Model-Driven Parallel and Distributed Stochastic Simulation of Chemical Reaction Networks

Simone Bauco*, Federica Montesano†, Adriano Pimpini*, Romolo Marotta*, and Alessandro Pellegrini*
*Tor Vergata University of Rome, Rome, Italy
†IASI-CNR, Rome, Italy

***Abstract*—Biochemical systems are characterized by rich stochastic behaviors and complex interaction patterns, often modeled through rule-based frameworks such as BioNetGen. Despite the expressiveness of such high-level specifications, simulating large-scale models remains computationally prohibitive. In this paper, we present a model-driven framework for the parallel and distributed execution of stochastic chemical reaction networks, enabled by a model-to-model transformation of BioNetGen descriptions. The transformation produces an intermediate representation based on the Actor Model, in which actors encapsulate local state and asynchronous communication, aligning with the concurrency of biochemical processes. From this representation, we generate executable code targeting ROOT-Sim, a speculative parallel discrete-event simulation (PDES) environment based on Time Warp. We propose an exact parallel implementation of the Stochastic Simulation Algorithm (SSA), employing reaction partitioning to minimize inter-process dependencies and leveraging an event-exchange protocol to ensure consistency and atomicity of reactant consumption across logical processes. We introduce refined rollback and reaction rescheduling mechanisms to address potential correctness issues such as reactant overconsumption. Extensive experiments on well-established models, such as FcεRI, demonstrate substantial speedups over sequential methods.**

***Index Terms*—SSA, BioNetGen, PDES, Model-Driven Engineering, Actor Model, Time Warp, Chemical Reaction Networks, Rule-Based Modelling.**

I. Introduction

Biological systems are characterized by intricate behaviours emerging from extensive interactions among numerous biochemical entities. Computational modelling of biochemical reaction networks is thus indispensable for researchers aiming to understand, predict, and manipulate biological processes. BioNetGen [1], a rule-based modelling framework, allows researchers to succinctly represent complex biochemical networks through its dedicated domain-specific language (DSL). Despite the expressiveness and usability of BioNetGen, efficiently simulating large-scale reaction networks remains challenging due to their inherent stochastic nature and intensive computational demands.

To overcome these computational hurdles, we propose a framework that leverages model-driven engineering (MDE) [2] in conjunction with the Actor model paradigm [3], [4]. Our approach involves a model-to-model (M2M) transformation that converts BioNetGen models into an intermediate representation explicitly based on the Actor model [5]. In this paradigm, computations are encapsulated within autonomous entities, known as actors, which operate concurrently, maintain local states, and communicate through asynchronous message passing. This structure naturally aligns with the inherent parallelism of biochemical reaction networks, allowing each actor to represent either individual biochemical entities or distinct reaction processes. The resulting actor-based intermediate representation facilitates automated translation into executable code optimized for parallel and distributed computational environments [5].

To support the execution of BioNetGen models on heterogeneous architectures, we also provide a parallel exact implementation of the Stochastic Simulation Algorithm (SSA) [6], using the next-reaction method [7]. SSA precisely models stochastic biochemical dynamics, capturing critical random fluctuations and rare events that significantly influence biological systems. Exact SSA methods are essential, particularly for accurately simulating systems with low molecular concentrations or reactions occurring infrequently but with considerable biological impact. Consequently, exact SSA simulations are fundamental for generating reliable predictions that can be experimentally validated.

Despite its advantages, efficient execution of SSA within distributed computing environments introduces notable technical complexities, particularly concerning state synchronization and consistency. Accurate synchronization of molecular populations across computational nodes is crucial, as reaction probabilities depend on timely and accurate data. Delays or errors in synchronization can lead to significant deviations from accurate stochastic dynamics. Additionally, distributed execution risks the occurrence of "double spending", where multiple nodes concurrently consume identical reactants, necessitating possibly complex synchronization and consistency controls.

To address these challenges, we tested various reaction partitioning strategies to strategically distribute reactions across computational nodes, aiming to minimize the number of shared reactants and thus reduce synchronization overhead. Nevertheless, certain reactants must remain shared to preserve parallelism. In these cases, we ensure correctness by employing an event-exchange protocol guaranteeing atomicity and synchronized consumption of reactants. This protocol efficiently prevents double spending and maintains simulation consistency across distributed nodes by leveraging the actor model's asynchronous communication capabilities.

Furthermore, to optimize simulation efficiency, we integrate speculative execution enabled by the Time Warp synchronization protocol [8]. Speculative execution permits computational

nodes to optimistically advance their simulation states without immediate synchronization, maximizing computational throughput and minimizing idle time. When speculative computations lead to inconsistencies, Time Warp addresses them through rollback mechanisms, reverting the simulation state to the last observed consistent one. This balance between computational performance and simulation accuracy can maximize resource utilization and thus performance.

We evaluate our framework using various biochemical models from the literature, representing a broad spectrum of complexity and computational demands. Our results demonstrate significant performance improvements relative to traditional sequential SSA methods, validating scalability and efficiency.

Our implementation is released to the public as open-source software [1].

The remainder of this paper is structured as follows. We discuss related work in Section II. The approach to efficiently execute parallel/distributed simulations of chemical reaction networks, and the transformations used to translate a set of reactions expressed in the BioNetGen DSL into our intermediate representation based on the Actor Model, are discussed in Section III, along with the details on the parallel exact SSA algorithm used to support the execution. The experimental evaluation is provided in Section IV.

II. Related Work

In this paper, we have a twofold goal: making a simple BioNetGen-like description of a reaction network compatible with speculative PDES simulators, and supporting their exact parallel/distributed simulations by means of an SSA algorithm.

We tackle the first goal by using MDE techniques to automate simulation model generation. Several recent works have explored this pathway, which has been regarded as highly promising. In [9], the authors have explored transformations to automatically generate model versions suitable for heterogeneous architectures. The work in [10] explicitly targets interoperability across simulators, enabling simulationists to execute identical models across multiple runtime environments. The work in [11] advocates an MDE-based domain-specific modeling approach focused on trustworthy agent-based simulations by improving traceability and reproducibility, particularly for critical decision-making contexts. We share the underlying goal of these works, but focus on the biochemical reaction domain.

Our second goal is to provide efficient execution of reaction networks using algorithms from the family of SSA algorithms. This problem has been of great interest, particularly because the strong connection between the subsequent reaction and the state of the entire system can make parallelization difficult. In particular, accuracy and computational cost have been recognized [12] as critical factors for an exact execution of SSA algorithms, which we explicitly consider in this paper, in contrast to approximate approaches.

[1] https://github.com/DomainProject/ParallelSSA.

Some works [13]–[15] have provided early optimistic implementations with centralized coordination and lease-based speculation. They share the use of master-worker approaches, centralized components (e.g., brokers or message servers), coarse-grained leasing or speculative windows, and explicit focus on limitations arising from centralized bottlenecks such as network round-trips, serial state uploads, and rollback cascades due to large speculative windows.

All these works demonstrated that exact spatial SSA can leverage optimistic Time-Warp synchronization across heterogeneous systems, but faced scalability limitations, especially under high rollback frequency. Our fully distributed Time Warp approach addresses these bottlenecks by clustering together reaction executions based on reactants, thus reducing message exchange and rollback probability. Relying on retractable messages also allows for reducing the burden at the model level, exploiting platform-provided facilities to quickly reorganize the future event set.

On the other hand, the works in [16]–[18] explicitly focus more on distributed simulations without centralized lease brokers, focusing on efficient management of rollbacks and memory management. In particular, these works aim to reduce the rollback overhead by employing incremental state saving or reverse computation techniques or relying on distributed GVT algorithms to reduce the memory footprint. Other works [19]–[21] introduce hybrid or controlled optimism schemes (e.g., Breathing Time Warp, dynamic local time windows) designed to explicitly limit speculative execution to avoid rollback explosions. They tend to evaluate more sophisticated synchronization optimizations such as event retraction throttling, selective rollback management, dynamic window-based throttling, and fine-grained rollback triggering conditions. As such, they are orthogonal to our proposal, because we focus on making the forward execution phase more efficient, by reducing the number of required interactions across Logical Processes (LPs), thanks to proper static allocation of the reactions. Moreover, we focus on the distributed interaction between the LPs employing other techniques like retractable events. The techniques studied in the above papers could also benefit from the proposals in this paper.

Few works [22], [23] study the possibility of exploiting accelerators (such as GPUs) or multicore machines to exploit hardware-specific optimizations, avoiding the burden of supporting rollbacks in distributed environments. Although we only perform an experimental evaluation of our proposal on CPUs, we do not require any specific hardware platform. Indeed, our proposal is compliant with the Time Warp synchronization protocol and could therefore be exploited by any platform abiding by it.

Hybrid methods and approximations have been studied in [24], [25]. Here, the authors exploit approximations or quasi-steady-state assumptions to improve performance at the expense of strict exactness or scalability. In particular, in [24] the authors propose a fully-sequential algorithm, while in [25], although the authors mention the Time Warp protocol, simulation trajectory corrections are strictly local: no state

saving, rollback, or global virtual time is maintained, execution remains single-threaded, and leap rejection is bounded by a constant to curb overhead. Approximation is also exploited in [26], where the authors rely on conservative synchronization, fixed time windows, and approximate reaction-diffusion decomposition strategies, offering scalability but sacrificing exactness or incurring load imbalance due to conservative policies. Unlike all these works, we tackle a fully parallel and distributed optimistic execution scenario, while avoiding any form of approximation.

The problem of handling the zero-lookahead nature of SSA (biochemical) simulations has been thoroughly addressed in [16], [21], [27]. These works mainly deal with explicit minimization of rollback overhead through refined window estimation, incremental state saving, or selective rollbacks. In general, they study synchronization algorithms that differ from those we rely upon. Many of the organizations of our model at the LP level are also compatible with these works.

We share some background, goals, and approaches with the theoretical work in [28]. This work, which is primarily conceptual, introduces an SSA scheme based on Time Warp synchronization, although no practical implementation nor benchmarking is provided, while we have one. Also, differently from [28], we adopt a different distribution of reactions to LPs, which could provide improved performance, while reducing the number of simulation objects that are required to carry on the execution—in particular, we only have LPs which manage reactions, while the algorithm in [28] also requires LPs that manage shared populations of reactants. Also, from the details provided in [28], it is possible that, if two reactions require the same reactant at the same time, the same reactant will be used by both reactions, making the simulation's result incorrect. In our proposal, we explicitly deal with such corner cases in an attempt to achieve higher accuracy.

III. Parallel Simulation of Reaction Networks

In this section, we discuss how we handle the BioNetGen DSL to translate it into a parallel/distributed simulation model of chemical reaction networks. We also illustrate how we support the execution of such models at runtime.

A. The DSL Metamodel

We allow specifying a chemical reaction network using the BioNetGen DSL, which is automatically transformed into a parallel/distributed simulation model. We have realized a BioNetGen DSL metamodel using JetBrains Meta Programming System (MPS) as the reference language workbench. The BioNetGen metamodel's fundamental concepts and relations are reported in Figure 1. This metamodel is then used to transform a reaction specification into a software artifact (a simulation model) that can be executed concurrently, also in a distributed environment.

The central concept of the metamodel, encapsulating the entire BioNetGen DSL, is the Script. The model is divided into sections, as originally defined by the BioNetGen DSL.

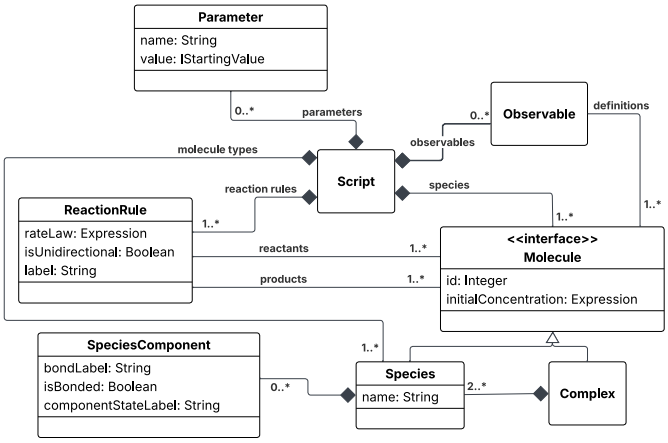


Fig. 1: The BioNetGen DSL Metamodel.

The *parameters* section declares all scalar quantities that may be used throughout the model, such as rate constants, initial concentrations, and other numerical values. Each parameter is assigned a symbolic name and a numeric value.

The model's fundamental molecular entities are specified in the *molecule types* section. For each entity, its binding sites, representing potential points of intermolecular interaction, and its internal states, denoting distinct biochemical configurations of those sites, are defined.

In BioNetGen, a molecule type represents a monomeric unit, which we call a Species in our model. Each Species includes binding sites, called SpeciesComponents, which may have optional state values and binding states. A Complex is formed when multiple Species bind together to create a larger structure.

While the *molecule types* section defines the building blocks and their component structure, it does not require all Complexes to be enumerated. Instead, these emerge dynamically through the reaction expansion, as will be discussed later.

The *species* section enumerates the molecules that are initially present in the system, together with their initial quantities, thus providing the initial conditions required to instantiate the reaction network or initiate a simulation.

The *reaction rules* section encodes the system's dynamic behavior in terms of pattern-based transformation rules. Each rule specifies a reactant pattern, a product pattern, and an associated rate law. Reaction rules may be reversible or irreversible and operate on molecular patterns rather than enumerated species, enabling the compact representation of systems with combinatorially large state spaces.

The *observables* section defines measurable quantities derived from the system state, such as the concentration or count of molecules with particular structural or state characteristics. Each observable is assigned a name and one or more reactant patterns to match. The value of the observable will be the sum of the populations of reactants matching the patterns.

B. The M2M Transformation

The M2M transformation that we have envisaged to transform a program expressed in the BioNetGet DSL into an executable artifact has to deal with several intricacies, proper to the DSL itself. Indeed, the DSL allows the specification of high-level rules that are easy to code for the domain expert, but must be transformed into the complete set of biochemically meaningful reactions that define the system's dynamical behavior. Therefore, we must expand the rules into the actual reactions, which are then mapped to the actor model for further transformation.

1) Reaction Expansion: We implemented a reaction rule expansion algorithm, similar in spirit to BioNetGen's, that generates all possible reactions and species reachable from an initial system description. The expansion algorithm proceeds iteratively and systematically constructs the complete reaction network implied by the rules and initial species, as defined in the subsections introduced. The algorithm operates as follows:

1) *Initialization*: The set of known species is initialized using the species defined in the `species` block. The initial reaction set is empty.
2) *Iterative expansion*: For each iteration (up to the user-defined maximum), the algorithm examines all *reaction rules* defined in the model. For each rule:
   a) Each *reactant pattern* in the rule (which may describe specific molecular components, states, and bond constraints) is matched against the current set of known species.
   b) *Pattern matching* identifies all *concrete species* that satisfy the structural and state constraints of the reactant pattern. If multiple reactant patterns are involved, the Cartesian product of all species matching each pattern is computed to enumerate all possible reactant n-tuples. In general, this is a problem of *subgraph isomorphism*, which we have solved by relying on the Ullman algorithm [29].
3) *Reaction instantiation*: For every valid combination of matching species, the rule is applied concretely to generate *new product species*. The transformation respects the binding, state changes, and topology specified in the rule. Each generated reaction and resulting species are:
   a) added to the chemical network only if they have not been previously encountered;
   b) used to update the global reaction and species lists.
4) *Termination*: The process repeats for the specified number of iterations or until no new species or reactions are generated.
5) *Observable expansion*: Once the whole network is constructed, each observable defined in the `observables` block is expanded by matching its reactant patterns against the final set of species. All species that match an observable's pattern are grouped as the *observed set* for that observable.

2) Mapping to the Actor Model: To transform a program implemented according to the above metamodel, we have relied on the Domain's Actor Language [5], by implementing a dedicated M2M transformation.

The resulting model defines a single actor type, representing a processing unit responsible for managing a specific set of reactions. The distribution of reactions across processing units can follow five strategies: a standard round-robin assignment based on a fixed number of units, a randomized round-robin assignment obtained by shuffling the set of reactions before applying a standard round-robin assignment, a batch assignment splitting the network into a fixed number of equal contiguous chunks, a grouping assignment that clusters all reactions sharing at least one reactant onto the same unit, or a graph-partitioning assignment trying to minimise dependencies between units.

Since only one actor type is defined, a single behaviour is generated, which is responsible for reaction scheduling, reaction processing, and population adjustment. This behaviour leverages the APIs provided by the runtime system developed within the simulation environment.

3) From the Actor Model to an Executable Simulation Model: To transform a reaction network expressed in the Actor Model, we exploit the Domain M2T [5]. With these, we target the ROme OpTimistic Simulator (ROOT-Sim) framework [30] as the underlying simulation runtime environment.

The Domain M2T transformations are based on the correspondence between the concept of actor and that of a Logical Process (LP) proper of DES. The actor's address is automatically mapped during the M2T transformation into an ID, which is used by ROOT-Sim to discriminate LPs and route events.

Each actor takes care of a subset of the reactions in the network. As such, all the actors share the very same behaviour, associated with the execution of the SSA algorithm and the management of the (local) population of reactants. Therefore, the M2T transformation generates a single routine to process each type of message (triggering of reactions, update of the local population), and then relies on the exact parallel SSA algorithm that we have devised. This algorithm is implemented as an external library, i.e. an opaque part of an actor's behaviour. Compiling the generated code against the ROOT-Sim and parallel SSA libraries generates a functional artifact that can simulate the evolution of the reactions initially specified in the BioNetGen DSL.

C. The Exact Parallel SSA Algorithm

In contrast to simulation methods that use deterministic rate equation models, that approximate reaction kinetics by assuming continuous concentrations and average behaviour, SSA directly addresses the inherent stochasticity and discreteness of chemical reactions by simulating a sequence of individual reaction events that occur probabilistically, based on reaction propensities, thus providing statistically exact trajectories of the reacting system's state in time.

The first algorithm to implement SSA was introduced by Gillespie [6]. This algorithm, called the *Direct Method* (DM),

Algorithm 1 SSA Next Reaction Method (Gibson and Bruck)

Procedure: SSA NEXT REACTION METHOD
Set initial species populations
Initialize dependency graph G
$t \leftarrow 0$ ▷ simulation time
for each reaction k **do**
  Compute initial propensity a_k
  Sample $r_k \sim \mathcal{U}(0,1)$
  Set $\tau_k \leftarrow t + \frac{1}{a_k} \ln\left(\frac{1}{r_k}\right)$
end for
while $t \leq$ max simulation time **do**
  Let $\mu \leftarrow \arg\min_k \tau_k$
  $t \leftarrow \tau_\mu$
  Update species populations according to reaction μ
  for each edge (μ, k) in the dependency graph G **do**
    Update a_k
    if $k = \mu$ **then**
      Sample new $r_k \sim \mathcal{U}(0,1)$
      Set $\tau_k \leftarrow t + \frac{1}{a_k} \ln\left(\frac{1}{r_k}\right)$
    else
      Update time without resampling:

$$\tau_k \leftarrow t + \frac{a_k^{\text{old}}}{a_k^{\text{new}}}(\tau_k - t)$$

    end if
  end for
end while
end procedure

simulates the chemical kinetics by sampling the time and identity of the next reaction based on the propensities. It therefore requires recalculating the total propensity and performing a linear search through all reactions at each step. While exact, its per-step complexity is $O(N)$, where N is the number of reactions, making it inefficient for large systems. Moreover, parallelizing is more difficult because the search requires a coherent view of the reactions and coordinated access to the data structures.

Gibson and Bruck [7] improved DM's efficiency by using a dependency graph, indexed priority queues, and scheduled reaction times, reducing per-step complexity to $O(\log N)$. This structure, named the *Next-Reaction Method* (NRM), enables event rescheduling without recomputing all propensities and is well-suited for parallelism, especially in spatial or multiscale models where localized reaction updates can be decoupled, allowing for asynchronous and distributed simulation with minimal global synchronization. We base our exact algorithm on NRM.

Each reaction is defined as a set of reactants (the input chemical species), a set of products (the output species), and a specific rate constant, relating the rate of the reaction to the quantities of reactants. For convenience, we also precompute a vector of stoichiometry changes, expressing the changes in the populations of the reactants consumed and produced. During the simulation, each reaction holds two dynamic properties: a propensity, which is the probability of occurrence, and the time at which it will next occur, if any. The propensity, denoted as a_k, is computed as:

$$a_k = c_k \cdot \prod_{i=1}^{n} (x_i)_{s_i} \tag{1}$$

where $(x_i)_{s_i} = x_i \cdot (x_i - 1) \cdots (x_i - s_i + 1)$, with c_k rate constants, x_i reactants population, s_i stoichiometry factors. The propensity computation is one of the crucial points of NRM, as the main difference between NRM and DM is the way propensities and reaction times are computed. In particular, the NRM generates only one uniform random number in each step, while the DM requires two. Consequently, the computation of the reaction time for each reaction becomes strictly dependent on the propensity for that reaction, while in the DM the reaction times are computed by choosing another random number and normalizing with respect to the sum of the propensities of all the reactions. The time of the next reaction in NRM, denoted as τ_j, is computed as:

$$\tau_k = \frac{1}{a_k} \cdot \ln\left(\frac{1}{r_k}\right) \tag{2}$$

where a_k is the propensity of reaction k and $r_k \sim \mathcal{U}(0,1)$.

The M2M transformations described in Section III-B provide us with a graph G between the different actors that characterizes the model's dependencies among reactions and species. The graph G is also partitioned into sub-networks of reactions that we can simulate in parallel. These sub-networks, thanks to the Domain M2T transformations described in Section III-B2, are mapped to Logical Processes (LPs), which are regarded by our exact-SSA library as *SSA simulation objects*.

Similarly to [28], we identify the species shared between multiple sub-networks of reactions but do not allocate them to a dedicated simulation object. Therefore, SSA objects are responsible for executing the algorithmic steps of the SSA algorithm, but also take on the responsibility of propagating notices of reaction executions to other SSA objects.

We use *retractable events* [31] (i.e., simulation events that can be removed from the system in case they are a-posteriori detected as erroneous by the simulation logic) to manage reaction execution events, while all the other events are regular messages. While one may argue that this could antagonise the simulation's speculation capabilities, we chose this approach because of the drastic reduction in the number of messages in the event queue and the leanness of implementation, which avoids costly slowdowns at runtime.

When the simulation starts, each LP populates its state and selects the first reaction to execute among the ones associated with it. At the beginning, the propensities and the reaction times for each reaction are computed, using Equations (1) and (2) described above. We find the next reaction to be executed r_μ (i.e. the one with minimum reaction time) and use the $schedule\_reaction$ method to schedule the execution of r_μ as a retractable event of type `execute_reaction`.

Upon receiving an event of type `execute_reaction`, the method $execute\_reaction$ is called to execute reaction r_μ. There, the LP responsible for executing r_μ, i.e. LP_e, first notifies other LPs affected by this reaction about the reaction execution. To do so, LP_e gathers the list of LPs that are *interested* in r_μ. A reaction r_d is *interested* in a reaction r_i if the execution of r_i modifies the population of some species used by r_d as input. An LP is *interested* in a reaction r_i if it manages a reaction r_d that is, in turn, *interested* in r_i. For each reaction, we pre-compute the dependency information by constructing the list of interested reactions, starting from the list of stoichiometry changes, which is in turn computed using the list of a reaction's inputs and outputs. LP_e sends an `apply_reaction` event to all LPs interested in r_μ, allowing them to update locally-stored population counts of each chemical species, which are the local views of the LP on the portion of simulation state it is interested in. The `apply_reaction` events are sent with zero-lookahead to ensure a consistent view of the global state across LPs. After this dispatching, the reaction is executed locally, by applying the stoichiometry changes to each reactant used and produced by the reaction.

After executing the reaction, the $select\_reaction\_after\_execution$ method is called in order to recompute reaction propensities and reaction times, as these depend on the possibly changed reactant populations. Finally, a zero-lookahead `schedule_reaction` event is scheduled to self. Note that $select\_reaction\_after\_execution$ is one of the two $select\_reaction\_x$ methods, which change based on the event types that led to having to select a reaction.

To handle a `schedule_reaction` event, the LP selects the reaction whose timestamp is closest in the future, and schedules it using the aforementioned $schedule\_reaction$ method.

Lastly, upon receiving an `apply_reaction` event, an LP must apply the population changes caused by the executed reaction to the locally tracked reactant populations. After updating the view on molecule counts, the LP uses the $select\_reaction\_after\_cancellation$ method to select the next reaction r_μ to be executed, and reschedules the retractable `execute_reaction` event in accordance with the (new) next reaction time τ_μ of r_μ. Similarly to $select\_reaction\_after\_execution$, $select\_reaction\_after\_cancellation$ recomputes reaction propensities and times.

The event ordering is paramount to guaranteeing execution correctness. The event precedence is as follows: `execute_reaction` precedes `apply_reaction`, which precedes `schedule_reaction`. This has to be respected because `apply_reaction` has to come after `execute_reaction`, so as not to alter reactant populations right before execution and both `execute_reaction` and `apply_reaction` modify the reactant populations, thus reaction scheduling with `schedule_reaction` has to necessarily come after them. However, we must add an important note about the special case where the last units of the reactant are used simultaneously.

A crucial point not considered in [28] is the population depletion upon reaction execution, an issue especially relevant when we consider parallel and speculative execution. For instance, given some reactant A, input of multiple reactions $r_{A,i}$, it can happen that two reactions $r_{A,1}$ and $r_{A,2}$ are fired at the same time by different LPs. While concurrent reactions can take place in SSA, the parallel nature of the presented approach means that edge cases can arise, in which the population of A is insufficient for both $r_{A,1}$ and $r_{A,2}$ to execute. In a (biased) sequential implementation, the second reaction would simply not be fired. However, in a parallel simulation, one has to devise a way to decide which reaction goes through and which does not.

For this reason, we see that the `execute_reaction` $\rightarrow$ `apply_reaction` $\rightarrow$ `schedule_reaction` ordering fails to correctly avoid the "double spending" of the last units of reactant on its own. We note that this corner case is likely very rare, but it could wreck simulation correctness.

To fix the issue, one of the following techniques must be employed:

1) The partitioning of the reaction graph G has to consider the usage of shared reactants. The only way for an LP to decide whether a reaction can be fired is to have the view of all the populations of reactants it uses. By assigning to a single LP all the reactions that share at least one input reactant, we completely avoid the possibility of scheduling two reactions that use the last unit of reactant at the same exact time, as the execution within a single LP is inherently sequential. We implemented this solution with the reaction clustering approach presented in Section III-B2. This however can lead to long chains of interdependent reactions, reducing the parallelisation opportunities even for larger models, and potentially leading to heavy workload imbalance.
2) A change in the tie-breaking rule, to include the sender LP's ID in the message ordering, before the event type. For simplicity and without loss of generality, suppose that a smaller ID is preferred by the tie-breaking. Suppose two LPs LP_1 and LP_2 manage reactions $r_{A,1}$ and $r_{A,2}$ respectively. Suppose that $r_{A,1}$ and $r_{A,2}$ both use reactant A. Should $r_{A,1}$ and $r_{A,2}$ be scheduled for the same time while not enough units of A are available for both reactions to execute, then LP_1's messages would take precedence over all of LP_2's messages. This way, LP_1 will handle an `execute_reaction` event e_1 locally, generating an `apply_reaction` event a_1 to send to LP_2, which will in turn execute a_1 from LP_1 over its own `execute_reaction` e_2, triggering a recalculation of reaction times, which will discover that $r_{A,2}$ is not fireable for lack of reagents, preventing e_2 from being scheduled in the first place. This is the solution we employed, as it was revealed to be the most efficient, while being less restrictive.
3) The usage of *superposition primitives* [32] offered by the underlying simulation runtime environment. These

allow the SSA library to look at all the simultaneous events, and possibly reorder them (or deny the execution of some of them) based on the current population.

We emphasise that this problem is not related to the absence of a dedicated object to manage the reactant population: even in this case, the same pattern could cause a "double spending". Again, deciding whether a reaction can be applied requires full knowledge of the current quantities of the reactants used: an object dedicated to managing the populations of shared reactants exclusively would not have access to enough information to decide whether to apply a reaction or not. To have enough information, the object would have to stop managing only shared reactant populations, while requiring a reaction network partitioning in the fashion of the first proposed technique, effectively moving the entirety of reactant population management away from the SSA objects.

A second point that deserves attention is that reactions may not be fireable because of a lack of reactants. An immediate implementation (which is also used in [28], without considering the case of non-fireable reactions) for $select\_reaction\_after\_execution$ and $select\_reaction\_after\_cancellation$, entails computing the new propensity of each involved reaction either from scratch using Equation (2), taking a new random number if the target reaction was the one just executed or previously non-fireable, or update it using Equation (3) otherwise. A more efficient solution (provided in the original NRM paper [7]) entails keeping track of the last non-zero propensity and reaction time interval, to use for computing the new reaction time once the required reactants are available again, and avoid drawing more random numbers. The formula for updating the reaction times without drawing random numbers is reported in Equation (3):

$$\tau_k = \frac{a_k}{\bar{a_k}} \cdot (\tau_k - t) + t \tag{3}$$

To address the occurrence of non-fireable reactions, we compute the new propensity and check whether it is zero or not. If it is zero, we must discriminate whether the reaction was the last one executed or not. In the former case, we set the reaction time to a large negative value, in order to know that this reaction's execution time will have to be generated from scratch once the reactant population becomes greater than zero, using Equation (2). In the latter case, we store the negative of the difference between the reaction time and the time at which the propensity became zero for the first time, in place of the next reaction time. That is, the $(\tau_k - t)$ portion of Equation (3). We also save the last non-zero propensity as the reaction's propensity. This mechanism allows us to manage non-fireable reactions while generating the same amount of random numbers as a complete sequential NRM implementation and reusing space.

When we compute the reaction times, we must consider the described protocol to take care of non-fireable reactions. Hence, whenever attempting to compute the next reaction time for a given reaction $\hat{r}$, we check the values stored for $\hat{r}$:

- If we are computing the reaction times after executing and $\hat{r}$ was the last executed reaction, $\hat{r}$'s reaction time needs to be recomputed as if it was just injected into the system, using a new random number as specified in Equation (2). Should $\hat{r}$ be non-fireable for lack of reactants, we instead do not compute the reaction time and store $-inf$ as its next reaction time for future reference.
- If $\hat{r}$ was not the last executed reaction and does not have enough reactants to be fired, we check the value stored as the next reaction time.
  - If it is positive, then $\hat{r}$ just stopped being fireable. We keep the last non-zero propensity and save $-(\tau_k - t)$ as its next reaction time.
  - If it is negative, then $\hat{r}$ stopped being fireable in the past, and we have already saved the relevant data and move on to dealing with the next reaction.
- If $\hat{r}$ was not the last executed reaction and has enough reactants to be fired, we check the value stored as the next reaction time.
  - If it is positive, $\hat{r}$ was fireable and continues to be. We calculate the new propensity and reaction time using Equations (1) and (3).
  - If it is negative, then $\hat{r}$ stopped being fireable in the past, and we check the stored reaction time t_s.
    - If it is $t_s = -inf$, then we compute the reaction time from scratch using Equation (2).
    - Otherwise, t_s is the inverse of the old reaction interval $t_s = -(\tau_k - t)$. We use the interval, the old propensity a_k and the new propensity $\bar{a_k}$ in Equation (3) to compute the new reaction time.

The implementations of $select\_reaction\_after\_execution$ and $select\_reaction\_after\_cancellation$ differ in that the former checks whether a reaction was the last executed one, while the latter does not need to.

IV. Experimental Evaluation

In our assessment, we have used real-world reaction networks from the literature, taken from epidermal growth factor receptor (EGFR) signalling models [33], [34].

FcεRI represents the early events of immune cells responding when they detect some kind of threat, such as allergens. It focuses on a receptor on the cell surface called FcεRI, which reacts with the proteins Lyn and Syk, and a bivalent ligand that aggregates FcεRI. The reaction network described by this model, called $fceri\_ji$, contains 354 molecular species and 3,080 reactions with non-zero rate. We considered the original FcεRI models as well as some extensions, one adding the Lyn kinase to the model, called $fceri\_fyn\_lig$, with 2,506 molecular species and 28,072 reactions with non-zero rate, and one adding the Fyn kinase to the model, called $fceri\_fyn$, with 1,281 molecular species and 12,904 reactions with non-zero rate. These extensions add more binding sites, species, and reaction rules, enlarging the considered network.

A. Testbed Configurations

We have run our experiments on a multi-processor machine equipped with 2 Intel Xeon Silver 4210R processors, each one hosting 10 cores and 20 hardware threads. The machine is equipped with 160GB of RAM, organized in two NUMA nodes, and has 8-way 640KB L1 caches, 16-way 20MB L2 caches, and 11-way 27.5 MB LLC.

For the parallel runs, we have executed the above models using the code generated according to the M2T transformations described in Section III-B, using ROOT-Sim [30]. We have varied the number of threads to analyse scalability from 2 to the maximum parallelism available on our machine, that is, 40 cores. We executed our tests using a target end-time in Virtual Time for both our and BioNetGen's simulations, set to 100 simulated seconds. The choice of 100s as end-time is a trade-off between the convergence to a steady-state (typically hundreds of seconds of virtual time), and performance accuracy and consistency across the simulators. Finally, we considered a GVT period of 100ms.

We test our implementation against BioNetGen [1]. As discussed, we implement the NRM for exact stochastic simulation, while BioNetGen implements DM. We also note that BioNetGen runs a sequential kinetic Monte Carlo simulation, hence computing the time and ID of the next reaction by sampling from the sum of reaction propensities at each time step, thus avoiding any overhead for managing reactions' and simulation's states.

To ensure fairness, both simulators were run on the same hardware, and all stochastic simulations were repeated 10 times per configuration to account for runtime variability due to system noise and stochastic event ordering, as we measured the elapsed time in seconds for both simulators.

Since our algorithm involves messages to notify LPs possibly affected by the execution of a reaction r_i, reaction partitioning has a crucial role in avoiding incurring communication overhead among LPs. We have explored several partitioning techniques to analyse if and how they affect simulation performance: *batch*, *round robin*, *randomized round robin*, *reactant clustering*, and *metis*. For each of them, we build models partitioned into 24, 100, and 1,000 LPs.

The *batch* partitioning consists of giving a set amount of contiguous reactions to each LP, possibly limiting the dependencies among reactions on different LPs, as reactions generated close in time to one another could be more likely to be interdependent. The expected advantage is to favour clustering of dependencies and reduce inter-LP communication. However, it could cause load imbalance, especially for models with few reactions.

The *round robin* approach is the simplest one, and it consists of giving one reaction to each LP in a round robin fashion, while *randomized round robin* shuffles the reactions before applying the round robin scheme. The advantage of round robin approaches is that they guarantee the fairness of the assignment and avoid load imbalance. However, it does not focus on dependencies across LPs.

The *reactant clustering* approach aimed to guarantee execution correctness by removing the possibility of "double spending" as discussed in Section III-C. The resulting partitions turned out to be highly imbalanced for all available models, making them unusable.

As for the *metis* partitioning, we used `gpmetis`, the well-known graph partitioning algorithm from the METIS suite [35], to partition the graph of reactions in order to reduce the dependencies across LPs. For this scheme, we represented the reactions as a graph $G = (V, E)$, V being the set of reactions, E being the set of edges, with u, v reactions, and $(u, v) \in E$ if reaction v is *interested* to reaction u, or vice-versa. This is because METIS works on undirected graphs. Each partition found using `gpmetis` is assigned to one simulation LP.

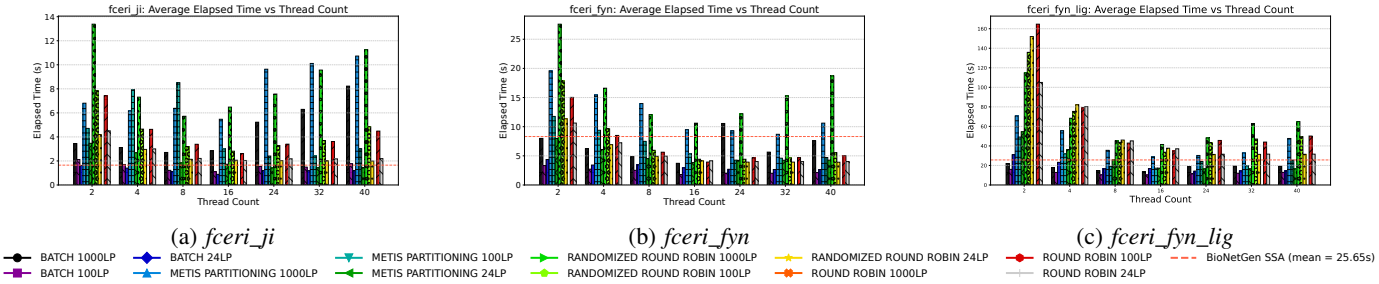
B. Results

In Figure 2, we report the elapsed time for each model and each configuration. Under the *batch* strategy, we observe that *fceri_ji*'s performance degrades with 100 and 1,000 LPs, likely due to the increased imbalance, as it is the smallest considered model. However, with 24 LPs, the same model outperforms BioNetGen, especially at high thread counts, indicating that *batch* partitioning can be effective when the number of LPs is small and the clustering of reaction dependencies is preserved. In contrast, both *fceri_fyn* and *fceri_fyn_lig* consistently outperform BioNetGen across all tested LP configurations when using *batch* partitioning, especially at higher thread counts. This suggests that for larger models, the benefits of co-locating dependent reactions outweigh the downsides of potential imbalance.

With *metis*, all models show improved performance over BioNetGen when using 24 LPs, highlighting the advantage of reducing cross-LP dependencies. While *fceri_fyn* performs similarly with 24 and 100 LPs, smaller LP counts generally yield better performance due to over-partitioning avoidance.

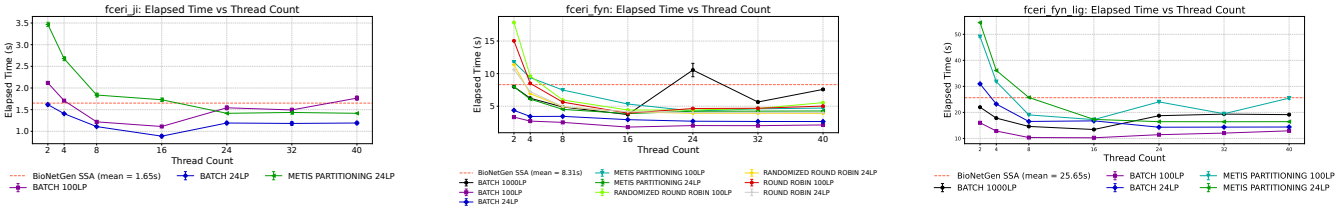
Conversely, the *round robin* and *randomized round robin* strategies fail to deliver consistent speedups. *fceri_ji* and *fceri_fyn* both underperform across most LP configurations, likely due to excessive inter-LP communication from the interleaved reaction assignment. An exception is partially given by *fceri_fyn_lig*, which sees improvements with 24 and 1,000 LPs, possibly due to better load balancing at scale. However, these benefits might be model-specific and unreliable overall. For clarity and readability, we omit the *round robin* results for *fceri_ji* with 1,000 LPs, as the results were severely impacted by the overhead of distributing 3,080 reactions across 1,000 LPs.

We also show in Figure 3 a detail on the elapsed time for a subset of the considered configurations, to better highlight the gains obtained by our SSA implementation leveraging the speculative PDES paradigm. Even for smaller models like *fceri_ji*, we observe a peak speedup of 1.85x with 16 threads using *batch* partitioning and 24 LPs. This highlights that over-partitioning such models introduces overhead, making fewer LPs preferable.



(a) *fceri_ji* (b) *fceri_fyn* (c) *fceri_fyn_lig*

Fig. 2: Elapsed time when varying thread count



(a) *fceri_ji* (b) *fceri_fyn* (c) *fceri_fyn_lig*

Fig. 3: Elapsed time of a subset of the models when varying thread count

The model *fceri_fyn* shows a more consistent performance improvement across the schemes. We reach the maximum speedup of 4.62x at 16 threads for the *batch* strategy with 100 LPs. Most configurations outperform BioNetGen from 8 threads onward (e.g. *metis* with 24 threads and 100 LPs provides 1.95x speedup). Despite the spike of *batch* with 1,000 LPs at 24 threads, we note that this strategy outperforms BioNetGen in most cases, suggesting that higher LP counts can be supported. While round robin variants offer slight gains with 24 LPs, their performance drops with more LPs due to increased dependencies, reinforcing the advantage of structured partitioning like *batch* and *metis*.

For the largest model, *fceri_fyn_lig*, *batch* and *metis* perform best, with a peak speedup of 2.5x at 16 threads for *batch* with 100 LPs. Notably, *batch* with 1,000 LPs outperforms smaller-LP configurations at higher thread counts (≥ 24), suggesting that higher LP counts can be beneficial for highly complex models, and that a more careful graph partitioning scheme might be needed.

V. Conclusions and Future Work

In this paper, we have demonstrated that a model-driven transformation of BioNetGen specifications into an actor-based intermediate representation enables efficient, parallel, and distributed stochastic simulation of chemical reaction networks. By combining the expressiveness of the Actor Model with the scalability of ROOT-Sim's speculative PDES environment, our exact SSA implementation achieves non-negligible performance improvements over traditional sequential methods while preserving stochastic accuracy. Our experimental results, carried out on well-known models from the literature, have confirmed the viability of actor-oriented model transformations as a foundation for high-performance biochemical simulation. Future work will focus on extending the framework to support hybrid simulation strategies, automatic tuning of partitioning schemes, and runtime adaptation to heterogeneous platforms.

Acknowledgements

This paper has been supported by the European Union—Next Generation EU, Mission 4, Component 2, CUP E53D23008200006 (Domain).

References

[1] M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek, "BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains," *Bioinformatics (Oxford, England)*, vol. 20, no. 17, pp. 3289–3291, Nov. 2004.

[2] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*, ser. Synthesis lectures on software engineering. Cham: Springer International Publishing, 2017.

[3] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," *International Joint Conference on Artificial Intelligence*, pp. 235–245, Aug. 1973.

[4] G. Agha and C. Hewitt, "Concurrent programming using actors: Exploiting large-scale parallelism," in *Lecture Notes in Computer Science*, ser. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 19–41.

[5] S. Bauco, G. De Angelis, R. Marotta, and A. Pellegrini, "A model-driven platform for software applications on heterogeneous computing environments," in *Proceedings of the 22nd IEEE International Conference on Software Architecture Companion*, ser. ICSA'25. Piscataway, NJ, USA: IEEE, Mar. 2025.

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

[7] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *The journal of physical chemistry. A*, vol. 104, no. 9, pp. 1876–1889, Mar. 2000.

[8] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, Jul. 1985.

[9] R. Marotta and A. Pellegrini, "Model-driven engineering for high-performance parallel discrete event simulations on heterogeneous architectures," in *Proceedings of the 2024 Winter Simulation Conference*, ser. WSC '24, H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, Eds. Piscataway, NJ, USA: IEEE, Dec. 2024, pp. 2202–2213.

[10] S. Bauco, R. Marotta, and A. Pellegrini, "DESL: A literate programming language framework for interoperable parallel discrete event simulation," in *Proceedings of the 2025 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '25. New York, NY, USA: ACM, Jun. 2025, p. 12.

[11] S. Zschaler and F. A. C. Polack, "Trustworthy agent-based simulation: the case for domain-specific modelling languages," *Software & Systems Modeling*, vol. 22, no. 2, pp. 455–470, Feb. 2023.

[12] M. E. Johnson, A. Chen, J. R. Faeder, P. Henning, I. I. Moraru, M. Meier-Schellersheim, R. F. Murphy, T. Prüstel, J. A. Theriot, and A. M. Uhrmacher, "Quantifying the roles of space and stochasticity in computer simulations for cell biology and cellular biochemistry," *Molecular biology of the cell*, vol. 32, no. 2, pp. 186–210, Jan. 2021.

[13] M. Jeschke, R. Ewald, A. Park, R. Fujimoto, and A. M. Uhrmacher, "A parallel and distributed discrete event approach for spatial cell-biological simulations," *Performance evaluation review*, vol. 35, no. 4, pp. 22–31, Mar. 2008.

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

[15] F. Xing, Y. P. Yao, Z. W. Jiang, and B. Wang, "Fine-grained parallel and distributed spatial stochastic simulation of biological reactions," *Advanced materials research*, vol. 345, pp. 104–112, Sep. 2011.

[16] Z. Lin and Y. Yao, "Parallel discrete event simulation of stochastic reaction and diffusion using reverse computation," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. IEEE, Dec. 2015.

[17] Z. Lin, C. Tropper, R. A. McDougal, M. N. I. Patoary, W. W. Lytton, Y. Yao, and M. L. Hines, "Multithreaded stochastic PDES for reactions and diffusions in neurons," *ACM Transactions on Modeling and Computer Simulation*, vol. 27, pp. 1–27, 2016.

[18] L. Dematté and T. Mazza, "On parallel stochastic simulation of diffusive systems," in *Computational Methods in Systems Biology*, ser. LNCS, M. Heiner and A. M. Uhrmacher, Eds. Berlin, Heidelberg, Germany: Springer, 2008, vol. 5307, pp. 191–210.

[19] B. Wang, Y. Yao, Y. Zhao, B. Hou, and S. Peng, "Experimental analysis of optimistic synchronization algorithms for parallel simulation of reaction-diffusion systems," in *2009 International Workshop on High Performance Computational Systems Biology*. IEEE, Oct. 2009.

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

[21] J. Lindén, P. Bauer, S. Engblom, and B. Jonsson, "Exposing interprocess information for efficient PDES of spatial stochastic systems on multicores," *ACM transactions on modeling and computer simulation: a publication of the Association for Computing Machinery*, vol. 29, no. 2, pp. 1–25, Apr. 2019.

[22] C. Dittamo and D. Cangelosi, "Optimized parallel implementation of gillespie's first reaction method on graphics processing units," in *2009 International Conference on Computer Modeling and Simulation*. IEEE, Feb. 2009.

[23] V. H. Thanh and R. Zunino, "Parallel stochastic simulation of biochemical reaction systems on multi-core processors," *Proc. of CSSim*, pp. 162–170, Sep. 2011.

[24] D. S. Tourigny, A. P. Goldberg, and J. R. Karr, "Simulating single-cell metabolism using a stochastic flux-balance analysis algorithm," *Biophysical journal*, vol. 120, no. 23, pp. 5231–5242, Dec. 2021.

[25] M. Sheldon and G. Casale, "TauSSA," *Performance evaluation review*, vol. 49, no. 4, pp. 70–75, Jun. 2022.

[26] W. Chen and E. De Schutter, "Parallel STEPS: Large scale stochastic spatial reaction-diffusion simulation with high performance computers," *Frontiers in neuroinformatics*, vol. 11, p. 13, Feb. 2017.

[27] P. Andelfinger, T. Köster, and A. M. Uhrmacher, "Zero lookahead? zero problem. the window racer algorithm," in *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. PADS '23. New York, NY, USA: ACM, Jun. 2023, pp. 1–11.

[28] A. P. Goldberg, D. R. Jefferson, J. A. P. Sekar, and J. R. Karr, "Exact parallelization of the stochastic simulation algorithm for scalable simulation of large biochemical networks," *arXiv [q-bio.MN]*, May 2020.

[29] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM*, vol. 23, no. 1, pp. 31–42, Jan. 1976.

[30] A. Pellegrini, R. Vitali, and F. Quaglia, "The ROme OpTimistic simulator: Core internals and programming model," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS. Brussels, Belgium: ICST, Apr. 2012, pp. 96–98.

[31] G. Lomow, S. R. Das, and R. M. Fujimoto, "Mechanisms for user-invoked retraction of events in time warp," *ACM transactions on modeling and computer simulation: a publication of the Association for Computing Machinery*, vol. 1, no. 3, pp. 219–243, Jul. 1991.

[32] D. R. Jefferson and P. D. Barnes, "Virtual time III: Unification of conservative and optimistic synchronization in parallel discrete event simulation," in *Proceedings of the 2017 Winter Simulation Conference*, ser. WSC '17, Wai Kin (Victor), A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, Eds. Piscataway, NY, USA: IEEE, Dec. 2017, pp. 786–797.

[33] M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek, "A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity," *Biosystems*, vol. 83, no. 2, pp. 136–151, 2006, 5th International Conference on Systems Biology. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0303264705001231

[34] B. N. Kholodenko, O. V. Demin, G. Moehren, and J. B. Hoek, "Quantification of short term signaling by the epidermal growth factor receptor*," *Journal of Biological Chemistry*, vol. 274, no. 42, pp. 30 169–30 181, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021925819518804

[35] G. Karypis and V. Kumar, "METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," Department of Computer Science and Engineering, University of Minnesota, Tech. Rep. 97-061, Nov. 1997.