed by two or more actions that modify the state of each process, typically decreasing or increasing the counter for the cardinality of the corresponding species.

So, a *biochemical system* $S = (P, E)$ can be seen as a set of processes $P = \{p_1, ..., p_n\}$, each holding a set of states and a set of actions that can be performed to modify its state, and a set of events $E = \{e_1, ..., e_m\}$, each composed by a set of

actions; typically, for every process p there will be two actions $(a+, a-)$ in charge of decreasing and increasing the counter for the cardinality of the corresponding species. However, it is possible and sometimes useful to add additional state variables and corresponding actions.

It is easy to see that, following this computational view, the simulation of a biochemical system with the Gillespie algorithm becomes a DES, where event times are generated by sampling an exponential distribution. The fact that times are generated by an exponential distribution leads to some insights in how this particular DES can be parallelised. In particular, we will show that it is almost never convenient to parallelize biochemical systems by using a conservative approach. In support of our analysis, we shall consider a *dependency graph* between events, defined as the graph of reactions introduced by Gibson and Bruck [22].

Definition 1. *Let* $Reactants(e)$ *and* $Products(e)$ *be the sets of reactants and products, respectively, involved in the event* e*.*

Here, for *reactants* we indicate the processes whose actions decrease the cardinality of their state variable, identified with the name of the process and a '-' suffix. For example, the event *e1* in Fig. 3 is composed by the actions $\{a-, b-, c+\}$; the actions $a-$ and $b-$ modify the state of A and B, so $Reactants(e1) = \{A, B\}$. *Products* are defined in a similar way as the processes whose actions increase the cardinality of their state variable.

Definition 2. *Let* $DependsOn(e)$ *be the set of processes whose state change affects the execution time of the event* e*, and* $Affects(e)$ *the set of processes whose state changes when an event is executed.*

Following the description of the SSA given in Sec. 2, $Reactants(e) = DependsOn(e)$. Typically, $Affects(e) = Reactants(e) \cup Products(e)$, or better, the set of processes on which the actions in e act. Sometimes, when two actions are complementary (i.e. one cancels the effects of the other), the set can be a little smaller. This is the case of the event $e3$ in Fig. 3, where $e-$ cancels $e+$ and $Affects(e3)$ can be reduced to $\{D, F\}$.

Definition 3 (Dependency graph). *The* dependency graph *of a biochemical system* S *is a directed graph* $G(V, E)$ *in which the set of nodes* V *corresponds to the set of events and there is a directed edge between each pair of nodes* $(V(e1), V(e2))$ *if and only if* $Affects(e1) \cap DependsOn(e2) \neq \emptyset$

The dependency graph can be used to show that the dependencies within reactions, united with the times sampled from an exponential distribution, in many cases lead to the need for sequential execution.

Definition 4. *Considering a system* S*, its dependency graph can be partitioned into a set of* strongly connected components*. We call the set of processes and events belonging to a strongly connected component of cardinality greater than one a* subsystem *(see Fig. 4).*

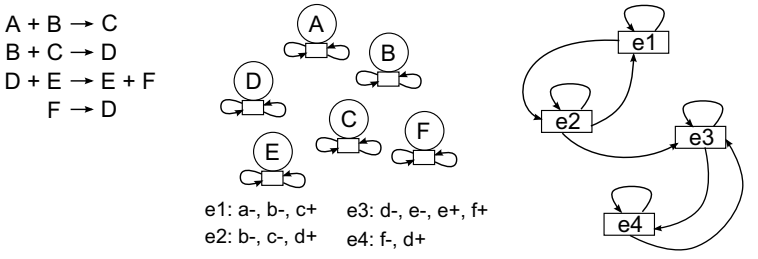


Fig. 3. The dependency graph (right) for a simple biochemical system (left)

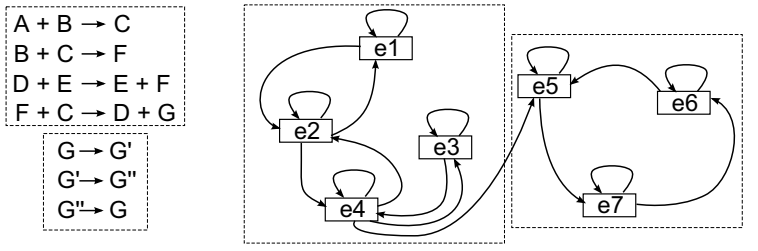


Fig. 4. A biochemical system partitioned into subsystems

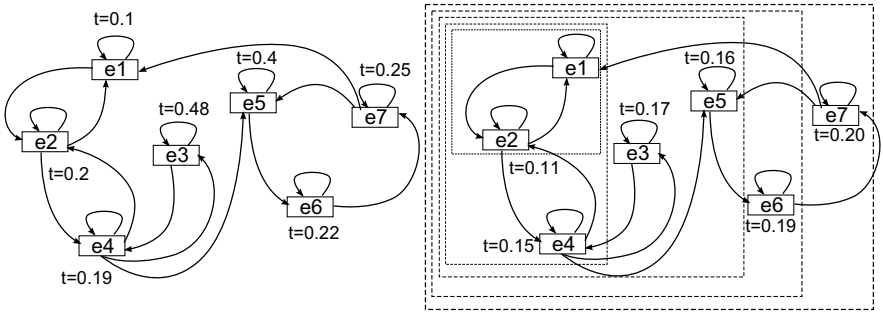


Fig. 5. Due to the exponential distribution used to generate execution times, the execution of an event can lead to the re-computation of the times of all the events in the same subsystem during the successive simulation steps

Proposition 1. *The execution of an event may lead to the need to recompute the next execution time for all the events in a subsystem.*

Whenever an event is executed, the next execution time of the events depending on it, i.e. its neighbours in the dependency graph, must be updated. Since times are exponentially distributed, there is no lower bound that guarantees us that the times we are going to recompute will be higher than a certain threshold.

The events with the new, lower, timestamps will in turn lead to the need for recomputing the time of other events, with the possibility of generating lower timestamps for the events they affect. By definition, in a strongly connected

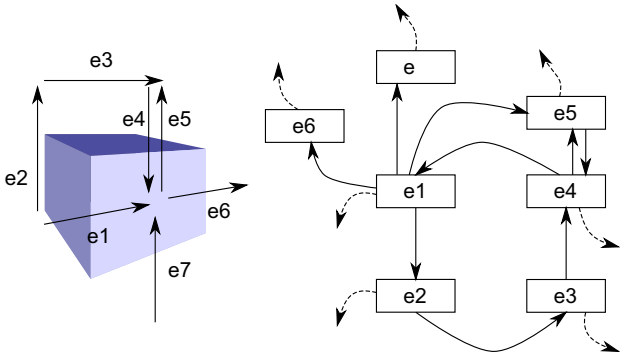


Fig. 6. Due to the diffusion events, a reaction-diffusion system has only one single subsystem

component there exists a path between any two vertexes, so it is possible that the generation of new times ripples and affects all the events in the subsystem (see Fig. 5). From this proposition, we can immediately derive two corollaries:

Corollary 1. *In a subsystem, the absence of causality errors is guaranteed whenever actions are executed in increasing time stamp order (*zero lookahead*).*

Corollary 2. *Since the absence of causality errors is guaranteed only if actions are executed one after the other, a pure conservative approach to PDES -which allows actions to be executed only when they cannot incur in causality errors- has a lookahead of zero, leading to a* serialized *execution where no speedup is possible.*

In [35], the authors considered several techniques for obtaining the *lookahead* necessary for concurrent execution of events under the conservative approach, such as artificially inserting lookahead into the computation and relaxing ordering constraint. We examined these approaches as well, and we came to the conclusion that using these techniques would lead to unacceptable compromises concerning the accuracy of the simulation.

An alternative approach for having some lookahead even in presence of exponential distributed random numbers is *pre-sampling*. Pre-sampling is a technique proposed by Nicol [36] for computing lookaheads in queueing network simulations with exponential distributed service time, and then used also for federated military simulations by Loper and Fujimoto [37]. At a glance, this technique seems to be applicable even in our domain. It carries a number of problems that makes it infeasible for our simulations. As noted by Nicol and Fujimoto, the service time variation has a strong effect on speedup. Under high variation, very small lookahead values are possible, meaning that lookahead is computed more often, thereby incurring in increased overhead. Furthermore, they also notice that rich interconnections between simulated entities, such as those used for simulating a continuous spatial environment, cause increased uncertainty in future behavior, resulting again in small lookaheads, with poor performances especially when using exponential distributed times.

Fujimoto conclusions that this technique requires (i) fixed sized time intervals, (ii) the same distribution for all messages, (iii) precise timestamps with few random number samples and (iv) knowledge concerning the number of messages produced in the near future [37] convinced us to discard it, as reaction-diffusion biochemical simulations do not meet any of these requirements.

Indeed, it is possible to make two crucial observations about reactive-diffusive simulations: (a) in a reaction-diffusion systems where species are free to diffuse in every direction, the dependency graph for diffusive events is fully connected; thus, the whole system is made of a single big subsystem (see Fig. 6) and (b) many biological systems show a little number of big subsystems; compounds, molecules and enzymes in a cell are reused over and over, forming big interconnected networks with loops. Indeed, regulation and transcription processes are often based on feedback loops, that shows up as connected components (subsystems) of dependency graphs.

In conclusion, the SSA can be characterized as a DES. Of the two main approaches to parallelize DES, the optimistic one is the most promising: as the two corollaries show, a pure conservative approach, united with exponentially distributed times and the particular dependency structure of biochemical systems, is very likely to perform poorly.

4 An Optimistic Reaction-Diffusion Simulator

As a proof of concept, we designed and developed a parallel stochastic reaction-diffusion simulator. While designing the simulator, we kept three goals in mind: (a) *correctness*: the simulator must respect the assumptions underlying the Gillespie method and its extension; (b) *scalableness*: the addition of further processing power must result in an increased execution speed-up; (c) *fastness*: the speed measured after running on a single processor must be comparable with that would be achieved if the simulator was strictly sequential.

4.1 Distributed Simulator Design

The first goal can be met by implementing the extension of the Gillespie method with diffusion events we introduced in Sec. 2.1, and the second and third goals can be fulfilled by using an approach based on PDES with an optimistic scheduling policy, as discussed in Sec. 3.

Notice that these two objectives must be considered together, as they heavily influence each other. Some methods, like the one presented in [38], chose to ignore correctness, violating the assumptions made by Gillespie and the properties stated by Bernstein with the aim to obtain fast parallel execution through volume subdivision. The algorithm, as the authors themselves say, can be useful in some cases, but it is not correct in a general sense. Indeed, when the spatial localization of molecules becomes important for the purposes of the experiment, the algorithm produces incorrect results.

For an effective implementation of the simulation algorithm as a PDES, spatial structures and partition of reactions into sub-volumes must be provided.

Moreover, state information should be maintained in a decentralized way, avoiding to keep global shared state informations, whenever it is possible. Partial local state can be processed and updated concurrently by different processors. The Next Subvolume Method (NSM) and algorithms derived from it employ spatial partitioning into sub-volumes, but they maintain information of execution times in global data structures; therefore they are not immediately adaptable to a parallel environment (even if a distributed version of the algorithm was recently proposed by Jeschke et. al. [35]).

We take a slightly different approach with respect to the NSM; we also divide reactions into sub-volumes (*cells*), but we consider each cell on the two or three-dimensional grid as an almost autonomous entity. We assume that every cell knows and stores its local information: concentrations of species, diffusion and reaction rates, next reaction time, as well as references to its neighbours. In each cell there are some dependency relations, both between species inside the same cell and between those in neighbour cells that can diffuse into it (see Fig. 7a). We have noticed that each cell on the grid can *evolve* (i.e. execute simulation events) independently from the other cells if the executed events do not violate the restrictions imposed by the dependencies. Following the optimistic approach, we let each cell evolve independently, up to a diffusion event occurs. When a neighbour notifies to the current cell a diffusion event with a clock T_{diff} smaller than the current clock T_{act}, reactions with times between T_{diff} and T_{act} are marked as *straggler*. So, we rollback every action executed within T_{diff} and T_{act}, recompute propensities and reaction times and restart the simulation of the events in that cell from time T_{diff}.

In our initial implementation, each cell was mapped onto a logical process (LP). The assignment of logical processes to physical processes may be done either dynamically, possibly by using a load balancing algorithm, or statically, by exploiting spatial locality to reduce communication overhead.

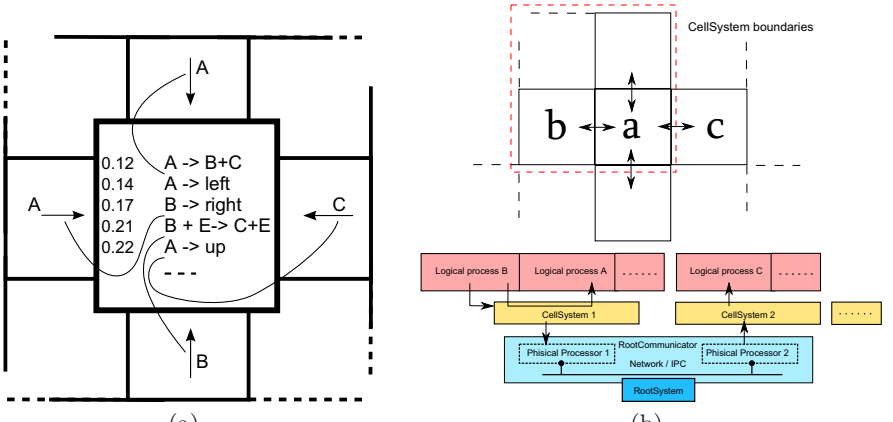


Fig. 7. Each cell is modelled as a process in a PDES (a). Cells are grouped into systems in order to reduce communication overhead (b).

4.2 Performance Considerations

The third goal, *fastness*, is not easy to achieve because a lot of practical, real world considerations have to be taken into account. The SSA was designed to run efficiently on hardware of the late '70; and it is indeed very efficient. An efficient implementation of the Gillespie algorithm can process and simulate roughly 10^5 reaction events per second; that is, a simulation loop takes approximately 10000-30000 CPU cycles to execute. Since a simulation loop is so fast, it is really difficult to speed it up by means of a parallel architecture. Execution of diffusion or reaction events on different processors requires synchronization in order to exchange messages. In the best of the hypotheses, processes can run on a single multi-core machine, where communication is done using shared memory and mutexes. According to the literature and to our own tests, even in this case the mere cost of context switching and proceeding the execution on a different thread (roughly 5000 CPU cycles) can easily result comparable to the loop time (see Fig. 8).

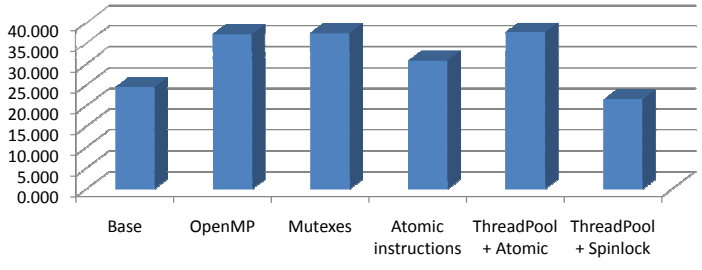


Fig. 8. The execution time of a parallel simulation (running on 2 processors) using various techniques of synchronization and inter-thread communication, compared to a serial simulation (Base). Notice that the overhead for running on multiple threads actually *increases* the execution times in all but the last case, where we used a pool of threads and hand-written assembly code for synchronization.

On a shared memory architecture, the problem is even worse. Even if current HPC architectures can rely upon very low-latency connections and upon very efficient message passing implementations (like the MPI interface we used), communication overheads can vanish any performance gain. For this reason, we chose a coarser granularity, in order to reduce the overhead to the minimum. For this reason, we designed our parallel simulator in a hierarchical way (Fig. 7b): cells are grouped into *Cell Systems*, that hold the partial state for a set of spatially contiguous cells. *Cell Systems* are then grouped and driven by a *Root System* that holds some topological information on the Cell Systems and that caches some essential information on the system global state. Every *System* has a specialized communicator. The *Root System* has a communicator based on MPI to let the *Cell Systems* it manages to communicate with each other across processor and machine boundaries. The *Cell Systems* have a single threaded, shared memory communicator in charge of maximizing the performance and reduce the

overhead on a single processor or core. A further layer can be added with the aim to manage groups of *Cell Systems* which execute on different CPUs or cores on the same computation node, i.e. on a machine that shares the same memory and that does not need for network communication or message passing in case of inter-groups interactions.

```
CellSystem ():
  while true do
    NextAction := FastestCell().FastestAction;
    StateChange := Action.Execute();
    History.Add(StateChange);
    UpdateClock(StateChange);
    if Action.IsDiffusion()
      if Action.TargetCell ∉ CellSystem.Cells
        RootComm.Notify(StateChange);

      else
        Action.TargetCell.Notify(StateChange);

    if RootComm.HasNotification
      Event := RootComm.HasNotification;
      switch Event.Type
        case Rollback :
          DoRollback(Event.Time);

        case Diffusion :
          Event.TargetCell.Notify(
                        Event.DiffusionAction);
```

```
RootSystem ():
  while true do
    Timer := StartTimer();
    Event := WaitForEvents(Timer, RootComm);
    switch Event.WakeReason
      case TimerTick :
        SendCheckpointCommand(GlobalTime);
        SystemState := RecvCheckpointData();
        DoCheckpoint(SystemState);
      case Communication :
        switch Comm.Type
          case Error :
            BroadcastRollback(Comm.Time);

          case Diffusion :
            TrgtSystem :=
                  LookupSystem(Comm.SourceCell);
            TrgtSystem.ForwardDiffusion(Comm);
    CurrentGlobalTime := Event.UpdateTime();
```

Fig. 9. Pseudo-code for *CellSystem* and *RootSystem*

Cells and *Cell Systems* communicate through a consistent interface, that is *transparent*, and that allows cells to communicate any diffusion information without taking care of the hierarchy. To communicate a diffusion from the cell C_1 to the cell C_2, C_1 sends a message to its *Cell System*; if C_2 is on the same physical processor (i.e. it belongs to the same Cell System), the information is directly propagated. If instead the *Cell System* realizes that C_2 does not belong to the set of cells it manages, it forwards the information up to the next System, until it reaches a System that knows C_2 or until it reaches the *Root System*. In the second case, the information is propagated using inter-thread communication or MPI messages (see the pseudo-code in Fig. 9).

The *Root System* is also responsible for checkpointing the system state, computing the global virtual time and propagate rollbacks, if necessary. In order to minimize the interprocessor communication, each *Cell System* has an incremental state history held in its own memory, as a queue of performed events. During the checkpoint phase, the *Root System* receives the partial state histories from the *Cell Systems*, computes the GVT, and commit all the events up to it. The commit is done by saving the system state to disk in an incremental way or, in alternative, by reconstructing the complete state on the fly. Each *Cell System* can then flush its own history up to the new GVT.

When a *Cell System* receives a straggler event, first it examines its queue and then it marks all the events with time-stamps greater than the straggler's one, performing a very quick rollback. If these events involve other *Cell Systems*, it informs the *Root System*, that take care of forwarding any rollbacks to the

other *Cell System(s)*. These examine their partial state and, if necessary, perform rollback on their state history. At this point, the *Root System* informs the *Cell Systems* to resume their computations. Since cells are grouped into *Cell Systems* -which are called LPs in PDES terminology- secondary rollbacks, and thus propagation of rollbacks, are very infrequent.

Care is taken that the two operations of checkpointing and rollback do not interfere with each other by means of a barrier.

5 Example

The simulator we developed accepts an input file that specifies reactions, reaction rates and diffusion coefficients, as well as the initial location of the chemicals. We also developed a visualizer, that is able to read the execution traces produced by the simulator and display them as a 3D rendering of the simulated volume.

We tested our simulator with some models (enzymatic reactions, oscillatory networks, chemotaxis pathway) under realistic conditions: most or all the molecules not attached to membranes have been let to move and, mostly important, the diffusion coefficients have been set always higher or at least comparable to the reaction rates. Such conditions obviously increase the number of messages sent, making harder for our simulator to appropriately scale. However, it is fundamental to provide a realistic model that respects the assumptions we made [20].

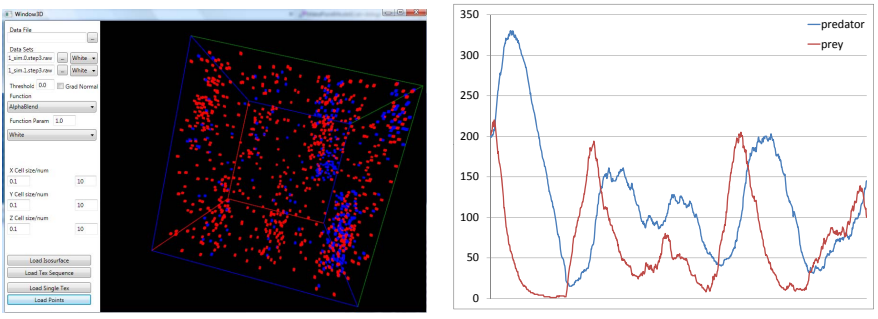
```
var predator : rate 100;
var prey : rate 100;

predator + prey -> predator + predator [55];
prey -> prey + prey [15];
predator -> nil [10];
```

Fig. 10. The input file for the 3D Lotka-Volterra model

As an example, we introduce a spatial version of the Lotka-Volterra predator-prey model. This model was chosen because it is simple but realistic; many other algorithms proposed in the same fields use completely artificial scenarios, with diffusion and reaction rates that are unrealistic (especially the ratio between them). Furthermore, a more complex model would not have contributed to the discussion.

This model allowed us to use rates taken from literature. With these using these rates, the model exhibits a different and interesting behaviour when ran in an environment that includes spatial information [39]. The results we obtained (see Fig. 11) are consistent with what we expected and with what is found in the literature [39].

This model allowed us to perform some initial performances estimations, listed in Table 1. We measured the execution time of the serial version of the algorithm, where all the inter-process communications were removed and substituted with direct manipulation of data structures in shared memory, and of the optimistic



(a) (b)

Fig. 11. A time-step of the Lotka-Volterra simulations (a) and the variation in cardinality of each species over time (b)

Table 1. Times in second for the execution of $5 \cdot 10^4$ simulation steps, 400 entities, (min/max/avg of five runs), and speedup for the parallel algorithm (*: on a 3D grid)

N cells	Serial	2 Cores		5 Cores		12 Cores	
256	1.5	14.8	0.1x	-	-	-	-
10000	13.1	10.7	1.22x	-	-	-	-
16384	17.4	(12.3/15.4/13.5)	1.29x	(9.1/10.1/9.4)	1.86x	-	-
26896	64.1	(34.7/42.8/38.7)	1.66x	(14.2/17.2/15.1)	4.25x	(7.0/9.4/8.0)	8.06x
32768*	75.8	(42.7/47.3/45.3)	1.67x	(18.4/20.7/19.2)	3.95x	(16.7/17.1/16.9)	4.49x

parallel algorithm. The hardware used for the simulation consists of PCs with AMD Opteron 64-bit CPUs at 2.4GHz, 4GB of Ram, interconnected with a 10Gbps Infiniband connection. We can observe that the overhead is significant when dealing with a small 16x16 2D grid, for a total of 256 node; the overhead starts to be less heavy starting with a 100x100 2D grid. As the grid becomes larger and larger, given a fixed number of subsystems, the diffusion events between different subsystems becomes less frequent. Note that a number of cells in the tens or hundreds of thousands is not unrealistic; for example, data for the last row of Table 1 were obtained for a 32x32x32 3D grid.

These preliminary figures are far from being complete performance measures, but they give an indication of the feasability of our approach. Since we do not have implemented or investigated work subdivision or load balancing techniques at this stage, we expect our method to perform well when each subsystem have to deal with the same amount of work, e.g. when there are not too big differences in the total concentration of elements (or *crowding*), with the same order of magnitude of activity. Note that concentration and activity of different compounds can vary significantly without any problems. Furthermore, using *Cell Systems* as computational units, we expect that our method performs well even in the hard but very common case of diffusion rates higher than reaction rates, as seen in the previous example.

6 Conclusion and Future Work

One of the obstacles on the way of computational systems biology is the scalability of the current approaches, i.e. their ability to deal with bigger and more complex models. With the aim to understand higher level behaviours, these complex models need for both powerful modelling tools and efficient simulation engines to analyse them.

In this paper we tackled the problem of designing a parallel simulator for biochemical systems, based on the theory developed by Gillespie, from both a theoretical and a practical point of view. The design of parallel and distributed algorithms requires indeed both a strong theoretical background, in order to guarantee that the designed algorithm is equivalent to the serial one, and a good deal of practical tricks and experience in order to make it really scalable and efficient.

Here we presented some first steps in this direction; although the results we obtained so far are promising, a lot of work needs to be done. In particular, Jeschke et. al. [35] conducted a parallel research on the same topic, focusing on the analysis of communication costs and on sizing of the window for optimistic execution in a distributed grid environment. It will be interesting to incorporate their studies and analysis of the window size to our framework, to see which are the differences between their grid-based and our HPC based approaches. Other problems we need to face are the analysis of the obtained data, whose dimension grows at an impressive rate when dealing with spatial simulations, load-balancing techniques for workload subdivision and analysis of the rollback mechanisms on different biochemical systems. Finally, we would like to perform an in-depth study of the performances, with different checkpoint frequencies, different number of nodes, different policy of cell allocation between nodes and different state saving strategies.

References

1. Fujimoto, R.M.: Parallel discrete event simulation. Comm. ACM 33(10), 30–53 (1990)
2. Ewing, G.C., McNickle, D., Pawlikowski, L.: Multiple replications in parallel: Distributed generation of data for speeding up quantitative stochastic simulation. In: Proceedings of the 15th Congress of Int. Association for Matemathics and Computer in Simulation, pp. 397–402 (1997)
3. Glynn, P.W., Heidelberger, P.: Analysis of initial transient deletion for parallel steady-state simulations. SIAM J. Scientific Stat. Computing 13(4), 904–922 (1992)
4. Newman, M.E.J., Barkema, G.T.: Monte Carlo Methods in Statistical Physics. Oxford University Press, Oxford (2000)
5. Glynn, P.W., Heidelberger, P.: Analysis of parallel replicated simulations under a completion time constraint. ACM TOMACS 1(1), 3–23 (1991)
6. Glynn, P.W., Heidelberger, P.: Experiments with initial transient deletion for parallel, replicated steady-state simulations. Management Science 38(3), 400–418 (1992)
7. Lin, Y.B.: Parallel independent replicated simulation on a network of workstations. ACM SIGSIM Simulation Digest 24(1), 73–80 (1994)

8. Yau, V.: Automating parallel simulation using parallel time streams. ACM TOMACS 9(2), 171–201 (1999)
9. Hybinette, M., Fujimoto, R.M.: Cloning parallel simulations. ACM TOMACS 11(4), 378–407 (2001)
10. Bononi, L., Bracuto, M., D'Angelo, G., Donatiello, L.: Concurrent replication of parallel and distributed simulation. In: Proceedings of the 19th ACM/IEEE/SCS PADS Workshop, pp. 430–436 (2005)
11. Streltsov, S., Vakili, P.: Parallel replicated simulation of markov chains: implementation and variance reduction. In: Proceedings of the 25th conference on Winter simulation, pp. 430–436 (1993)
12. Tian, T., Burrage, K.: Parallel implementation of stochastic simulation for large-scale cellular processes. In: Proceedings of of Eighth International Conference on High-Performance Computing in Asia-Pacific Region, pp. 621–626 (2005)
13. Burrage, K., Burrage, P.M., Hamilton, N., Tian, T.: Computer-intensive simulations for cellular models. In: Parallel Computing in Bioinformatics and Computational Biology, pp. 79–119 (2006)
14. Schwehm, M.: Parallel stochastic simulation of whole-cell models. In: Proceedings of ICSB, pp. 333–341 (2001)
15. Mazza, T., Guido, R.: Guidelines for parallel simulation of biological reactive systems. In: Proceedings of NETTAB 2008, Bioinformatics Methods for Biomedical Complex System Applications, pp. 83–85 (2008)
16. Gillespie, D.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J. Phys. Chem. 22, 403–434 (1976)
17. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)
18. McCammon, J.A., Harvey, S.C.: Dynamics of Proteins and Nucleic Acids. Cambridge University Press, Cambridge
19. Andrews, S.S., Bray, D.: Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. Phys. Biol. (1), 137–151 (2004)
20. Bernstein, D.: Exact stochastic simulation of coupled chemical reactions. PHYSICAL REVIEW E 71 (April 2005)
21. Elf, J., Ehrenberg, M.: Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. Syst. Biol. 1(2) (December 2004)
22. Gibson, M., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. J. Phys. Chem. 104, 1876–1889 (2000)
23. Ferscha, A.: Parallel and Distributed Simulation of Discrete Event Systems. McGraw-Hill, New York (1996)
24. Chandy, K.M., Misra, J.: Distributed simulation: A case study in design and verification of distributed programs. Comm. ACM 24(11), 198–206 (1981)
25. Chandy, K.M., Misra, J.: Asynchronous distributed simulation via a sequence of parallel computations. IEEE Trans. on Software Engineering SE-5(5), 440–452 (1979)
26. Jefferson, D.R.: Virtual time. ACM Transactions on Programming Languages and Computer Systems 7(3), 404–425 (1985)
27. Holt, R.C.: Some deadlock properties of computer systems. ACM Computing Surveys 4(3), 179–196 (1972)
28. Cai, W., Turner, S.J.: An algorithm for distributed discrete-event simulation - the 'carrier null message' approach. In: Proceedings of the SCS Multiconference on Distributed Simulation, vol. 22, pp. 3–8 (1990)
29. Wood, K.R., Turner, S.J.: A generalized carrier-null method for conservative parallel simulation. In: Proceedings of the 8th PADS Workshop, pp. 50–57 (1994)

30. Bain, W.L., Scott, D.S.: An algorithm for time synchronization in distributed discret event simulation. In: Proceedings of the SCS Multiconference on Distributed Simulation, vol. 19, pp. 30–33 (1988)
31. Groselj, B., Tropper, C.: The time-of-next-event algorithm. In: Proceedings of the SCS Multiconference on Distributed Simulation, vol. 19, pp. 25–29 (1988)
32. Cota, B.A., Sargent, R.G.: A framework for automatic lookahead computation in conservative distributed simulations. In: Proceedings of the SCS Multiconference on Distributed Simulation, vol. 22, pp. 56–59 (1990)
33. Prakash, A., Ramamoorthy, C.V.: Hierarchical distributed simulations. In: Proceedings of the 8th International Conference on Distributed Computing Systems, pp. 341–348 (1988)
34. Rukoz, M.: Hierarchical deadlock detection for nested transactions. Distributed Computing 4, 123–129 (1991)
35. Jeschke, M., Ewald, R., Park, A., Fujimoto, R., Uhrmacher, A.: Parallel and distributed spatial simulation of chemical reactions. In: Proceedings of the 22nd ACM/IEEE/SCS PADS Workshop (2008)
36. Nicol, D.M.: Parallel discrete-event simulation of fcfs stochastic queueing networks. SIGPLAN Not. 23(9), 124–137 (1988)
37. Loper, M.L., Fujimoto, R.M.: Pre-sampling as an approach for exploiting temporal uncertainty. In: PADS 2000, pp. 157–164 (2000)
38. Ridwan, A., Krishnan, A., Dhar, P.: A parallel implementation of gillespie's direct method. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3037, pp. 284–291. Springer, Heidelberg (2004)
39. Schinazi, R.B.: Predator-prey and host-parasite spatial stochastic models. The Annals of Applied Probability 7(1), 1–9 (1997)