Technical Report

Algorithm/Architecture Co-Design of a Stochastic Simulation System-on-Chip

Hyungman Park, Xiaohu Shen, Haris Vikalo, Andreas Gerstlauer

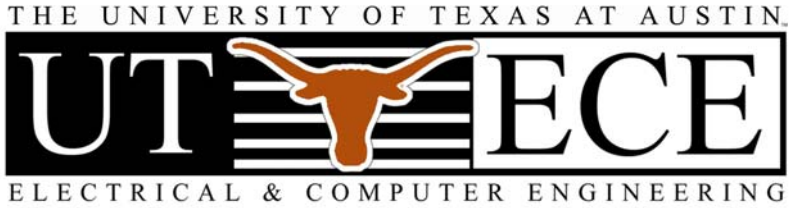
The University of Texas at Austin

UT-CERC-11-01

February 7, 2011

Computer Engineering Research Center
The University of Texas at Austin

University Station, C8800
Austin, Texas 78712-0323
Telephone: 512-471-8000
Fax: 512-471-8967
http://www.cerc.utexas.edu



Contents

List of Figures

List of Tables

Algorithm/Architecture Co-Design of a Stochastic Simulation System-on-Chip

Hyungman Park, Xiaohu Shen, Haris Vikalo, Andreas Gerstlauer
Computer Engineering Research Center
The University of Texas at Austin

Abstract

Computational models of gene regulatory networks (GRNs) well describe the behavior of interactions among molecular species over time. For larger networks, the problem of state-space explosions makes such approaches practically unsound. D. T. Gillespie discovered a statistically identical way of simulating the time evolution of species populations, leveraging a Monte Carlo technique, so-called stochastic simulation algorithm (SSA). The SSAs lend themselves accurately well to stochasticity in GRNs and other biochemical reaction systems in a well-stirred environment. The computational burdens of SSAs, however, incur immensely slow simulation run times needed to simulate a biological time of interest. In this report, we investigate various SSAs and introduce a custom yet highly scalable stochastic simulation system-on-chip (SSSoC) architecture which can achieve greater speed-ups in the simulation. With careful co-design of algorithms and microarchitectures, we compare and predict the possible SSA candidates that are well suited for hardware acceleration. Furthermore, we show how the architecture can be operated in different networking modes by exploiting coarse-grain parallelism in the algorithms. Based on our theoretical analysis, results show that our approach can achieve orders of magnitude higher performance than software simulations on a typical workstation. We believe the initial studies carried out in this report render us some guidelines toward the future research ahead of us.

1 Introduction

Gene regulatory networks (GRNs) are biological systems in which biomolecular species, such as genes, mRNA, and proteins, chemically interact with each other through such an intricate process of gene expression. DNA molecules in a gene contain all the information to code for the amino acid sequences of proteins and this information is transcribed into RNA molecules, which in turn orchestrates the underlying chemical mechanisms to translate the message sent from DNA molecules into a protein—known as the central dogma of molecular biology [1, 2]. Although what the central dogma states is simple per se, all the machineries involved in the process is not, and many biochemists have attempted to unravel the structure and dynamics of GRNs because this is the key to advancing the knowledge of the functionality of micro- and macro-organisms, to revealing mechanisms of genetic diseases, and to supporting the drug discovery process [3, 4].

Innovations in high throughput instrumentation and experimental systems—e.g., DNA and protein microarray—have aided the study of GRNs to a great extent [2]. However, practically feasible experimental studies provide relatively few data points compared to the size of the network and are adversely affected by strong biochemical and measurement noise. Due to these detrimental effects, network structures and their properties deduced from experimental results are somewhat speculative. Extensive experimental studies may potentially address this impediment, yet they are both costly and time-consuming and, for very large networks, simply not feasible at this time [5].

On the contrary, the development of computational models for GRNs and other biochemical processes have gained a lot of attention owing to its capability of predicting the network behavior without the need of extensive experimental studies [6, 7, 8, 9, 5, 10, 11]. Several computational models developed over the years include Boolean and Bayesian networks, deterministic differential equations, the chemical master equation [7], and the chemical Langevin equation [8]. Among various approaches, solving the chemical master equation using a stochastic simulation algorithm (SSA), or Monte Carlo

simulation technique, proposed by D. T. Gillespie has shown promising results with accurate characterization of the network model [11]. However, the Achilles' heel lies in its computational complexity and because of the astronomical number of iterations required to simulate biologically interesting phenomena over a reasonably long period of simulated time, even this approach easily becomes intractable with larger network systems [5].

Hence, improving the performance of the SSA is critical to understanding the structure and dynamics of GRNs and other biochemical systems. To this end, biochemists and computer scientists have introduced numerous enhanced versions of Gillespie's SSA and have built both software tools and hardware platforms to perform the simulation of varying algorithms. In this report, as part of the preliminary investigation into the design of a novel stochastic simulation system-on-chip (SSSoC), we review existing stochastic simulation algorithms and evaluate the performance of their custom hardware implementations to compare with software counterparts.

The rest of the report is organized as follows: Section 2 provides with mathematical formulations of the dynamics of GRNs and briefly reviews various versions of the SSA and their computational complexities. For better understanding of the concepts, a simple model of intracellular viral infection is illustrated with its simulation result. Section 3 reviews related work as an effort to enhance the simulation speed by means of different software and hardware platforms. Section 4 shows how our SSSoC architecture is organized and describes its core building blocks supporting different modes of operation. We describe details of several SSAs and suggest their possible hardware architectures. We also perform a theoretical analysis of latency and throughput on the microarchitectures of different algorithms and compare the results with software simulations. Section 5 describes how models with different network sizes can be partitioned and mapped onto the SSSoC architecture arranged in a network-on-chip fashion. Finally, we conclude with a summary of the report in Section 6.

2 Modeling and Simulation of Gene Regulatory Networks

The signals in gene regulatory networks are carried by molecules. For instance, proteins which enable initiation of the gene transcription to mRNA—so-called transcription factors—can be considered as input signals. They bind to the so-called promoter regions adjacent to the regulated gene and, in doing so, enable an RNA Polymerase to perform the transcription. On the other hand, proteins that are translated from the mRNA can be considered as output signals. Moreover, some of the created proteins may act as transcription factors themselves and upregulate or downregulate gene expressions, i.e., activate or suppress the transcription process. This creates feedback loops in the network allowing direct or indirect self-regulation. Therefore, it is apparent that we need some modeling and simulation methods to characterize the relationship between the input and output signals thus to accurately depict the behavior of gene expression or other kinds of biochemical systems. As an example of such systems, Figure 1(a) illustrates binding of a transcription factor to a motif. Figure 1(b) shows a small subnetwork extracted from a much larger regulatory network in yeast.

In the following sections, we show both deterministic and stochastic ways of modeling biological networks with different mathematical formulations and explain how stochasticity in gene expression lends the use of SSAs well to the simulation of GRNs.

2.1 Modeling Methods

The most common approach to the modeling of GRNs and other biochemical systems is to mathematically formulate their dynamics using a set of differential equations. Before delving into the details of such mathematical formulations, some nomenclatures must be defined. Generally, in characterizing the dynamics of a system having the N molecular species $\{S_1, \ldots, S_N\}$ that chemically interact through M specified reaction channels $\{R_1, \ldots, R_M\}$, we consider a well-stirred mixture of those N molecular species inside some volume Ω at constant temperature, and what intrigues us is the time evolution of an N-element species vector $X(t) = [x_1(t), x_2(t), \ldots, x_N(t)]'$, where $x_n(t)$ is the number of molecules for the n^{th} species S_n at time t in the system. The dynamics of the m^{th} reaction channel R_m is depicted by a stoichiometric change vector—or simply, state change vector—$\mathcal{V}_m = [\nu_{1m}, \nu_{2m}, \ldots, \nu_{Nm}]'$, where ν_{nm} denotes a change in the population of the n^{th} molecular species S_n as a result of an occurrence of reaction R_m. In addition, given the system state at time t denoted as $X(t) = X$, the probability that a reaction will occur somewhere inside Ω in the next infinitesimal time interval $[t, t + dt)$ is defined as $a_m(X)dt$, where $a_m(X)$ is called the propensity function of reaction

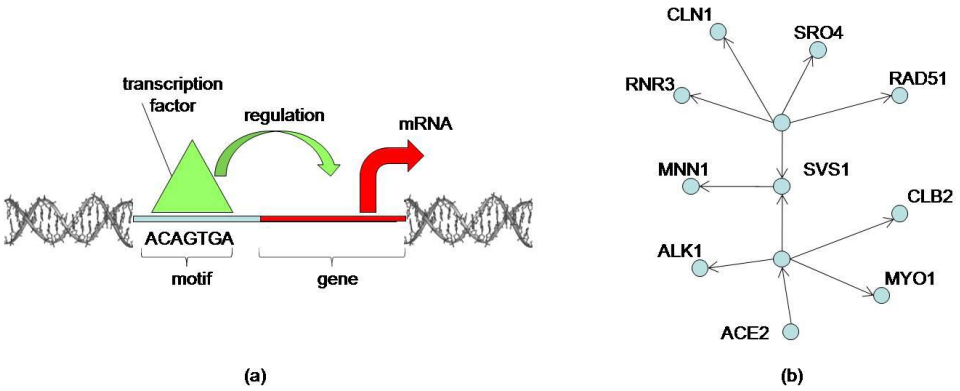


Figure 1: *Gene expression with (a) transcription factor binding to a regulatory region and thus regulating transcription of the gene downstream, and (b) subnetwork extracted from a much larger regulatory network in yeast.*

R_m. The propensity function can further be expressed as $a_m(X) = c_m h_m(X)$, where c_m is a stochastic rate constant implying the probability that one reaction takes place in the time interval $[t, t+dt)$, and $h_m(X)$ denotes all possible combinations of individual R_m reactant molecules at instance t. Note that $h_m(X)$ is typically expressed in three different forms as a complex reaction comprising more than two reactant species can be further decomposed into a number of elementary reactions [1] [12]. With these notations defined, we will now review how the system behavior can be modeled using different mathematical formulations.

2.1.1 Chemical Master Equation

As the probability of a reaction occurring in the next infinitesimal time interval $[t, t+dt)$ depends only upon the state $X(t)$ at time t, i.e., the future state depends only upon the present state, we can model $X(t)$ as a Markov process with discrete states, where the time evolution of the state probabilities $P(X,t)$ is given by the chemical master equation (CME) [7],

$$\frac{\partial P(X,t)}{\partial t} = \sum_{m=1}^{M} [a_m(X-\nu_m)P(X-\nu_m,t) - a_m(X)P(X,t)]. \quad (1)$$

As shown in Equation (1), the CME is a stochastic model in the form of a linear ordinary differential equation (ODE) that exactly describes the probability of a system being in a particular state X at time t. Although the CME models the network behavior in an exact manner, the biggest challenge comes when one wishes to solve this equation computationally. However, this approach becomes inviable as we may easily run into the problem of state-space explosions. In other words, the number of states for a system consisting of N species with a population size of n per species is given by n^N.

[1] The three elementary reactions include a monomolecular reaction (type-1; $S_1 \rightarrow S_2$), a bimolecular reaction with reactant species of different kinds (type-2a; $S_1 + S_2 \rightarrow 2S_3$), and a biomolecular reaction with reactant species of the same kind (type-2b; $2S_1 \rightarrow S_2$). Thus, h_m can be written in the following forms: $h_m = x_1$ for type-1, $h_m = x_1 x_2$ for type-2a, and $h_m = x_1(x_1-1)/2$ for type-2b, where x_1, x_2, and x_3 represent the number of molecules for species S_1, S_2, and S_3, respectively.

2.1.2 Chemical Langevin Equation

Given $X(t) = X$ denoted as the current system state at time t, let a random variable $\mathcal{K}_m(X,\tau)$ for any $\tau > 0$, be the number of reactions that occur in the next time interval $[t, t+\tau]$. Then, the future system state at time $t+\tau$ is given by

$$X(t+\tau) = X(t) + \sum_{m=1}^{M} \mathcal{V}_m \mathcal{K}_m(X,\tau). \quad (2)$$

Notice Equation (2) is in the form of a stochastic differential equation (SDE), and it is not a trivial task to obtain a probability distribution function for $\mathcal{K}_m$. However, imposing certain conditions on the equation above, we can achieve a good approximation to the random variable $\mathcal{K}_m$.

Condition 1: τ must be small enough such that none of the propensity functions in the system changes its value appreciably during the time interval $[t, t+\tau]$, i.e.,

the propensity functions satisfy

$$\begin{aligned} &a_m(X(t')) \cong a_m((X(t)) \\ &\forall t' \in [t, t+\tau] \quad \text{and} \quad \forall m \in [1, M]. \end{aligned} \tag{3}$$

If condition 1 is satisfied, then all reactions occurring in the time interval will be independent and each $\mathcal{K}_m$ will be a statistically independent Poisson random variable $\mathcal{P}_m(a_m(X(t)), \tau)$ resulting in

$$X(t+\tau) = X(t) + \sum_{m=1}^{M} \mathcal{V}_m \mathcal{P}_m(a_m(X(t)), \tau). \tag{4}$$

Condition 2: τ must be large enough such that the expected number of occurrences for reaction R_m during the time interval $[t, t+\tau]$ is much larger than 1, i.e.,

$$\begin{aligned} &\langle \mathcal{P}_m(a_m(x), \tau) \rangle \\ &\quad = a_m(X(t))\tau \gg 1 \quad \forall m \in [1, M]. \end{aligned} \tag{5}$$

Although Condition 2 is counter to Condition 1 and it may look rare to meet both conditions simultaneously, Gillespie stated in [8] that sufficiently large molecular populations are likely to suffice Equations (3) and (5), simultaneously. The value of τ satisfying both conditions is considered as a macroscopic infinitesimal dt, and Equation (4) can be further approximated by a nonlinear stochastic differential equation, so-called the chemical Langevin equation (CLE) [8],

$$\begin{aligned} X(t+\tau) = X(t) &+ \sum_{m=1}^{M} \mathcal{V}_m a_m(X(t))dt \\ &+ \sum_{m=1}^{M} \mathcal{V}_m \sqrt{a_m(X(t))dt}\mathcal{N}_m(t), \end{aligned} \tag{6}$$

where $\mathcal{N}_m(t)$, $\forall m \in [1, M]$, are independent standard Gaussian random variables with a zero mean and a unit variance. Notice that Equation (6) is the canonical form of the standard Langevin equation [13], and fulfilling both conditions makes the problem change from solving a discrete-state Markov process in the CME to solving a continuous-state Markov process.

2.1.3 Reaction Rate Equation

A biochemical system can also be deterministically modeled using a set of ordinary differential equations, so-called reaction rate equations (RREs), based on the law of mass action. The RRE can be written in general form as

$$\frac{dY(t)}{dt} = \sum_{m=1}^{M} \nu_m \tilde{a}_m(Y(t)), \tag{7}$$

where $Y(t) = X(t)/\Omega$ and $\tilde{a}_m = a_m/\Omega$. Once the initial conditions and rate constants of a given system are known, the future states of the system can be predicted deterministically by solving equations in the form above.

Interestingly, it can be easily observed that Equation (7) is derived from the CLE under the assumption of a thermodynamic limit, in which both the number of molecules in the system and the system volume Ω approach ∞ while maintaining species concentrations. This is attributed to the fact that, in such a condition, the second term in Equation (6) vanishes as it becomes dominated by the first term. Therefore, all of the three modeling approaches explained so far are interconnected each other such that the CME is approximated by the CLE and the RRE is another form of the CLE in the thermodynamic limit [8].

Although this deterministic approach models well for such systems with large populations of species, it is not sufficiently accurate for capturing the dynamics of the system with a small number of molecules for certain species on the order of 10 to 100 due to the tendency to show stochasticity in its behavior [14]. Moreover, the complexity of solving the equations grows vastly with increases in the number of reactions and species in the system.

2.2 Stochastic Simulation Algorithms (SSAs)

To accurately describe the dynamics of biochemical systems, it is important to employ the right model among different approaches described above. Molecular interactions in gene regulatory networks are subject to significant spontaneous fluctuations. For example, to allow binding of an RNA Polymerase to a promoter region, certain enzymes act as a catalyst and set the promoter into an active state. Thermal fluctuations in the cell cause promoters to randomly switch between an active and a repressed state, effectively making the transcription a random event. As a result, the number of created proteins is a random variable. The fluctuations in the number of proteins are enhanced by the protein degradation, which is a stochastic process itself. Moreover, mRNA may also be degraded, which results in variations of the mRNA available for translation. Finally, binding of the previously mentioned transcription factors to the promoter regions, needed to

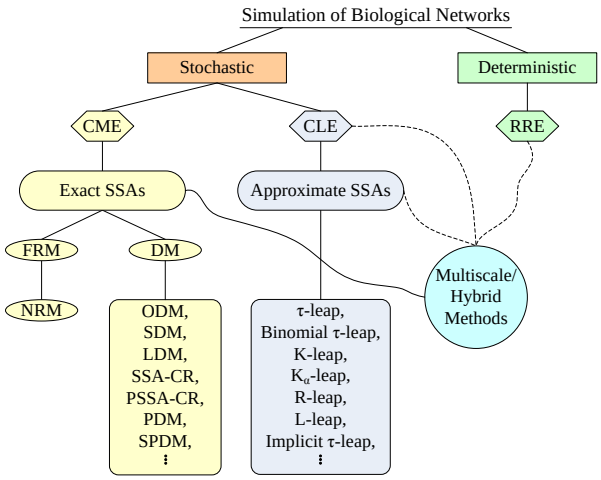


Figure 2: *Taxonomy tree of various methods.*

initiate RNA Polymerase, are also probabilistic events. Therefore, to fully capture the dynamics of molecular interactions in GRNs, we need a stochastic network model thus the CME or CLE is often used to reflect stochasticity inherent in gene regulatory networks.

As discussed earlier, the CME precisely models the system state $X(t)$ as a Markov process with discrete states and provides with the time evolution of the state probabilities. By contrary, the CLE approximately models the time evolution of the system state $X(t)$ as a Markov process with continuous states. Formulations of the CME or CLE for most biochemical systems with larger network sizes, however, result in an impractically large set of ordinary or stochastic differential equations. Consequently, both CME and CLE becomes intractable to be solved in conventional ways. To overcome such high computational complexity, Gillespie pioneered the way toward efficiently solving both types of equations, in either an exact or approximate manner, by leveraging a Monte Carlo method, so-called stochastic simulation algorithm. Thus far, numerous variants of the Gillespie's original algorithms have been introduced, and we will now briefly review some of the popular SSAs by classifying them into exact, approximate, and hybrid methods as shown in the taxonomy tree[2] of Figure 2.

Gillespie originally introduced two different SSAs called the direct method (DM) [11] and the first reaction method (FRM) [15]. Because they assume the same probability model on which the CME is based, both the DM and the FRM are regarded as the exact SSA. In such algorithms, reactions are evaluated in a continuous, stepwise fashion to execute the one most likely to occur next. Since they simulate individual reactions over time, exact SSAs are accurate but computationally very intensive. Both the DM and the FRM have an algorithmic complexity of $O(EM)$, which is linear in the size of the network (number of reactions M) and the number of simulated events (number of executed reaction cycles and time steps E). However, in regular sequential implementations, the DM is typically more efficient. In each time step, the DM randomly generates the time τ until the next reaction and the channel μ where it takes place. By contrast, the FRM has a smaller fixed cost but needs to generate M exponentially distributed random numbers to determine τ and μ as the minimum over individual times τ_m when reaction m will occur next. In both cases, all M propensity functions $a_m(X(t))$ need to be evaluated in every time step.

To address the problem of high computational complexity, implementations of the exact DM and FRM with optimized data structures for efficient data reuse and caching have been developed. In the optimized direct method (ODM) [16], the sorting direct method [16], and the next reaction method (NRM) [12], τ_m are only computed for reactions in which any of the input species concentrations has changed, leading to a complexity of $O(ED \log M)$ (where D is the average number of updates per time step).

On the other hand, Gillespie also presented an approximate SSA called the τ-leap method [17, 18] assuming all M reactions fulfill the so-called leap condition (Condition 1 in Section 2.1.2) [3]. In the τ-leap method, a number of time steps are leaped over by amount τ and, within that τ period of time, all M reactions in the system are executed a $\mathcal{K}_m$ number of times, where $\mathcal{K}_m$ is a Poisson random variable of reaction R_m with an expected value of $a_m\tau$, whereas exact methods advance only a single time step at a time by executing a single, selected reaction that is most probable to occur next. As a result, despite the fact that the procedure of evaluating τ is much compute intensive than that of exact methods, as long as the leap condition for a network allows a large enough number of events to be aggregated into a single

[2]Note that the list of various SSAs in this taxonomy tree is not meant to be complete and there exists a plenty of more algorithms accounting for the amount of efforts put into overcoming the complexity.

[3] It is also shown in the Gillespie's original τ-leap paper [17] that, as species populations become large enough to meet all M conditions given by Condition 2 in Section 2.1.2, in addition to meeting the leap condition, executing the τ-leap method is equivalent to solving the chemical Langevin equation.

time step, the fewer time steps in leaping methods can significantly improve performance with acceptable losses in accuracy. Additionally, many improved τ-leap methods have been introduced to avoid negative populations of reactant species [19, 20, 21] and to prevent large changes in propensity functions [21, 22].

Another kind of approximate methods is the multiscale or hybrid method. Such methods enhance the inefficiencies of both exact methods and variants of the τ-leap method by conforming to the multiscale nature of gene expression and other chemically reacting systems. Because species populations and reaction rates vary dynamically both over time and among different reactions in the network, some reactions are slow while others are fast. In a system with the majority of fast reactions, exact methods become extremely inefficient and may not well represent the system whose critical behavior is governed by slow reactions. Likewise, when a system comprises mostly slow reactions, τ-leap methods are likely to have small sizes of time steps akin to those of exact methods thus become computationally very inefficient with losses in accuracy. To address such inconsistencies, hybrid methods classify reactions, either dynamically or statically, into slow and fast reactions and execute, generally, an exact method on slow reactions and an approximate method on fast reactions. Many approaches have been proposed by varying the approximate methods applied to fast reactions, i.e., using τ-leap methods [23, 24] or solving the CLE or RRE [25, 26, 24]. Additionally, other methods simulate only slow reactions by incorporating the effect of fast reactions into slow reactions by making a quasi-steady state assumption [25, 27, 28, 29, 30, 31, 32].

2.3 Example: Intracellular Viral Infection

To illustrate the concepts, we consider a simple model of intracellular viral infection [33, 28, 34, 35] with 4 molecular components and 6 reactions,

$$R_1 : \text{RNA} \xrightarrow{c_1} \text{DNA} \qquad R_2 : \text{DNA} \xrightarrow{c_2} \text{DNA + RNA}$$
$$R_3 : \text{RNA} \xrightarrow{c_3} \text{RNA + P} \qquad R_4 : \text{RNA} \xrightarrow{c_4} 0$$
$$R_5 : \text{P} \xrightarrow{c_5} 0 \qquad R_6 : \text{DNA + P} \xrightarrow{c_6} V.$$

The above reactions describe insertion of the viral sequence into the host DNA (R_1), transcription of the viral DNA to viral RNA (R_2), translation of the viral RNA to the protein P (R_3), degradation of the RNA and protein (R_4 and R_5, respectively), and the creation of the new viral structure which leaves the host cell (R_6). The stochastic rates of the reactions in the channels R_1 to R_6 are $c_1 = 1/\text{day}$, $c_2 = 0.025/\text{day}$, $c_3 = 100/\text{day}$, $c_4 = 1/\text{day}$, $c_5 = 1.99/\text{day}$, and $c_6 = 11.25 \times 10^{-6}/\text{day}$ [35, 33]. Let $x_1(t)$, $x_2(t)$, $x_3(t)$, and $x_4(t)$ denote the number of molecules of the viral DNA, RNA, protein, and viral structure, respectively. The vectors describing the changes in $x_i(t)$, $1 \leq i \leq 4$ can be stated by inspection of the reaction rules,

$$\mathcal{V}_1 = [\ 1 \quad -1 \quad 0 \quad 0\ ]' \quad \mathcal{V}_2 = [\ 0 \quad 1 \quad 0 \quad 0\ ]'$$
$$\mathcal{V}_3 = [\ 0 \quad 0 \quad 1 \quad 0\ ]' \quad \mathcal{V}_4 = [\ 0 \quad -1 \quad 0 \quad 0\ ]'$$
$$\mathcal{V}_5 = [\ 0 \quad 0 \quad -1 \quad 0\ ]' \quad \mathcal{V}_6 = [\ -1 \quad 0 \quad -1 \quad 1\ ]'.$$

We simulated the reactions with an ODM and τ-leap method using the StochKit2 [36] software package. The initial condition was chosen as $X(0) = [700 \quad 10 \quad 200 \quad 0]'$. Figure 3 shows the variations in the number of the protein molecules $x_3(t)$ over a period of 30 days. The graph plots both a single realization of the Markov process $x_3(t)$ and an average value over 1000 runs of the ODM and τ-leap SSAs. For this simple network, simulation of 1000 runs on a 2.4 GHz Intel Core2 Quad workstation required a total of 4.4s and 1.8s, respectively. For larger problems, i.e., realistic networks with a large number of species, reactions, time steps and simulation runs, run times on regular workstations quickly become prohibitive.

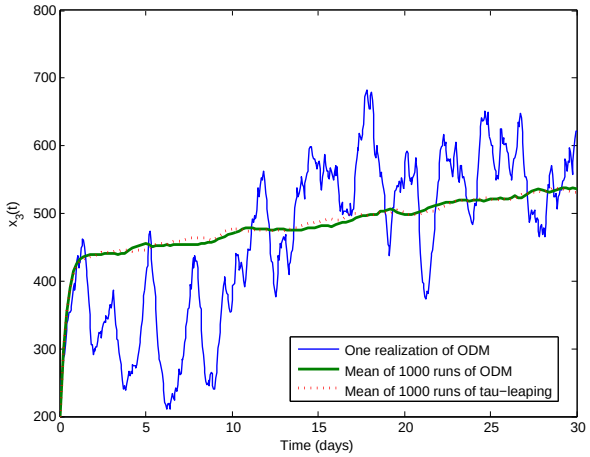


Figure 3: *Simple viral network simulation.*

3 Related Work

The SSAs are traditionally implemented on general purpose workstations. Such realizations tend to be slow and become prohibitive for all but the simplest networks with very few components and time steps to be simulated. For example, simulating expression of

one gene in one generation of E. coli (with 30 min. simulated real time between cell divisions) can take more than 20 h [37]. Simulation of the whole cell, which encompasses more than 10^{14} events [38], requires 30 years even on modern GHz-class workstations [39]. SSAs exhibit potentially massive parallelism, both fine-grain across concurrent evaluations of reaction channels in each time step, and coarse-grain across multiple instances of the SSA system. To exploit this, parallel SSA implementations on supercomputers [40, 41], compute clusters [42, 43] and emerging many-core and GPU platforms [44, 45] have been investigated. These approaches all exploit parallelism across multiple independent simulations, but do not reduce prohibitively long simulation times for a single instance of a large-scale system. In such cases, the simulation algorithm itself can be parallelized by partitioning the system model and distributing reaction computations across a networked cluster [46, 47, 48].

Previous approaches for custom hardware realizations of DM [49], FRM [50] or NRM [51] SSAs on FPGAs have shown promising results. However, the flexibility provided by reconfigurable hardware fabrics limits their size and performance. Furthermore, FPGAs typically require difficult and time-consuming redesign processes for each new problem instance, which involves complex synthesis tools that are not intuitive to the intended users in the natural sciences. While some approaches allow for reconfiguration without the need to resynthesize [51], they are limited to a particular SSA algorithm and impose tight restrictions on parameters such as network size.

4 SSSoC Architecture

From this section onward, we introduce the design of our SSSoC architecture and its core building components, and compare the performance of different SSAs in the form of custom hardware implementations. We envision SSSoC architecture will act as a general platform capable of simulating biochemical networks using various SSAs discussed in the previous sections. As illustrated in Figure 4, SSSoC is realized as a scalable array of Stochastic Processing Elements (SPEs) exploiting both coarse- and fine-grain SSA parallelism across and within SPEs, respectively. Each SPE contains an exact and an approximate execution engine to allow for simulation of fine-grain reactions, species updates and associated time steps following either an exact, a leaping or a hybrid scheme. SPEs can be arranged in an on-chip or off-chip network for simultaneous simulation of multiple independent network instances or simulation runs. SPEs can optionally exchange species updates with their neighbors for partitioned simulation of larger networks or co-simulation of several interacting networks, such as tissue cells interacting through diffusion.

The computation in SPEs will be driven by a hierarchical combination of local and global control that determines the reactions and species updates to be evaluated in each reaction cycle and time step. SSSoCs will be predominantly hardcoded but will be able to execute arbitrary network descriptions that are pre-loaded into SPE-internal reaction tables and species memories. As a result, programming of SSSoC will be straightforward with the help of simple tools that can read network descriptions in standard formats, such as the Systems Biology Markup Language (SBML) [52], and download them into the SPEs.

Stochastic Processing Elements (SPEs) perform computation of the time series of species concentrations and reactions in a given GRN model following a specific simulation algorithm. At their core, SPEs contain two execution units that let them operate in either an exact mode, a τ-leap mode, or a hybrid combination of the two. Exact and approximate units share a common species memory, control and router. A central controller holds tables with reactions assigned to the SPE and distributes and orchestrates computation in coordination with local state machines in each execution unit. As discussed, reaction channels can always be broken down into elementary reactions with not more than two reacting species [12] and hence only three possible input combination functions $h_1(x_{mi})$, $h_2(x_{mi}, x_{mi})$, and $h_3(x_{mi}, x_{mj})$. Without loss of generality, it is therefore sufficient to store for each reaction the index m, the indices mi and mj of participating species and the coefficient c_m in a central reaction memory of fixed width. Likewise, a vector table stores change vectors $\mathcal{V}_m$ encoded as quadruples $\mathcal{V}_m = [(i, v_i)\ \ (j, v_j)\ \ (k, v_k)\ \ (l, v_l)]$. In every reaction cycle, each execution unit then computes the τ and $\mathcal{V}$ across all assigned reactions. As will be described below, execution units are internally pipelined and R-way parallelized to fetch concentrations from the species memory and compute reaction times at a maximum rate of R x_{mi}/x_{mj} pairs per clock cycle. One of the most crucial challenges to be addressed will be the memory bandwidth needed to keep execution units supplied with input data. Given complex tradeoffs, design parameters, such as the banking of memories, will be carefully balanced using thorough, and potentially

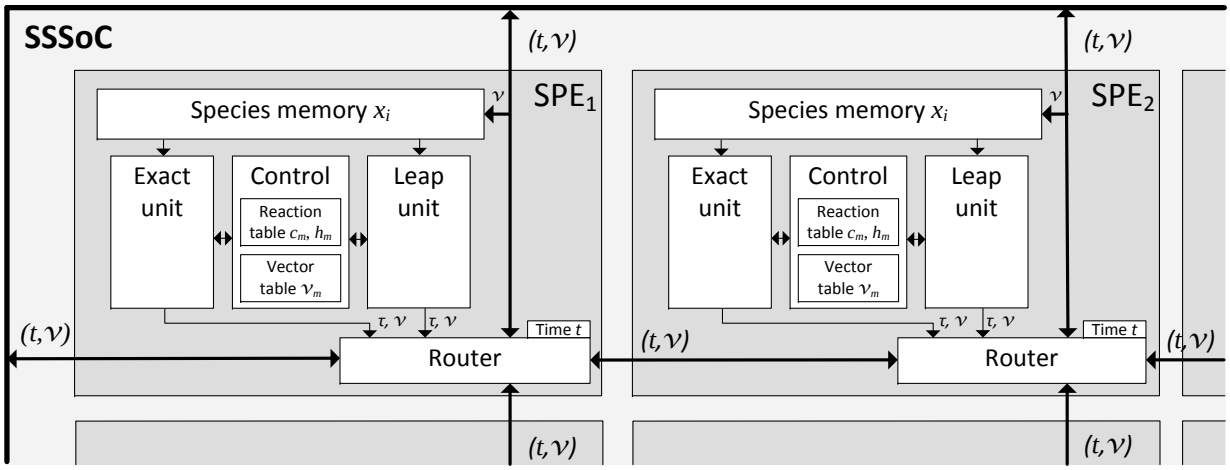


Figure 4: *Stochastic Simulation System-on-Chip (SSSoC) architecture.*

automated [53, 54, 55], exploration of the design space for each considered SSA choice.

4.1 Exact SSAs and Architectures

In this section, we elaborate on various exact SSAs and design their corresponding hardware architectures in block diagram form. We also compare the performance of different architectures by estimating the total latency needed to simulate every Monte Carlo iteration of the SSA.

4.1.1 The First Reaction Method (FRM)

The first reaction method [15] computes a tentative time τ_m of reaction R_m at time instance t as given by

$$\tau_m = -\frac{\ln r}{a_m(X(t))} = \frac{-\ln r}{c_m h_m(X(t))}, \tag{8}$$

where r is a uniformly distributed random number in the interval $(0, 1)$. Then, $t + \tau$ is the time at which the next reaction is likely to occur, where t is the current time and τ is given by the minimum value of all τ_m for M reactions, i.e.,

$$\tau = \min\{\tau_1, \ldots, \tau_M\}. \tag{9}$$

The reaction R_μ, which is determined as the one most likely to occur in the next infinitesimal time interval $(t + \tau, t + \tau + d\tau)$, is obtained by taking the index μ corresponding to τ, i.e.,

$$\mu = \text{argmin}_m\{\tau_1, \ldots, \tau_M\}. \tag{10}$$

The FRM algorithm is explained below:

1. **Initialization.** *Initialize the number of molecules with $X(0)$ and set the current time to $t \leftarrow 0$.*

2. **Propensity functions.** *Calculate propensities $a_m(X(t))$ for all M reactions.*

3. **Reaction times.** *Generate M independent, uniformly distributed random numbers r between 0 and 1; and calculate the tentative reaction times τ_m for all M reactions by Equation (8).*

4. **Reaction selection.** *Find τ and μ by Equation (9) and (10)*

5. **Reaction execution.** *Update the current time and the number of molecules by $t \leftarrow t + \tau$ and $X(t) \leftarrow X(t) + \mathcal{V}_\mu$, respectively, where $\mathcal{V}_\mu$ is a stoichiometric vector for the selected reaction.*

6. **Termination.** *If $t < t_{desired}$ or no more reactant species remain in the system, terminate the simulation; otherwise go to step 2.*

A software implementation of the FRM becomes inefficient in proportion to the size of the network since we need to generate M tentative reaction times as well as M exponentially distributed random numbers for every iteration of the algorithm. In hardware, however, computation of -ln r and a_m can be performed in parallel. Hence, the overhead for generating one random number per reaction can be effectively amortized. In addition, hardware replication for computing the reaction times can further enhance throughput of the algorithm.

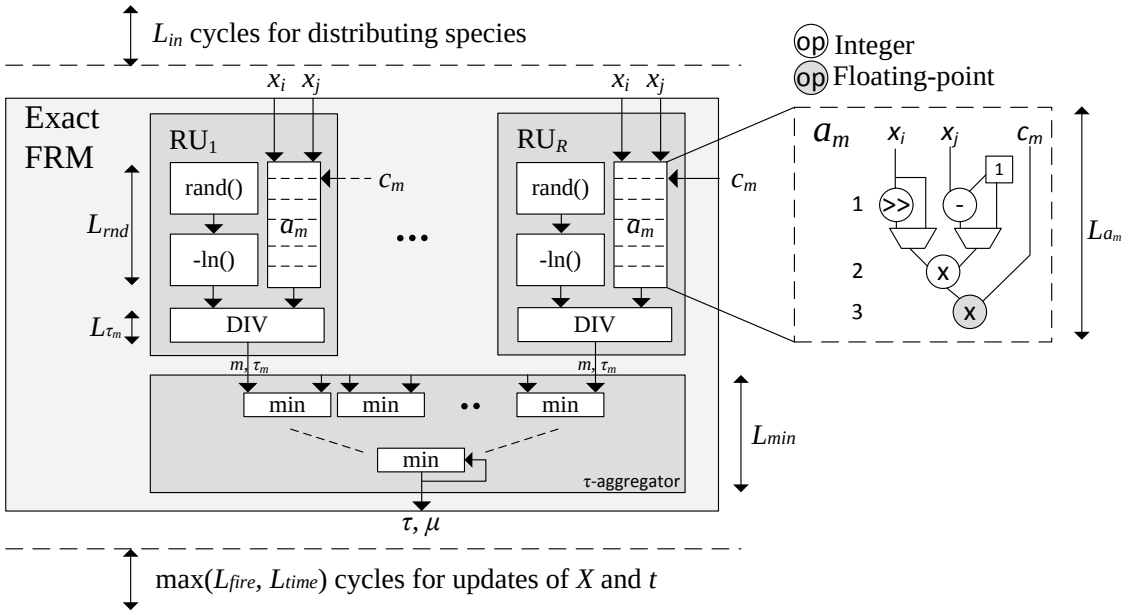


Figure 5: *Microarchitecture of FRM unit.*

By exploiting such parallelism, a hardware unit for the FRM can be designed as shown in Figure 5. It adopts a deeply pipelined architecture consisting of multiple reaction units $\{\mathrm{RU}_1 \ldots \mathrm{RU}_R\}$ for computing the tentative times of different reactions and a τ-aggregator unit for determining both the next occurrence time τ and its associated reaction index μ. Inside each RU, a random number generator rand() and a logarithmic function generator -ln() are sequentially connected, and in parallel to those two components, a propensity function generator a_m computes propensities using the rate constant c_m and the number of molecules x_i and x_j as input. As discussed in Section 2.1, propensities are calculated only for elementary reaction types so its implementation can be simplified to a mix of arithmetic operators and multiplexers as shown in Figure 5 on the right. The τ-aggregator unit is structured as a binary tree comparing R sets of data (m, τ_m) coming from RUs to compute τ and μ.

The data that flows through the FRM unit can be managed by the external control unit (not shown in Figure 5). The control unit continuously fetches data from external memories and distributes them across different RUs. Meanwhile, the propensities computed by RUs are passed continuously into the tree-structured τ-aggregator to be compared among each other. After a certain number of cycles, both τ and μ are available at the output, and the control unit in turn loads a stoichiometric vector addressed by the index μ and finally pass it, together with τ, to the external memory unit for updates via the external router.

Table 1: *Latency cycles of various operators.*

Operations	# of cycles	Float	References
rand()	4	Yes	[50]
-ln(), e^x	12	Yes	[56]
add/subtract	3	Yes	[57, 58, 59, 60, 61]
	1	No	[61]
multiply	3	Yes	[57, 58, 60, 61]
	2	No	[61]
divide	20	Yes	[57, 58, 62, 60, 61]
shift	1	Yes, No	[61]
compare	1	Yes	[63, 58, 60, 61]
min	1	Yes	[63, 60, 61]
absolute	1	Yes	[61]

Exploiting parallelism both across multiple RUs and within a single RU, we can potentially achieve some speed-ups over software realizations. For a quantitative study, we perform a theoretical analysis to evaluate the latency and throughput of the FRM unit. As to estimating the latency of every building block in the design, we refer to various literature sources and meticulously selected latency numbers practically applicable to the actual implementation. Latency numbers are labeled next to each building component in Figure 5 and are also summarized in Table 1 with

relevant references. We will be using the same latency numbers listed in this table throughout the report for consistent analysis.

The total number of cycles needed for each time step can be evaluated by summing up all the latency numbers along the critical path. As such, the overall latency can be expressed as

$$L_{frm} = L_{in} + \max(L_{rnd}, L_{a_m}) + L_{\tau_m} + L_{min} + \max(L_{fire}, L_{time}) \quad (11)$$

where L_{in} is the latency for distributing R sets of input data (c_m, x_i, x_j) into RUs; L_{rnd} for generating an exponentially distributed random number; L_{a_m} for computing a propensity; L_{τ_m} for computing a tentative time; L_{min} for determining τ and μ; L_{fire} for firing the selected reaction and updating the species; and L_{time} for updating the simulated time by adding τ to the current time.

Using the banking of memories for storing the input data of RUs, we assume the latency for species distribution can be reduced to

$$L_{in} = \lceil \log R \rceil. \quad (12)$$

As the tree height of τ-aggregator is $\lceil \log R \rceil$ and a total of M reactions are to be processed, the number of cycles required for computing τ and μ is expressed as

$$L_{min} = \lceil \log R \rceil + \left\lceil \frac{M}{R} \right\rceil. \quad (13)$$

Substituting Equations (12) and (13) into Equation (11), and using the fixed values of $L_{rnd} = 36$, $L_{a_m} = 6$, $L_{\tau_m} = 20$, $L_{fire} = 3$, and $L_{time} = 4$ [4], the overall latency of the FRM turns into

$$L_{frm} = 2\lceil \log R \rceil + \left\lceil \frac{M}{R} \right\rceil + 40. \quad (14)$$

4.1.2 The Next Reaction Method (NRM)

One variant of the FRM is the next reaction method [12], which scales well specifically for a loosely-coupled system with larger network sizes. The main idea of this method is threefold: first, using a dependency graph depicting interactions among all reactions in the network, it computes tentative reaction times only for those reactions affected by the previous reaction occurred; second, it uses absolute time rather than relative time for time between reactions, making it possible to reuse the previous tentative reaction times for unaffected reactions without generating new random numbers; lastly, it employs a data structure of indexed priority queue (IPQ) to reduce the time needed for finding the minimum of tentative reaction times.

[4] For the time update, we assume 3 cycles for the floating-point addition of $t \leftarrow t + \tau$ and 1 additional cycle for writing the new t to the corresponding register, which together make the latency a total of 4 cycles. For the species update, assuming 1 cycle for loading a stoichiometric vector, 1 cycle for adding or subtracting the number of molecules, and 1 cycle for storing the updated species data to the memory, a latency of 3 cycles in total is needed. Thus, we can implement the two updates in 4 cycles by operating them concurrently.

Table 2: *Simple dependency graph.*

Index	Reaction	Depends on	Affects	Computes
R_1	$A + B \rightarrow C$	A, B	A, B, C	R_1, R_3
R_2	$D + E \rightarrow E + F$	D, E	D, F	R_2
R_3	$C \rightarrow D$	C	C, D	R_2, R_3, R_4
R_4	$D \rightarrow \emptyset$	D	D	R_2, R_4

Computing propensities sequentially for all reactions at every iteration of the algorithm is costly. In the NRM, this is amortized by creating a dependency graph before initiating a new simulation and by calculating propensities for affected reactions only. Table 2 shows an example of the dependency graph for a simple network with 5 species and 4 reactions. As can be seen, we only need to compute propensities for reactions containing one or more reactant species whose number of molecules have changed by the previously executed reaction. Suppose R_1 was the last reaction executed. Then, the number of molecules for species A, B, and C must have been changed by that reaction, thus reactions R_1 and R_3 need to be recomputed as they contain at least one of the aforementioned species as reactant species. Therefore, only a portion of the entire reactions is considered for the computation of propensities, especially in the loosely-coupled cases.

Evaluating tentative reaction times in the NRM is somewhat distinct from the FRM. For the last reaction executed, its tentative reaction time is computed by the same equation

$$\tau_{\mu,new} = -\frac{\ln r}{a_{\mu,new}(X(t))} = \frac{-\ln r}{c_\mu h_{\mu,new}(X(t))} \quad (15)$$

as in the FRM, and thus a random number is needed to be generated. However, for reactions affected by the previous execution, $\tau_{m,new}$ is computed by scaling the previous $\tau_{m,old}$ with the ratio of old and new propensities, i.e.,

$$\tau_{m,new} = \frac{a_{m,old}}{a_{m,new}}(\tau_{m,old} - t) + t \quad (m \neq \mu), \quad (16)$$

where $a_{m,new}$ is a newly computed propensity for reaction R_m; $a_{m,old}$ and $\tau_{m,old}$ are, respectively, a

propensity and a tentative reaction time obtained from the previous iteration; and t is the current time. A caution needs to be taken for the notion of time in the equation. Unlike the FRM, all the time variables are regarded as absolute time, not relative time. In this respect, the current time t is, in fact, equal to $\tau_{\mu,old}$, which is the time at which the last reaction executed. As a result, by using absolute time, it is mathematically proven in [12] that only a single random number generation per iteration is necessary, and reuses of τ_m values for the reactions not affected by the prior execution are possible.

The NRM further enhances its performance by using the IPQ data structure together with an index data structure when searching for the minimum τ. All τ_m values are maintained in a binary-tree data structure such that it takes $O(\log M)$ operations to update a new τ_m and a $O(1)$ operation to take the minimum τ_μ. Details of how to manage the data structures are beyond the scope of this report (see the NRM paper [12]).

The NRM algorithm is explained below:

1. **Initialization.** *Initialize the number of molecules with $X(0)$ and set the current time to $t \leftarrow 0$; generate a dependency graph; and execute the FRM algorithm for one iteration and maintain tentative reaction times in an IPQ.*

2. **Propensity functions.** *Calculate only affected propensities according to the dependency graph.*

3. **Reaction times.** *Generate a uniformly distributed random numbers between 0 and 1; calculate a tentative reaction time for the last reaction executed using Equation (15); calculate tentative reaction times for affected reactions using Equation (16); and update the IPQ, accordingly.*

4. **Reaction selection.** *Get the minimum τ and its corresponding reaction index μ from the IPQ.*

5. **Reaction execution.** *Update the current time and the number of molecules by $t \leftarrow \tau$ and $X(t) \leftarrow X(t) + \mathcal{V}_\mu$, respectively, where $\mathcal{V}_\mu$ is a stoichiometric vector for the selected reaction.*

6. **Termination.** *If $t < t_{desired}$ or no more reactant species remain in the system, terminate the simulation; otherwise go to step 2.*

The core unit of the NRM is shown in Figure 6. Since the NRM is derived from the FRM, it shares the basic framework of the FRM unit but additionally incorporates, in each RU, a τ_m generator, a propensity memory (denoted as a_m Mem), and a τ_m tree implementing the IPQ data structure.

Based on the dependency graph stored in a table, the external control unit (not shown in Figure 6) distributes R sets of input data (c_m, x_i, x_j) across RUs, only for those reactions whose propensities need to be recomputed by the propensity generator (denoted as a_m). Once a valid propensity comes off the pipeline of the propensity generator, it is stored into the propensity memory for reuse in the next time step, and at the same time, it is driven into the two data paths generating two different τ_ms given by Equations (15) and (16). Upon determining whether τ_m being generated is associated with the last executed reaction, the control unit selects a proper τ_m from the right data path—either $\tau_{\mu,new}$ or $\tau_{m,new}$. The tree is then traversed and the nodes are updated with the computed values τ_m. Lastly, as all τ_ms for affected reactions have been evaluated by RUs, the minimum value read from the root node of the tree in each RU acts as input to the tree-structured τ-aggregator by which the minimum τ of all reactions and the index μ are finally computed.

The tree data structure can be implemented in a pipelined fashion similar to what has been implemented in [64]. The basic approach is illustrated by an abstract view of the τ_m tree in the dotted oval of Figure 6. As shown in the diagram, a series of pipeline stages are connected one after another and a memory storing the nodes at each level of the tree is separately attached to each pipeline stage. What makes the pipelining possible is that unlike the case of the software IPQ, the tree is updated in one direction only—i.e., traversed from the root node to one of the bottom nodes. Because at most one node per tree level is traversed and updated, it is sufficient to have a single *read-compare-write* operation in each pipeline stage.

We make an analytical model of the NRM architecture representing the number of cycles needed for the computation of $\lceil D \cdot M \rceil$ reactions per each progress of time step, where the dependency factor D is defined as an average of the ratio of the number of reactions needed for the recomputation of propensities to the number of total reactions M in the network. To begin with, the overall latency can be expressed as

$$\begin{aligned} L_{nrm} &= L_{in} + \\ &\max(\max(L_{rnd}, L_{a_m}) + L_{\tau_\mu}, L_{a_m} + L_{\tau_m}) \\ &\quad + L_{\tau\text{-}tree} + L_{min} + \max(L_{fire}, L_{time}), \end{aligned} \tag{17}$$

where L_{τ_μ} is the latency for computing a tentative reaction time of the fired reaction in the previous time step; L_{τ_m} for computing a tentative time of a reaction

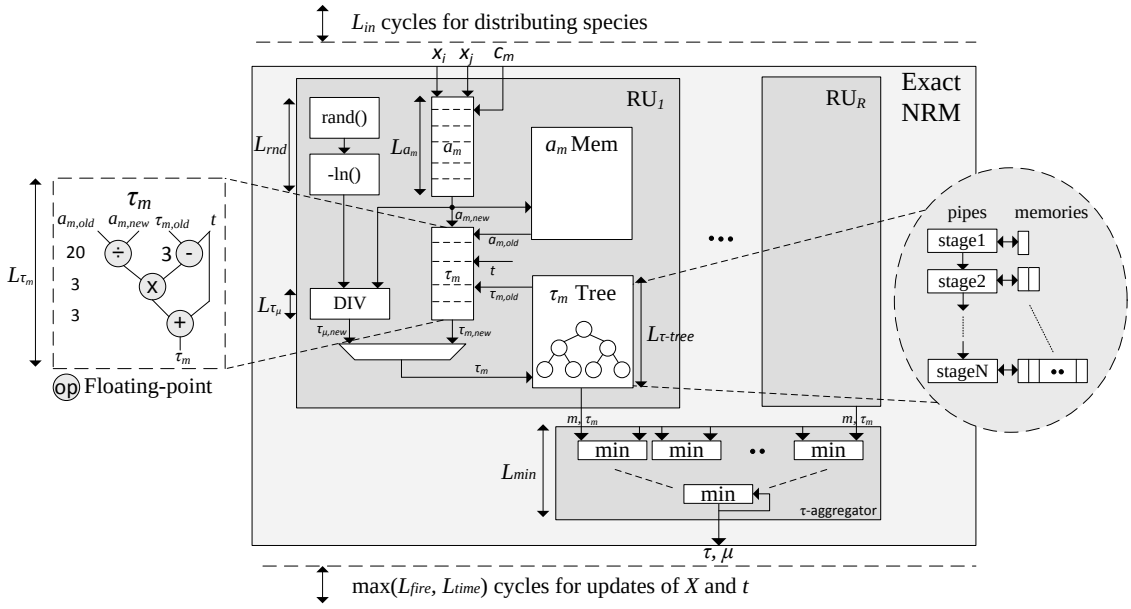


Figure 6: *Microarchitecture of NRM unit.*

affected by the previous fire; $L_{\tau-tree}$ for updating the IPQ tree with newly computed tentative reaction times. The rest parameters have been already defined in Section 4.1.1 for the FRM. Assuming $M_a \gg 1$, where M_a is the number of reactions affected by the previous fire, computation takes place mostly in the data path generating $t_{m,new}$ as opposed to the one generating τ_μ. For this reason, the first argument of the outer $\max$ operation can possibly be ignored, thus Equation (17) turns into

$$L_{nrm} = L_{in} + L_{a_m} + L_{\tau_m} + L_{\tau\text{-}tree} + L_{min} + \max(L_{fire}, L_{time}). \quad (18)$$

Because the tree height of the IPQ data structure expands up to $\lceil \log M \rceil$, each operational stage takes 3 cycles for *read-compare-write*, and only one node per tree level is to be traversed, the maximum latency for updating the tree with the tentative times of $\lceil D \cdot M/R \rceil$ reactions can be expressed as

$$L_{\tau-tree} = 3\lceil \log M \rceil + \left\lceil \frac{D \cdot M}{R} \right\rceil. \quad (19)$$

Therefore, substituting Equation (19), $L_{in} = L_{min} = \lceil \log R \rceil$, and the constants of $L_{a_m} = 6$, $L_{\tau_m} = 26$, $L_{fire} = 3$, and $L_{time} = 4$ into Equation (18), the total number of cycles required to compute on $\lceil D \cdot M \rceil$ reactions throughout the time evolution is expressed as

$$L_{nrm} = 2\lceil \log R \rceil + 3\lceil \log M \rceil + \left\lceil \frac{D \cdot M}{R} \right\rceil + 36. \quad (20)$$

Note that to achieve such latency as given by this analytical expression, a good load balancing of data across RUs is needed such that the reactions coupled with the previously executed reaction should be evenly distributed and mapped onto different RUs, which will be one of our potential research challenges in the future.

4.1.3 The Direct Method (DM)

The direct method [11] is similar to the FRM but only requires a single τ computation according to the equation given by

$$\tau = -\frac{\ln r_1}{a_0}, \quad (21)$$

where r_1 is a uniformly distributed random number in the interval (0,1) and

$$a_0 = \sum_{m=1}^{M} a_m(X(t)) = \sum_{m=1}^{M} c_m h_m(X(t)). \quad (22)$$

The reaction R_μ, determined as the one most likely to occur in the next infinitesimal time interval $(t + \tau, t +$

$\tau + d\tau$), is obtained by taking the index μ satisfying the inequalities given by

$$\sum_{m=1}^{\mu-1} a_m < a_0 r_2 \leq \sum_{m=1}^{\mu} a_m. \quad (23)$$

In other words, the propensities are successively accumulated until the accumulated value is greater than or equal to $a_0 r_2$, and μ is selected as the index of the last a_m term accumulated. r_2 is another independent random number uniformly distributed in the interval (0,1). Thus, the DM requires a total of two random numbers per each time step, effectively reducing the simulation run time consumed by the random number generation, as compared to the FRM.

The DM algorithm is explained below:

1. **Initialization.** *Initialize the number of molecules with $X(0)$ and set the current time to $t \leftarrow 0$.*
2. **Propensity functions.** *Calculate the propensity functions $a_m(X(t))$ for all M reactions; and take the sum of all propensities to get a_0.*
3. **Reaction time.** *Generate a uniformly distributed random number r_1 between 0 and 1; and calculate the next reaction time τ by Equation (21).*
4. **Reaction selection.** *Generate another independent, uniformly distributed random number r_2 between 0 and 1; and accumulate propensities until the next reaction index μ satisfying the inequalities of Equation (23)* is found.
5. **Reaction execution.** *Update the current time and the number of molecules by $t \leftarrow t + \tau$ and $X(t) \leftarrow X(t) + \mathcal{V}_\mu$, respectively, where $\mathcal{V}_\mu$ is a stoichiometric vector for the selected reaction.*
6. **Termination.** *If $t < t_{desired}$ or no more reactant species remain in the system, terminate the simulation; otherwise go to step 2.*

4.1.4 The Optimized Direct Method (ODM)

The optimized direct method [14] is a variant of the original direct method. As the NRM enhances the FRM by considering dependencies among different reactions, the ODM also relies on the dependency graph to enhance the DM. In addition, to boost the time spent on searching for the reaction which meets the inequalities given by Equation (23), the ODM, during its initialization phase, performs a presimulation for a certain period of time and reorders reactions such that reactions found to be executed more often than others are placed in higher priorities of the search order.

As for computing τ and μ, Equations (21) and (23) from the original DM algorithm can be reused. However, how to calculate the sum of propensities is somewhat different from the DM as only affected reactions are considered in calculating propensities. That is, an old propensity of the previous time step $a_{m,old}$ is subtracted from a new propensity of the current time step $a_{m,new}$. Therefore, the sum of propensities $a_{0,new}$ is given by

$$a_{0,new} = a_{0,old} + \sum_m (a_{m,new} - a_{m,old}), \quad (24)$$

where $a_{0,old}$ is the sum of propensities from the previous time step and m belongs to reactions affected by the last executed reaction.

The ODM algorithm is explained below:

1. **Presimulation.** *Simulate the network for a given period of time and gather a histogram of the number of executions for all M reactions; and sort the reactions such that reactions more frequently executed than others is placed in higher orders.*
2. **Initialization.** *Initialize the number of molecules with $X(0)$ and set the current time to $t \leftarrow 0$; generate a dependency graph; evaluate and maintain propensities a_m for all M reactions in the network; take the sum of all propensities a_0; and go to step 4.*
3. **Propensity functions.** *Using the dependency graph, evaluate and maintain propensities of affected reactions only; and update the sum of propensities using Equation (31).*
4. **Reaction time.** *Generate a uniformly distributed random number r_1 between 0 and 1; and calculate the next reaction time τ by Equation (21), i.e., $\tau = -(\ln r_1)/a_0$.*
5. **Reaction selection.** *Generate another independent, uniformly distributed random number r_2 between 0 and 1; and accumulate propensities in the order of the sorted frequency until the next reaction index μ satisfying the inequalities of Equation (23) is found, i.e., $\sum_{m=1}^{\mu-1} a_m < a_0 r_2 \leq \sum_{m=1}^{\mu} a_m$.*
6. **Reaction execution.** *Update the current time and the number of molecules by $t \leftarrow t + \tau$ and $X(t) \leftarrow X(t) + \mathcal{V}_\mu$, respectively, where $\mathcal{V}_\mu$ is*

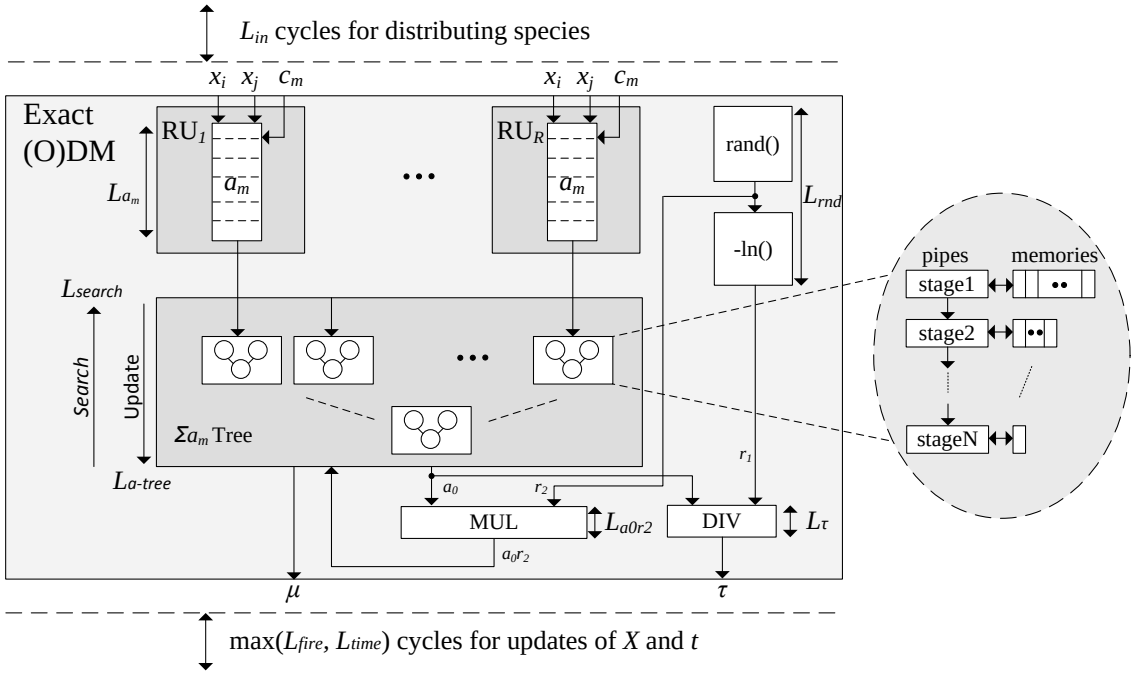


Figure 7: *Microarchitecture of (O)DM unit.*

a stoichiometric vector for the selected reaction channel.

7. **Termination.** *If $t < t_{desired}$ or no more reactant species remain in the system, terminate the simulation; otherwise go to step 3.*

Since the ODM is derived from the DM, the basic framework of hardware architecture can be shared between the two algorithms. Although propensities and their sum can be computed concurrently by multiple RUs in parallel, unlike the FRM or the NRM, it is not easy to parallelize the algorithms in hardware especially because of the sequential nature of how the reaction selection takes place in the algorithms. A search for the next reaction index takes at most M operations, and thus it can act as a limiting factor of the overall throughput according to Amdahl's law.

To address this problem, we can employ the binary-tree data structure suggested in the appendix of the NRM paper [12] for storing accumulated sums of propensities. By leveraging such data structure, we can improve the search time to $\log M$ operations. In addition, we can create a number of sub-trees at different granularity by partitioning the tree into fragments, making concurrent data processing of the tree possible. Furthermore, since only one node at a tree level is to be processed, a single operation of either *read-accumulate-write* or *read-subtract-compare* can be performed in a pipelined stage without adding much hardware resources.

Figure 7 shows an implementation of the DM and ODM algorithms. The reaction data are distributed across multiple RUs to generate propensities, which are then continuously passed into the $\sum a_m$ tree unit comprising a number of partitioned sub-tree unit arranged in a tree form. Similar to what we have shown in the NRM, each sub-tree unit consists of a number of operational stages with dedicated memories attached to each stage for storing the node values at the same tree level. Notice unlike the tree unit shown in the NRM architecture, the tree nodes are updated from a bottom node to the root node. As the sum of propensities a_0 is available at the output of the $\sum a_m$ tree unit, a uniformly distributed random number r_2 between 0 and a_0 is generated by the multiplier and the random number generator, and the generated random number is written back into the tree unit for further search of the next reaction index μ, where the tree nodes are traversed in a reverse direction, i.e., from the root node to a bottom node. While the tree nodes are traversed, the next simulated time τ is also evaluated via a division operation on a_0 and r_2.

We make an analytical model of the DM and ODM architecture expressing the latency needed to advance a single step of the time evolution. To begin with, the total number of cycles is expressed as

$$L_{(o)dm} = \max(L_{re}, L_t), \tag{25}$$

where L_{re} is the latency for the selection and execution of the next reaction and L_t is the latency for computing the simulated time. L_{re} and L_t are further expressed as

$$L_{re} = L_{a_0} + L_{a_0 r_2} + L_{search} + L_{fire} \tag{26}$$

and

$$L_t = \max(L_{a_0}, L_{rnd}) + L_\tau + L_{time}, \tag{27}$$

where L_{a_0} is expressed as

$$L_{a_0} = L_{in} + L_{a_m} + L_{a\text{-}tree}. \tag{28}$$

The new parameters defined for the NRM are as follows: L_{a_0} for computing a sum of propensities; L_{a-tree} for updating the tree nodes with newly computed propensities; $L_{a_0 r_2}$ for generating a uniformly distributed random number between 0 and a_0; L_{search} for traversing the tree nodes to find the next reaction; and L_τ for computing the time step τ by dividing r_1 by a_0.

Given the height of the tree is $\log M$ and an operation of *read-accumulate-write* is needed at each tree level, the latency for updates is expressed as

$$L_{a\text{-}tree} = 5\lceil \log M \rceil + \left\lceil \frac{D \cdot M}{R} \right\rceil, \tag{29}$$

in order to process a total of $\lceil D \cdot M/R \rceil$ reactions. Likewise, given the height of the tree is $\log M$ and an operation of *read-subtract-compare* is needed at each tree level, the latency for a search is expressed as

$$L_{search} = 5\lceil \log M \rceil. \tag{30}$$

In the equations above, we have assumed 1 cycle for *read*, *write*, and *compare* each and 3 cycles for *accumulate* and *subtract* each. L_{search} does not contain the second term because only one node at a tree level is to be traversed (See [12] for details).

Overall, substituting $L_{in} = \lceil \log R \rceil$, $L_{a_m} = 6$, $L_{rnd} = 16$, $L_{a_0 r_2} = 3$, $L_{fire} = 3$, $L_\tau = 20$, and $L_{time} = 4$ into Equations (26)–(30), the overall latency

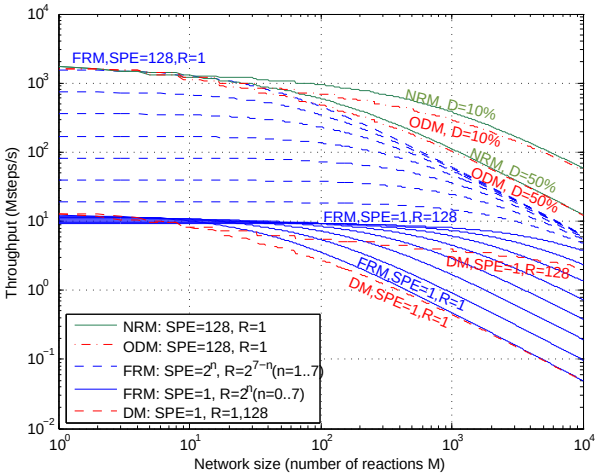


Figure 8: *Performance of various exact SSAs.*

given by Equation (25) becomes

$$\begin{aligned} L_{(o)dm} &= \max(L_{re}, L_t) \\ &= \max\Bigg\{ \lceil \log R \rceil + 10\lceil \log M \rceil \\ &\qquad + \left\lceil \frac{D \cdot M}{R} \right\rceil + 12, \\ &\qquad \max\left(\lceil \log R \rceil + 5\lceil \log M \rceil \right. \\ &\qquad \left. + \left\lceil \frac{D \cdot M}{R} \right\rceil + 6, 16 \right) + 24 \Bigg\} \end{aligned} \tag{31}$$

where $D = 1$ for the DM and $0 < D < 1$ for the ODM.

4.1.5 Performance Comparison

Table 3 summarizes the analytical latency models of various architectures discussed so far, and conservatively assuming a low clock frequency of 500 MHz in cost-effective legacy 180nm or 90nm technology, Figure 8 compares their throughput on network models with varying network sizes (M) and dependencies (D). In addition, assuming a chip that can fit a maximum of 128 RUs[5], to see the effect of parallelism, the graphs plot the throughput for different configurations in terms of the number of RUs per SPE (R) and the number of SPEs in the SSSoC. We define the throughput as the number of simulated time steps per second in accordance with

[5]This matches typical GPU architectures.

Table 3: *Latency cycles of various exact hardware units.*

Exact H/W unit	Latency in # of cycles
FRM	$L_{frm} = 2\lceil \log R \rceil + \lceil \frac{M}{R} \rceil + 40$
NRM	$L_{nrm} = 2\lceil \log R \rceil + 3\lceil \log M \rceil + \lceil \frac{D \cdot M}{R} \rceil + 36$
(O)DM [a]	$L_{(o)dm} = \max \left\{ \lceil \log R \rceil + 10\lceil \log M \rceil + \lceil \frac{D \cdot M}{R} \rceil + 12, \right.$ $\left. \max \left(\lceil \log R \rceil + 5\lceil \log M \rceil + \lceil \frac{D \cdot M}{R} \rceil + 6, 16 \right) + 24 \right\}$

[a] $D = 1$ for the DM and $0 < D < 1$ for the ODM.

Table 4: *Comparison of latency for reaction selection.*

Architecture	Update	Search
FRM	$\lceil \log R \rceil + \lceil \frac{M}{R} \rceil$	1
DM	$5\lceil \log M \rceil + \lceil \frac{M}{R} \rceil$	$5\lceil \log M \rceil$
NRM	$3\lceil \log M \rceil + \lceil \frac{D \cdot M}{R} \rceil$	1
ODM	$5\lceil \log M \rceil + \lceil \frac{D \cdot M}{R} \rceil$	$5\lceil \log M \rceil$

literature [50, 45], and assume each SPE simulates a given network independently of others. (In Section 5, we will specifically explore opportunities for a SPE network that can be flexibly configured, depending on network characteristics, to act as either many small or one large simulation system.) We use dependency factors of $D = 10\%$ and 50% to represent models at different levels of coupling. In fact, counting the number of affected reactions on actual models, we measured 10% for heat shock response [65] and 50% for intracellular viral infection [33].

Somewhat surprisingly, and contrary to the situation in software, despite the parallelism exploited in all architectures, an FRM or NRM outperforms, to varying extents, a DM or ODM, respectively, as network size grows. Upon closer inspection, this is mainly because the reaction selection part of the FRM-based architectures takes by far less latency than that of the DM-based architectures. The number of cycles needed to update relevant data and search for the next reaction is summarized in Table 4. Clearly, the FRM-based architectures perform predominantly better in searching for the next reaction—constant vs. logarithm of network size.

In the case of the update phase, however, where the tentative times τ_m are evaluated for the FRM and NRM and the partial sums $\sum a_m$ are evaluated for the DM and ODM, subtle distinctions exist in comparing the latency among different architectures. Firstly, with no parallelism ($R = 1$), the latency of both FRM and DM approaches M as the network size increases. With a certain level of parallelism ($R > 1$), the FRM outperforms the DM because the logarithmic term is not negligible any more, thus the latency degradation of the DM is much noticeable than that of the FRM. Secondly, regardless of the level of parallelism, the dependency factor D taken into consideration makes the NRM and ODM outperform the FRM and DM, depending on the level of coupling among reactions. Lastly, while the NRM generally outperforms the ODM owing to its fast search time, as either the network size or the dependency factor increases, the throughput of the ODM approaches that of the NRM because the logarithmic term in the update phase becomes eminent compared to the linear term.

Overall, in the graphs of Figure 8, for small network sizes, the throughput ranges from around 10 Msteps/s for a single SPE up to 1280 Msteps/s for a chip with 128 SPEs. For large network sizes, the single SPE performance approaches a peak rate of R reactions every cycle for a maximum throughput of $500R$ million reactions or $500\frac{R}{M}$ million steps per second. We can note that the peak throughput for the full SSSoC is equivalent to the performance of a single SPE with $R = 128$ (i.e., $64/M$ billion steps per second).

4.2 Approximate SSAs and Architectures

In this section, we will discuss approximate SSAs focusing on a hardware implementation of the Gillespie's improved τ-leap algorithm [18]. In addition, we will briefly mention how our SSSoC architecture can be leveraged to implement so-called hybrid SSAs.

4.2.1 The τ-leap Method

Gillespie originally proposed the τ-leap method in 2001 [17] and, later in 2003, improved its simulation accuracy by calculating the variance, as well as the mean value, of the propensity changes over a leap period [18]. Both of the τ-leap methods fire, within a leap time, all reactions in the system as many times as given by Poisson distributions, whereas exact methods execute only a single reaction at every time step. We will consider implementing the improved version as it is a superset of the other.

The τ-leap method is derived from the assumption that the leap time τ must be small enough such that the propensity changes across all reaction channels remain infinitesimally small during the leap period $[t, t+\tau)$, where t is the current time. Given the state vector containing the number of molecules for each species at time t is $X(t) = \mathbf{x}$, τ can be obtained from the following equations [18]:

$$f_{mm'}(\mathbf{x}) = \sum_{i=1}^{N} \frac{\partial a_m(\mathbf{x})}{\partial x_i} v_{im'}, \tag{32}$$

$$\eta_m(\mathbf{x}) = \sum_{m'=1}^{M} f_{mm'}(\mathbf{x}) a_{m'}(\mathbf{x}), \tag{33}$$

$$\sigma_m^2(\mathbf{x}) = \sum_{m'=1}^{M} f_{mm'}^2(\mathbf{x}) a_{m'}(\mathbf{x}), \tag{34}$$

$$\begin{aligned} \tau &= \min_{m\in[1,M]} \left\{ \tau_{\eta,m},\, \tau_{\sigma,m} \right\} \\ &= \min_{m\in[1,M]} \left\{ \frac{\epsilon a_0(\mathbf{x})}{|\eta_m(\mathbf{x})|}, \frac{\epsilon^2 a_0^2(\mathbf{x})}{\sigma_m^2(\mathbf{x})} \right\}, \end{aligned} \tag{35}$$

where $m, m' \in [1, M]$; η_m and σ_m^2 are respectively a mean and a variance for the m^{th} reaction; ϵ is an error control parameter close to 0 ($0 < \epsilon \ll 1$); and a_0 is a summation of the propensities of all reactions. Notice that, prior to advancing the simulated time by the leap time τ, the leap condition must be checked if τ is greater than a few multiples of $1/a_0(\mathbf{x})$, which is the mean time step for the exact SSA. If τ fails to meet such condition, it would be efficient to run the exact method rather than to leap over time.

The τ-leap algorithm is explained below:

1. **Initialization.** *Initialize the number of molecules with $X(0)$ and set the current time to $t \leftarrow 0$.*
2. **Propensity functions.** *Calculate propensities a_m for all M reactions and take a summation of a_m to get a_0.*
3. **Reaction time.** *Calculate τ according to Equations (32) through (35).*
4. **Leap condition.** *If $\tau < \beta/a_0$ (where $1 < \beta < 10$), execute an exact SSA for a number of successive time steps (e.g. 100) and go to step 2; otherwise proceed to the next step;*
5. **Poisson generation.** *For each $m \in \{1 \ldots M\}$, generate a Poisson random number $\mathcal{K}_m$ with a parameter of $a_m\tau$.*
6. **Reaction execution.** *Update the current time and the number of molecules by $t \leftarrow t+\tau$ and $X(t) \leftarrow X(t) + \sum_{m=1}^{M} \mathcal{K}_m \mathcal{V}_m$, respectively, where $\mathcal{V}_m$ is a stoichiometric vector for the m^{th} reaction.*
7. **Termination.** *If $t < t_{desired}$ or no more reactant species remain in the system, terminate the simulation; otherwise go to step 2.*

An architecture implementing the τ-leap algorithm is shown in Figure 9. It mainly consists of two data paths to compute two different tentative times. On one side of the data path, the η and τ_η units calculate a tentative time $\tau_{\eta,m}$ of reaction R_m by evaluating the mean η_m of the propensity change in reaction R_m, and on the other side, the σ^2 and τ_σ units calculate another tentative time $\tau_{\sigma,m}$ of reaction R_m by evaluating the variance σ_m^2 of the propensity change in reaction R_m. Calculation of some intermediate variables such as $f_{mm'}$, a_m, and a_0 can be shared between the η unit and the σ^2 unit, thus these variables are computed only within the η unit and are passed onto each of the σ^2, τ_η and τ_σ units, accordingly. As all M tentative times have passed into each of the τ_η and τ_σ units in a pipelined fashion, the minimum value is determined as $\tau_{\eta,min} = \min_{m=1}^{M} \tau_{\eta,m}$ and $\tau_{\sigma,min} = \min_{m=1}^{M} \tau_{\sigma,m}$ by the τ_η unit and the τ_σ unit, respectively. Subsequently, the leap checker compares both of the minimum tentative times and take the smaller value as the leap time τ.

To avoid the situation where only a few reactions are leaped over thus it would be rather efficient to reject

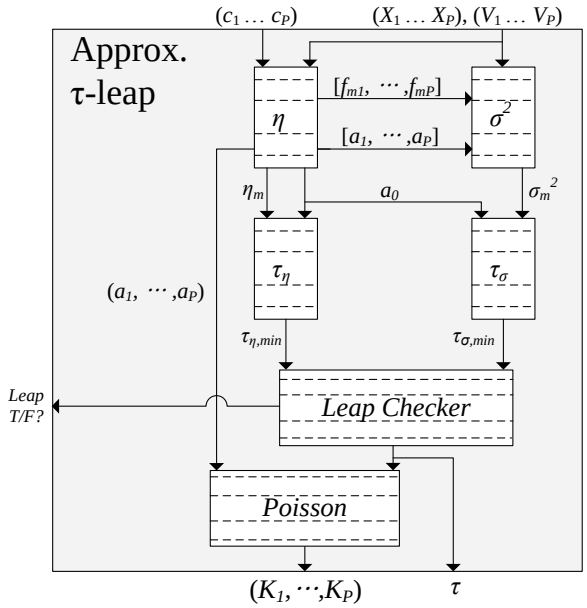


Figure 9: *Top block diagram of τ-Leap unit.*

τ-leap and run the exact unit instead, the leap checker compares τ with some threshold level (i.e., multiples of $1/a_0$) and notifies the result to the external control unit. If τ fails to be greater than the threshold, the control unit disables the τ-leap unit and triggers the exact unit with a signal, *Leap T/F*. Otherwise, the Poisson unit next to the leap checker generates a Poisson random number for each reaction with a Poisson parameter of $a_m\tau$. To parallelize the process, P Poisson random numbers $(\mathcal{K}_1 \dots \mathcal{K}_P)$ are generated at a time and are driven to the external control unit for generation of a state change vector $\mathcal{V}$, given by $\mathcal{V} \leftarrow \mathcal{K}_m\mathcal{V}_m$, which in turn is used for an update of the system state according to $X \leftarrow X + \mathcal{V}$. The simulated time t is also to be advanced by amount τ, i.e., $t \leftarrow t + \tau$.

As can be noted in Equations (32)–(35), the process of calculating η_m and σ_m^2 is accomplished by numerous matrix operations. To achieve better performance gain, we can exploit data parallelism existing in the matrix operations and implement the τ-leap unit as a pipelined vector machine. Consequently, each unit contains the same kind of P matrix operators in parallel to operate on the whole sequence of data at a time. For this reason, both η and σ^2 units take a P sequence of data in a vectorized form as labeled in Figure 9. We will now elaborate further on details of each unit by referring to the microarchitecture diagram shown in Figure 10.

η unit. The η unit computes a sequence of sum-of-product operations given by Equations (32) and (33). The first stage of the pipeline executes, on the m^{th} reaction, a vector multiplication of

$$\begin{bmatrix} f_{m1} & \cdots & f_{mM} \end{bmatrix} = \begin{bmatrix} \frac{\partial a_m}{\partial x_i} & \frac{\partial a_m}{\partial x_j} \end{bmatrix} \begin{bmatrix} v_{i1} & \cdots & v_{iM} \\ v_{j1} & \cdots & v_{jM} \end{bmatrix}. \quad (36)$$

Since the number of reactions for a large network is typically on the order of several hundreds to thousands, rather than operating on the whole M sequence of data, we apply strip-mining and perform a partial vector operation of size P one after another until the full vector length M has reached. Therefore, the hardware is pipelined to execute $\lceil M/P \rceil$ vector multiplications of the form above with P sets of $f_{mm'}$ data processed at a time (i.e., $m' = \{1 \dots P\}$).

The derivatives of the propensity function a_m of reaction R_m with respect to x_i and x_j can easily be evaluated due to the fact that the reaction is only constrained by three elementary types of reactions (see Section 2.1). Based on reaction types, six different combinations of derivatives are possible (0, c_m, $c_m(x_i - 1/2)$, $c_m x_i$, $c_m(x_j - 1/2)$, and $c_m x_j$) and the operations are implemented in the $\partial a/\partial x$ unit as drawn in the upper left corner of Figure 10.

In parallel to the evaluation of those $f_{mm'}$ data, the propensity functions of all M reactions need to be calculated and, as before, P sets of input data $\{(c_1, x_s, x_t) \dots (c_P, x_u, x_v)\}$ among all M reactions are passed into P propensity units (a_1 through a_P unit) until all $\lceil M/P \rceil$ sets of data are driven through the pipeline.

In the following stages of the pipeline, the mean value associated with a reaction is computed by taking the sum of $f_{mm'}a_{m'}$ as given by Equation (33). Reiterating the equation, we can write the vector multiplication in the form of

$$\eta_m = \begin{bmatrix} f_{m1} & \cdots & f_{mM} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_M \end{bmatrix}. \quad (37)$$

This vector operation is implemented using P multipliers followed by a binary tree-structured η-aggregator, and again, $\lceil M/P \rceil$ operations are pipelined to finally get the mean value η_m. Concurrent to the η-aggregator, the a-aggregator accumulates all propensities to obtain a_0.

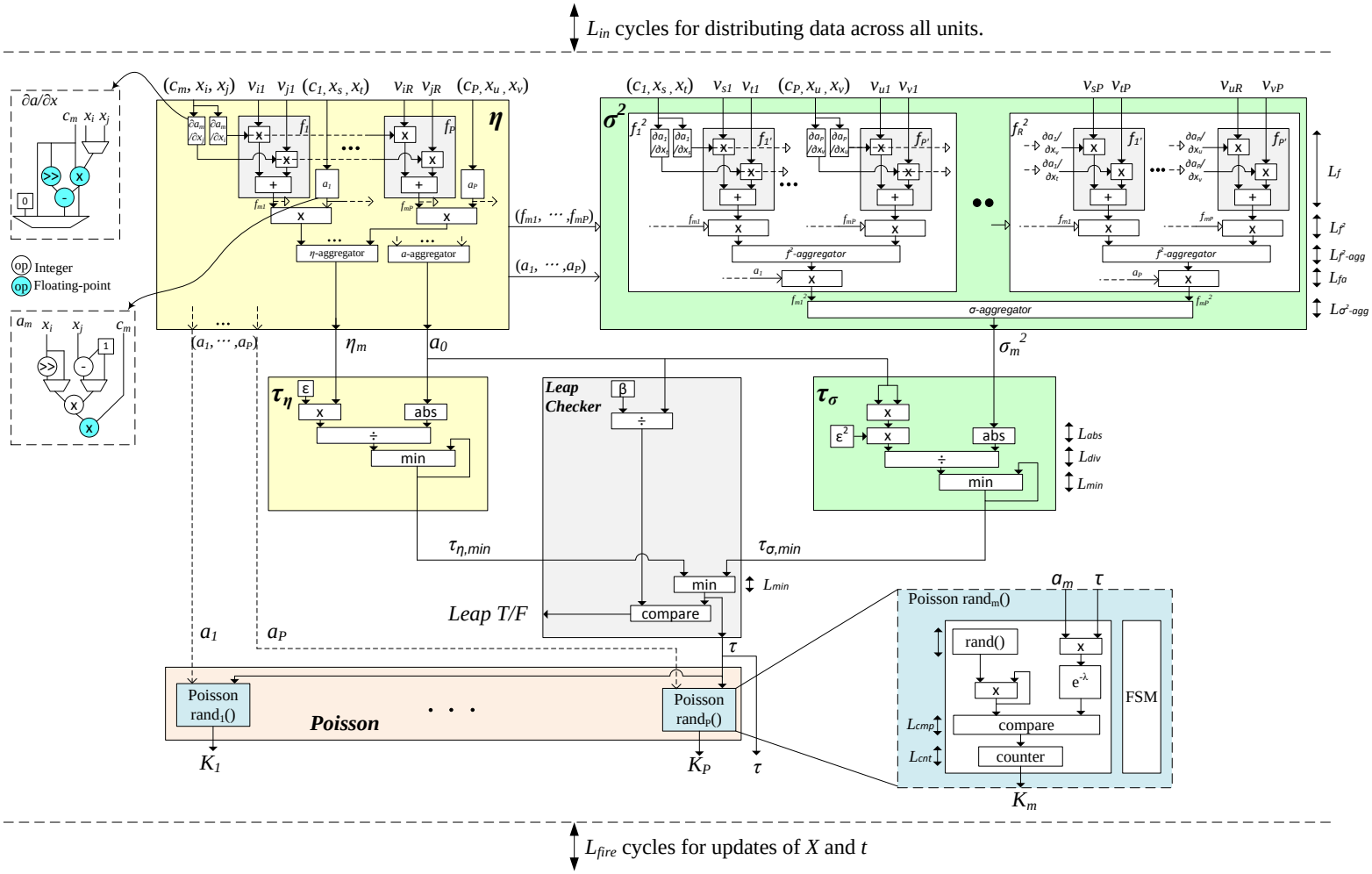


Figure 10: *Microarchitecture of τ-Leap unit.*

σ^2 **unit.** The basic flow of operations is similar to the η-unit but an additional level of parallelism is necessary for computation of the $f^2_{mm'}$ term in Equation (34). In each of the f^2_j units ($j = 1, \ldots, P$), a vector of $(f_{m1} \cdots f_{mR})$ is multiplied by a column vector of the same $f_{mm'}$ matrix. For instance, in the f^2_1 unit, a vector multiplication of

$$f^2_{m1} = \begin{bmatrix} f_{m1} & \cdots & f_{mP} \end{bmatrix} \begin{bmatrix} f_{11} \\ \vdots \\ f_{P1} \end{bmatrix} \tag{38}$$

is performed, whereas the f^2_P unit evaluates a vector multiplication of

$$f^2_{mP} = \begin{bmatrix} f_{m1} & \cdots & f_{mP} \end{bmatrix} \begin{bmatrix} f_{1P} \\ \vdots \\ f_{PP} \end{bmatrix}. \tag{39}$$

If there are more than P reactions to process ($M > P$), the same vector multiplications can be performed by streaming data into the pipeline. These vector operations are performed via the multipliers followed by the f^2-aggregator.

As a vector of $(f^2_{m1} \cdots f^2_{mP})$ is available at the output of all f^2-aggregator units, The following vector multiplication given by Equation (34) is evaluated:

$$\sigma^2_m = \begin{bmatrix} f^2_{m1} & \cdots & f^2_{mM} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_M \end{bmatrix}. \tag{40}$$

That is, the vector $(f^2_{m1} \cdots f^2_{mP})$ is multiplied by the propensity vector $(a_1 \cdots a_P)$ obtained from the η unit, using additional multipliers and the σ-aggregator next to the f^2-aggregator units.

τ_η and τ_σ units. The τ_η unit evaluates $\tau_{\eta,m}$ for all M reactions according to $\tau_{\eta,m} = \epsilon a_0 / |\eta_m|$, as expressed in Equation (35), and outputs their minimum (i.e., $\tau_{\eta,min} = \min_{m=1}^{M} \tau_{\eta,m}$) to the leap checker. The control parameter ϵ is a constant value preloaded in a register. Likewise, the τ_σ unit evaluates $\tau_{\sigma,m}$ for all reactions, according to $\tau_{\sigma,m} = \epsilon^2 a_0^2 / \sigma_m^2$, and outputs their minimum (i.e., $\tau_{\sigma,min} = \min_{m=1}^{M} \tau_{\sigma,m}$) also to the leap checker. In this case, an additional multiplier is needed to compute a_0^2. ϵ^2 is also a constant value preloaded in a register.

Leap Checker. This unit simply compares the two tentative times of $\tau_{\eta,min}$ and $\tau_{\sigma,min}$, each generated by the τ_η unit and the τ_σ unit, respectively, and determines the smaller value to be the leap time τ. In addition, as argued in the discussion of the algorithm, the leap condition is checked by comparing the selected τ with a threshold of β/a_0, where β ranges typically from 1 to 10. If τ turns out to be greater than the threshold, the leap flag is enabled; otherwise it becomes disabled to alert the external control unit.

Poisson unit. Finally, M Poisson random numbers are generated by executing the Poisson unit $\lceil M/P \rceil$ times. It contains P subunits which generate Poisson random numbers of $(\mathcal{K}_1 ... \mathcal{K}_P)$ with the expected occurrences of $(\lambda_1, \ldots, \lambda_P) = (a_1\tau, \ldots, a_P\tau)$. Each subunit is implemented using a simple algorithm [66] for which a pseudo-code is listed in the following.

Algorithm 1 *A simple algorithm for Poisson-distributed random number generation*

procedure $Poisson(\lambda)$
1: $k \leftarrow 0$ $p \leftarrow 1$ $L \leftarrow e^{-\lambda}$
2: **repeat**
3: $k \leftarrow k + 1$
4: Generate a uniform random number u in $[0, 1]$
5: $p \leftarrow p \times u$
6: **until** $p > L$
7: **return** $k - 1$

In the algorithm, a uniformly distributed random number is generated at every iteration of the loop until p becomes greater than L, where p is the accumulated multiplications of a sequence of uniform random numbers generated over a number of iterations and L is an exponential value calculated by $e^{-\lambda}$. λ is an expected value of a Poisson-distributed random variable. Upon meeting the condition to break from the loop, $k-1$ is determined as a Poisson random number, where k is a counter value representing the total number of loop iterations.

As can be seen, the complexity of this algorithm is linear in λ. We will eventually have to explore enhanced algorithms to address this complexity. However, due to its ease of implementation in hardware, we use this algorithm as a baseline for further improvements. An implementation of this algorithm is sketched in Figure 10. As latency for the initial calculation of $p \leftarrow p \times u$ can be amortized while calculating $e^{-\lambda}$ by means of a multiplication and a table look-up, and a valid p is generated every cycle, the hardware can potentially perform better than software despite the same complexity limited by λ.

Just as with the exact SSA units, we follow along the critical path of the pipeline, discarding all potential latencies that can be hidden, and express the total latency as a function of the mean of Poisson numbers $\mathcal{K}_{mean}$, the network size M, the number of parallel processing elements P. As a starter, the total latency is expressed simply as

$$L_{leap} = L_{in} + L_{\sigma^2} + L_{\tau_{\sigma^2}} + L_{Poisson} + L_{fire}, \quad (41)$$

where L_{in} is the number of cycles for distributing input data across all units; L_{σ^2} is for evaluating the variance ${\sigma_m}^2$ of all reactions; $L_{\tau_{\sigma^2}}$ is for computing the minimum $\tau_{\sigma,min}$ and checking the leap condition on its value; $L_{Poisson}$ is for generating Poisson random numbers of all reactions; and L_{fire} is for executing all reactions based on their corresponding Poisson parameters. (In the following, refer to Figure (10) for newly defined latency parameters.)

Given $P + P^2$ input processing elements in the η and σ^2 units, and leveraging the banking of memories, we assume the input latency to be

$$L_{in} = \lceil \log (P + P^2) \rceil, \quad (42)$$

where P is defined as a degree of parallelism implying the number of units which operate concurrently on the data—somewhat similar to the term, *vector length*, used in vector processors. Specifically, P refers to the number of parallel processing units which include the f_P, ${f_{P'}}^2$, and ${f_P}^2$ units, the Poisson unit and so forth.

Both of the η and σ^2 units operate concurrently so the critical path lies in the one with a longer pipeline depth, which in this case is the σ^2 unit. Hence, we only consider the data path associated with computing $\tau_{\sigma^2,min}$. Simply summing the latency of each component in the σ^2 unit, the total latency is expressed as

$$L_{\sigma^2} = L_f + L_{f^2} + L_{f^2\text{-}agg} + L_{fa} + L_{\sigma^2\text{-}agg}, \quad (43)$$

and the latency of both f^2 and σ^2 aggregator units is further expressed as

$$L_{f^2\text{-}agg} = L_{\sigma^2\text{-}agg} = \lceil \log P \rceil + \left\lceil \frac{M}{P} \right\rceil, \quad (44)$$

in order to process the data of M reactions using P concurrent units.

In the τ_{σ^2} unit, the latency for calculating a_0^2 can be hidden as a_0 is available even before a valid output comes from the σ^2 unit. Likewise, the latency for calculating β/a_0 in the leap checker can also be amortized. In addition, we can discard the latency for the compare operation in the leap checker as it can also be hidden by Poisson random number generation. Thus, the latency accounting for both the τ_σ^2 unit and the leap checker is expressed as

$$L_{\tau_{\sigma^2}} = L_{abs} + L_{div} + 2L_{min} + M, \quad (45)$$

in order to process the data of M reactions.

Determining the latency of the Poisson unit is not a trivial task as it depends on the generated Poisson value which turns out to be random. To address this, we define a parameter $\mathcal{K}_{mean}$ as the mean value of a set of generated Poisson numbers collected from experimental runs. In other words, $\mathcal{K}_{mean}$ is the number of fires or time steps advanced per reaction per leap, requiring $\mathcal{K}_{mean} + 1$ compare operations followed by updates of the counter value. Despite several cycles are initially needed to fill up the pipeline for evaluating $p \leftarrow p \times u$ and $e^{-\lambda}$ in hardware, once the pipeline is filled with valid data, the multiplier provides the valid output of $p \times u$ at every cycle. In addition, calculation of $e^{-\lambda}$ is needed only once per a single Poisson random number generation. Furthermore, the combined latency of the uniform random number generator and the multiplier can effectively be hidden if we let them run from the very beginning (e.g., $t = 0$). Hence, many cycles can potentially be hidden, and it is reasonable to express the latency as

$$L_{Poisson} = (\mathcal{K}_{mean} + L_{cmp} + L_{cnt}) \times \left\lceil \frac{M}{P} \right\rceil. \quad (46)$$

Assuming we can further hide the latencies for updating the simulated time and loading both the state change vector and the system state vector while generating Poisson random numbers, the latency for firing the whole M reactions (i.e., $X + \mathcal{K}_m \mathcal{V}_m$) is expressed as

$$L_{fire} = (L_{mul} + L_{add} + L_{store}) \times \left\lceil \frac{M}{P} \right\rceil. \quad (47)$$

All in all, substituting Equations (42)–(47) into Equation (41) and using constant values of $L_f = 9$, $L_{f^2} = L_{fa} = L_{abs} = 3$, $L_{div} = 20$, $L_{min} = L_{cmp} = L_{cnt} = L_{add} = L_{store} = 1$, $L_{mul} = 2$, the total latency is expressed as

$$\begin{aligned} L_{leap} = {} & \lceil \log (P + P^2) \rceil + 2 \lceil \log P \rceil \\ & + (7 + \mathcal{K}_{mean}) \times \left\lceil \frac{M}{P} \right\rceil + M + 40. \end{aligned} \quad (48)$$

In contrast to the exact methods, translating Equation (48) into a throughput equation is not a

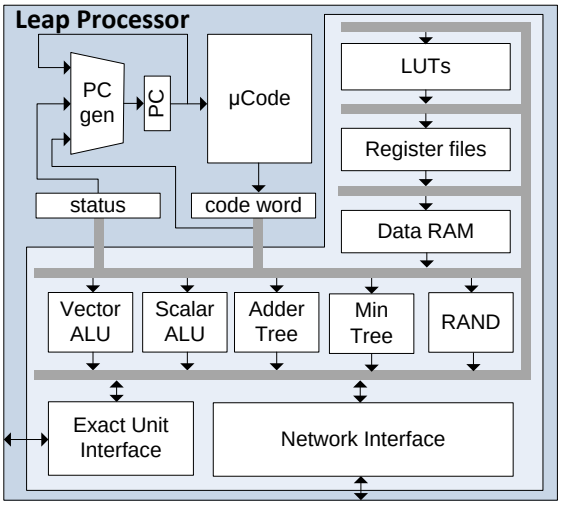


Figure 11: *Microcoded leap processor.*

simple task as the total number of reactions executed per leap is nondeterministic—i.e., not only is $\mathcal{K}_{mean}$ random but also a function of propensities and leap times. Therefore, rather than plotting a generalized throughput graph in this section, we will show experimental results of certain network examples in Section 4.3.

There is a variety of methods and new ones continue to be developed. Methods differ in their complexity and accuracy for determining the leap condition and the τ interval, but no clearly superior approach seems to exist [33, 21, 19]. Furthermore, as will be confirmed by experimental results in Section 4.3, derivation of τ by bounding expected propensity variations requires relatively complex and irregular computations that are not amendable to easy hardware parallelization, resulting in only moderate speedups. Towards this end, a research challenge will be to design an approximate unit that can flexibly support current and future leaping methods yet is specialized enough to do so optimally. We can conclude that a direct hardware implementation of the τ-leap unit with such high hardware complexity may not be the best approach. Instead, we can envision a specialized, microcoded processor (as sketched in Figure 11) that can be reprogrammed to execute different leaping variants on top of a custom data path, which includes dedicated hardware for leaping-specific operations such as random number generators and a_m pipelines and memories. With careful co-design of microarchitectures, this should allow us to achieve the same performance as full custom hardware while supporting a wide range of leaping algorithms.

4.2.2 Hybrid SSA

Overall, the envisioned SPE design represents a flexible yet high-performance architecture that combines a programmable τ processor with a dedicated exact SSA unit. In addition to operating in any of the two modes, the architecture naturally supports hybrid simulations in which reactions are statically [28, 34, 25, 29] or dynamically [23, 24] partitioned into slow and fast ones running in parallel on the exact and approximate units, respectively. The central controller thereby maintains the classification of reactions and distributes them to the appropriate processing unit. To support dynamic partitioning, the execution units can in turn be extended to include reaction classifiers that report status changes back to the controller, e.g. following the procedure outlined in [23]. Finally, the router receives τ and $\mathcal{V}$ from both units and, depending on the smaller τ, combines them into an overall time and species update. In this context, we will also investigate support for other dynamic partitioning approaches [24], where we will address challenges for integrating such support into the controller and execution units while maintaining overall high performance across all supported SPE modes.

4.3 Experiments and Results

For performance comparison to hardware models, we simulate several network examples (both artificial and real ones), on a 2.67 GHz Intel Core i7-920 processor, by using the StochKit2 [36] software package, where we explicitly select, among many available methods, the ODM [14] and the τ-leap [18] to achieve the best performance results in software. Based on the simulation information captured during software runs, we evaluate the speed-ups of our analytical hardware models of the NRM and the τ-leap against software realizations of the ODM and the τ-leap, respectively. We select particularly the NRM and compare its performance with that of the ODM in software owing to its promising result of performance in hardware, as previously discussed in Section 4.1.5.

4.3.1 Partially Coupled Decay System

We define an artificial network named Partially Coupled Decay System (PCDS) whose network size and dependency factor can easily be adjusted. A general

form of the PCDS network containing $m \times n$ reactions and $m \times (n+1)$ species is expressed below:

$$\begin{aligned} R_{set_1} &= \{S_1 + S_{i_1} \xrightarrow{c_1} \emptyset : m+1 \leq i_1 \leq m+n\} \\ R_{set_2} &= \{S_2 + S_{i_2} \xrightarrow{c_2} \emptyset : m+n+1 \leq i_2 \leq m+2n\} \\ &\vdots \\ R_{set_m} &= \{S_m + S_{i_m} \xrightarrow{c_m} \emptyset : \\ &\qquad m+(m-1)n+1 \leq i_m \leq m+mn\}, \end{aligned} \tag{49}$$

where all of the indices ($i_1 - i_m$, m, and n) are positive integers; and m and n are constant values which can be parameterized depending on the desired network size and dependency factor. Notice that each of the reaction sets, R_{set_1} – R_{set_m}, contains n reactions which are coupled each other via the first reactant species, and reactions across different sets are independent from each other as they do not share any of the same reactants. For example, if we were to create a PCDS network having 4 reactions, 6 species, and a dependency of 50%, the generated network would look as follows:

$$\begin{aligned} R_{set_1} &= \{S_1 + S_3 \xrightarrow{c_1} \emptyset, S_1 + S_4 \xrightarrow{c_1} \emptyset\} \\ R_{set_2} &= \{S_2 + S_5 \xrightarrow{c_2} \emptyset, S_2 + S_6 \xrightarrow{c_2} \emptyset\} \end{aligned} \tag{50}$$

4.3.2 Performance Comparison for PCDS Networks

For both the ODM and τ-leap simulations, we created a range of PCDS networks with predefined initial conditions by varying network size and dependency: $M = \{1,000i : 1 \leq i \leq 10, i \in \mathbf{N}\}$, $D = \{10\%, 50\%\}$, and $\{c = 1e\text{-}9, X_{t0} = 1e5\}$. The same rate constant was applied to all reactions, and all reactants were initialized with the same number of molecules. Using a Python script which takes on libSBML libraries [67], we generated PCDS networks with combinations of M and D in SBML-L2V4 format and subsequently converted them into StochKit2 format using the tool (sbml2stochkit) included in the StochKit2 software package.

For the ODM simulations, we used the latest beta-3 version of StochKit2 to simulate each network for a simulated time of 15s. We simulated a given network 11 times and during each run, we measured a dataset of the simulation time ($t_{odm,sw}$) and the number of time steps (N_{step}), and extracted the one containing the simulation time occurred to be the median of all 11 experiments. Based on such measurements, we were able to predict the simulation time in hardware, as if the hardware of the NRM would have simulated the same model, according to

$$t_{nrm,hw} = \frac{L_{nrm} \times N_{step}}{f} \tag{51}$$

where L_{nrm}, given by Equation (20), is the latency in cycle needed to simulate a single time step and f is an operating frequency of the hardware.

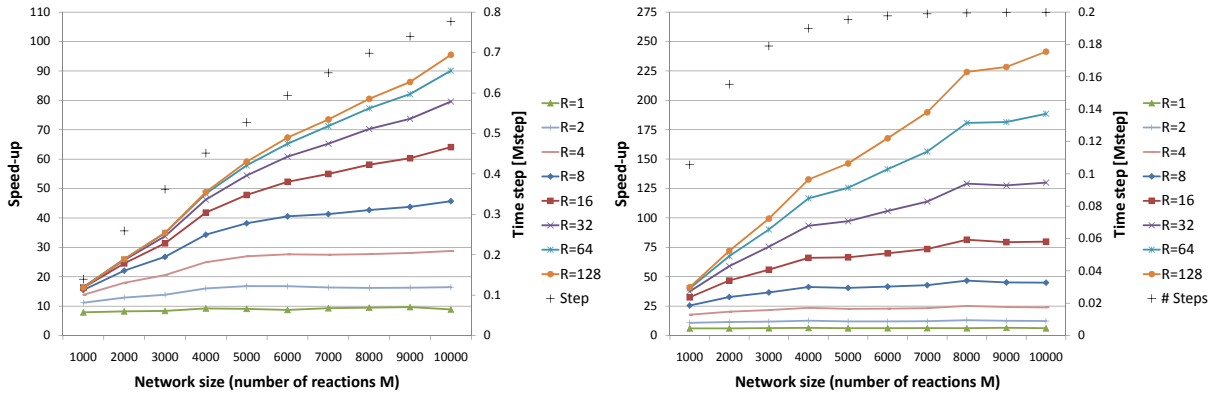
Figure 12 plots the performance gains of the NRM in hardware, defined as $t_{odm,sw}/t_{nrm,hw}$, where we assume a single SPE containing from 1 RU up to 128 RUs at 500 MHz. The measured number of time steps for each network size is further plotted along the secondary axis. We can see in the plots that as the level of parallelism in the hardware (R) and the dependency among reactions (D) increase, speed-ups also increase from 8x up to 96x for networks with a dependency of 10%, and 7x up to 241x for networks with a dependency of 50%. As expected, speed-up enhancements over increases in network size and dependency tend to become prominent as a higher degree of parallelism is involved.

For the τ-leap simulations, we used the beta-1 version of StochKit2 because the latest version implements a modified τ-leap method [20] in which the issue of negative populations is addressed and thus the performance can slightly be degraded in software. Since the beta-1 version halts the simulation or notifies with warnings as a certain number of the occurrences of negative populations is detected, we were able to manage, throughout the experiments, to have zero occurrence of such exception. As in the ODM simulations, we used the same set of network models, and executed the software 11 times on each network type for a simulated time of 15s. During each run, we measured a dataset of the simulation time ($t_{leap,sw}$), the number of leaps (N_{leap}), the number of exact fallback steps (N_{fb}), and the average number of fires for each reaction per leap (i.e., Poisson random number $\mathcal{K}_{mean}$), and extracted the one containing the median of all simulation times. Based on the measurements, we evaluated the simulation time in hardware according to

$$t_{leap,hw} = \frac{(L_{nrm} \times N_{fb}) + (L_{leap} \times N_{leap})}{f}, \tag{52}$$

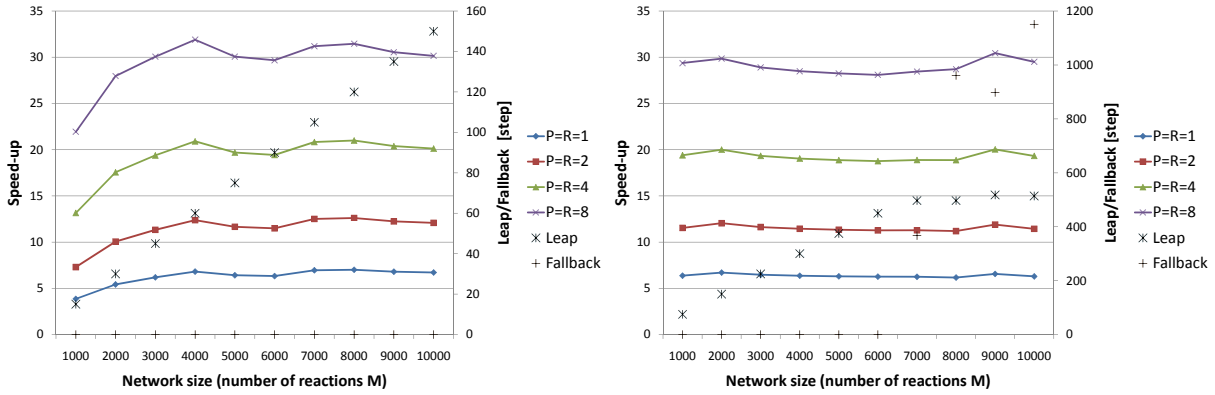
where L_{leap}, given by Equation (48), is the latency in cycle needed to simulate a single leap.

Assuming a single SPE having R RUs in an exact unit and a P degree of parallelism in a leap unit, and all operating at 500 MHz, Figure 13 plots the performance gains of the τ-leap in hardware over the software realizations. We only observe the data with the hardware size of up to $P = R = 8$ as its



(a) D=10% (b) D=50%

Figure 12: *Performance gains of the NRM architecture simulating artificial PCDS networks with an initial condition of* $\{c = 1e\text{-}9, X_{t0} = 1e5\}$ *for a simulated time of 15s, and the associated number of time steps* (1 SPE @500 MHz).



(a) D=10% (b) D=50%

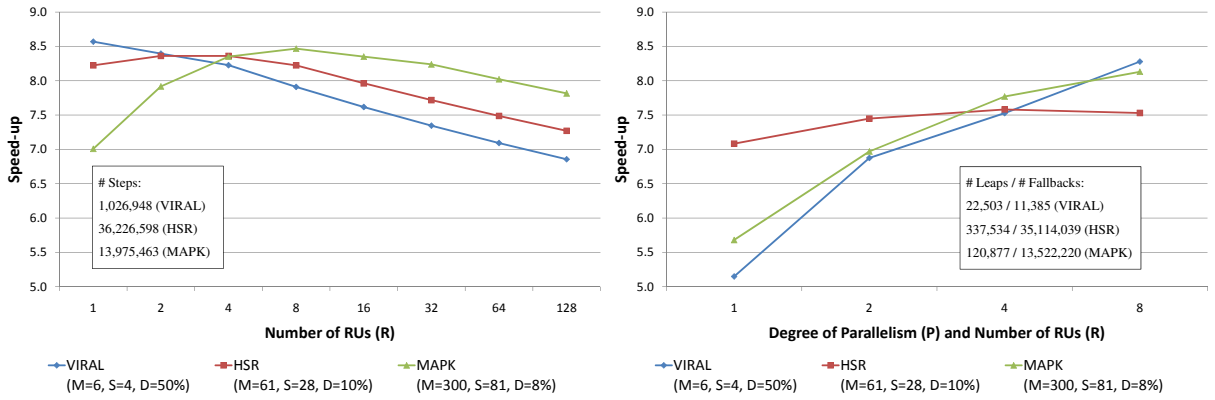
Figure 13: *Performance gains of the* τ*-leap architecture simulating artificial PCDS networks with an initial condition of* $\{c = 1e\text{-}9, X_{t0} = 1e5\}$ *for a simulated time of 15s, and the associated number of leaps and fallbacks* (1 SPE @500 MHz).

area cost is estimated to be comparable to 128 RUs in the exact SSA simulation. For each network size, the number of leaps and the number of fallbacks are also plotted along the secondary axis, and four cases of exact SSA fallback steps are observed as shown in Figure 13(b). As a result, speed-ups range from 4x up to 32x for networks with a dependency of 10%, and from 6x up to 31x for networks with a dependency of 50%. Although neither network size nor dependency has much impact on speed-up enhancements, we still observe improvements in speed-up as a higher degree of parallelism is exploited.

Overall, we have observed that exploiting concurrency in hardware influences the speed-ups of both exact and approximate simulations to varying extents, and thus it is just a matter of, given a frequency constraint, how much degree of parallelism (SPE, R, P) we can accommodate in a single SSSoC to achieve better speed-ups, and thus an additional study of area estimations remains as part of the future work.

4.3.3 Performance Comparison for Real Networks

In addition to the artificial network, we simulated real biological networks including intracellular viral infection [33], heat shock response (HSR) [65], and mitrogen-activated protein kinase (MAPK) cascade [68], each of which has distinct network characteristics in terms of the number of reactions (M),



(a) Exact method (b) τ-leap method

Figure 14: *Performance gains for real network examples* (Simulated time = 300s, 1 SPE @500 MHz).

the number of species (S), and the dependency (D). Figure 14 shows the speed-up results for both exact and τ-leap methods simulating aforementioned models for a biological time of 300s. The number of different types of time steps for each network is additionally shown in the figure. As in the artificial case, we simulated a given network 11 times and took the dataset containing the median simulation time, and using Equations (51) and (52), we evaluated the predicted simulation times of both the NRM and the τ-leap method in hardware.

As a result, speed-ups ranging from 7x to 9x for the NRM and from 5x to 8x for the τ-leap method were achieved under the assumption of utilizing single SPE with a number of sub-units as given by P and R in Figure 14. In Figure 14(a), the reason speed-ups gradually degrade as R increases is due to the fact that there exist crossover points in our throughput plots of various exact methods (see Figure 8) where adding more RUs in an SPE adversely affects on the throughput especially for smaller network sizes and dependency factors.

5 SSSoC Networking

An SSSoC combines multiple SPEs that are interconnected in a network-on-chip (NoC) fashion to exploit coarse-grain SSA parallelism [69]. External ports of SPEs at the borders of the SSSoC are used for communication with the host PC and will allow multiple chips to be combined and connected at the board and cluster level, further increasing scalability. Routers connect SPEs to their neighbors and are configurable in the way they communicate and exchange updates. We will specifically develop SSSoC support for both: (1) independent simulation of the same network in multiple SPEs (parallelism across the simulation); and (2) partitioning of a larger network simulation across multiple SPEs (parallelism across the method). In the process, we will explore novel parallel and distributed SSA variants specifically targeted at SSSoC execution.

Standalone Simulation In a standalone simulation, the same network is pre-loaded into all SPEs. SPEs independently perform simulations with different random seeds, where particular care has to be invested to ensure the independence of generated random numbers [41], e.g. by seeding hardware from random physical events such as thermal noise. Routers are configured to operate locally only and no updates are exchanged with other SPEs. Instead, routers send species information to the host PC, which aggregates and averages data collected over multiple instances. An open issue is the bandwidth needed for communication with the host. Utilizing high-speed host interfaces such as PCI-Express, we except to be able to fully hide host communication latencies in parallel to SPE computations. To address any potential bottlenecks, we will consider SSSoCs that can collect statistics at run time directly in an attached external, on-board memory.

Within the framework of standalone operation, we will investigate tradeoffs between SPE size (number of RUs) in relation to the overall SSSoC area (number of SPEs). Preliminary results (see Figure 8) indicate that for independent Monte-Carlo simulations, a large number of small SPEs is optimal in terms of utilizing chip resources. However, the situation is reversed for simulation of a single large network. By

reconfiguring a network of small SPEs to act as a single, combined large network simulation as described below, we expect the final SSSoC to flexibly support both cases at near-optimal performance. Overall, we will perform design space exploration for various SSSoC operation modes and configurations, including heterogeneous arrays of differently sized SPEs, under area, performance and power considerations.

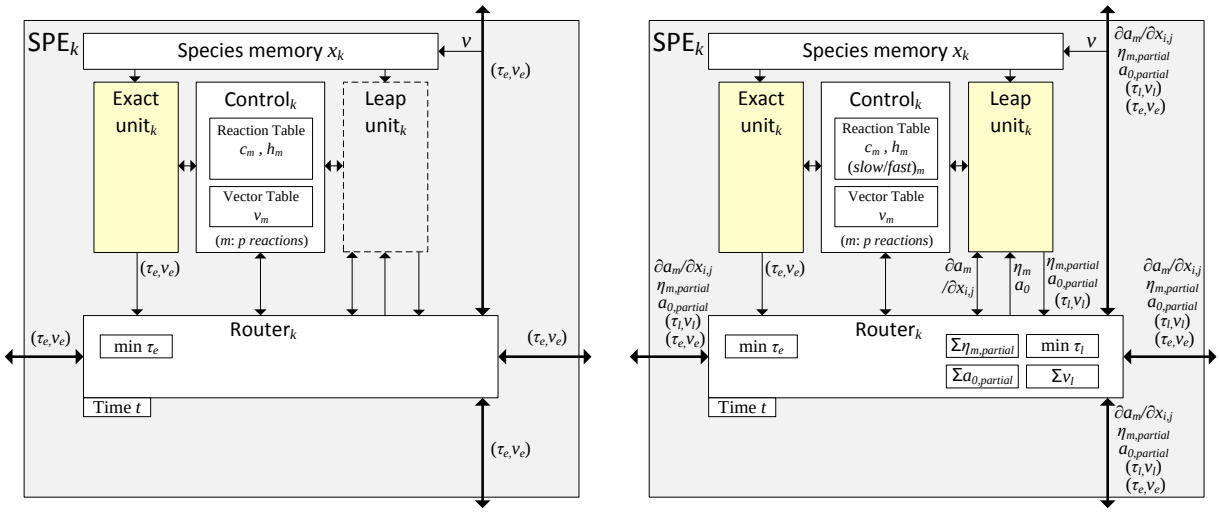
Partitioned Simulation A standalone simulation mode can effectively parallelize many independent runs, but does not help in speeding up individual simulations of large networks. Furthermore, SPEs are limited by practical considerations in the size of their memories and tables, and hence the size of the network they can support. To overcome these limitations, SSSoCs will support a configuration in which multiple SPEs collectively co-simulate a larger network based on either graph [46] or geometric [47, 48] partitioning. Again, key research challenges stem from the question of how to support different modes in a most optimal fashion. Note that in contrast to other architectures, the SSSoC approach will be inherently scalable, e.g. by avoiding the bottleneck of a global shared memory.

In graph-based methods, the model to be simulated is partitioned and distributed such that each SPE processes a subset of reactions assigned to it. Species memories in SPEs thereby hold a copy of all input concentrations needed by assigned reactions[6]. In exact simulations as illustrated in Figure 15(a), a subset of p reactions are partitioned and mapped onto a SPE by programming its reaction and vector tables with stochastic rate constants c_m, reaction types h_m, and stoichiometric vector $\mathcal{V}_m$. While the leap unit being disabled, the exact unit evaluates the next reaction most likely to occur in the nearest time point and sends the associated reaction time τ_e and stoichiometric vector $\mathcal{V}_e$ to the router, which in turn communicates with its adjacent routers to exchange time-stamped update events $(t_e, \mathcal{V}_e)$ across the network. After receiving updates from all other SPEs, routers locally select the event with the smallest time stamp and update local time and species values accordingly.

By contrast, distributed τ-leaping simulations will be more challenging as computation of the time leap interval requires various intermediate data to be available across the whole network, i.e., the data dependency across SPEs can degrade the overall concurrency of the SSSoC with increased overhead in communication. With that said, we may perhaps have to compromise some accuracy by only accounting for the mean value of the propensity change when determining the leap interval, i.e., by discarding the variance evaluation. Figure 15(b) shows a SPE implementing such approximate mode and includes the required data to be communicated through routers to other SPEs. Similar to the exact case, a subset of p reactions are partitioned and mapped onto each SPE with pre-loaded table values. While a SPE in the exact mode independently evaluates the time step τ_e of its assigned subset of reactions, in the approximate mode, it is more efficient for a SPE to evaluate the mean and thus the leap interval cooperatively with other SPEs so as to minimizing the communication overhead. This is due to the fact that calculating the mean of a given reaction channel requires the evaluation of parameters such as propensities and stoichiometric vectors across all reaction channels as well as its own.

Revisiting the process of the τ-leap algorithm in Section 4.2.1, we can partition the calculation of η_m in Equation (33) (i.e., $\eta_m = \sum_{m'=1}^{M} f_{mm'} a_{m'}$) into p separate equations and map them across different SPEs. For example, one SPE processes a partial mean of $\eta_{m,partial,1} = \sum_{m'=1}^{p} f_{mm'} a_{m'}$, another SPE processes a partial mean of $\eta_{m,partial,2} = \sum_{m'=p+1}^{2p} f_{mm'} a_{m'}$, and so on. Notice that as a prerequisite to the evaluation of $f_{mm'}$ given by Equation (32) (i.e., $f_{mm'} = \sum_{i=1}^{N} (\partial a_m / \partial x_i) v_{im'}$), the SPE assigned to the reaction m needs to distribute a pair of $(\partial a_m/\partial x_i,\ \partial a_m/\partial x_j)$ values to all SPEs. The partial mean values calculated by SPEs are then aggregated through routers and thus η_m is obtained. Likewise, routers collect the partial sum of propensities $a_{0,partial}$ from SPEs and aggregate them to get a total sum of propensities a_0 for all reactions in the network. With availability of a_0 and η_m, the leap interval τ_m is calculated according to $\epsilon a_0 / |\eta_m|$ by utilizing the pipeline of the leap unit. In the same manner as described above, the whole SPEs cooperatively process the τ_m values for all reactions in a pipelined fashion, and the leap unit in each SPE finds τ_l as the local minimum of τ_m. Lastly, routers further determine the minimum of τ_l as the final leaping time τ. Upon completion of processing the final τ, routers distribute the value across all SPEs so that each leap unit accordingly generates Poisson random numbers $\mathcal{K}_m$ for those p reactions assigned to its own and provides its local router with a partial sum of stoichiometric vectors (i.e., $\mathcal{V}_l = \sum_m \mathcal{K}_m \mathcal{V}_m$ with m being constrained to the assigned p reactions). Lastly, routers accumulate the stoichiometric vectors $\mathcal{V}_l$ from all SPEs and distribute

[6]This may require duplication of species values in multiple memories.



(a) Exact mode

(b) Approximate mode (τ-leap or hybrid)

Figure 15: *Operational modes for partitioned simulation.*

the final values of τ and $\mathcal{V}$ for updates of local time and species values. Additionally, the exact unit is enabled in case the fallback condition occurs, and in the hybrid mode, both exact and leap units dynamically partition all reactions into slow and fast reactions by setting or clearing the $(slow/fast)_m$ entry in the reaction table.

As can be noted, it is not a trivial task to manage all the synchronization between SPEs as well as to reduce communication overhead. It becomes even challenging when it comes to implementing the partitioned simulation for the algorithm that considers both the mean and the variance. Perhaps, we can optimize necessary communication overhead, e.g., by exchanging intermediate results that can be pre-computed locally in each SPE. Overhead of broadcasts across the mesh network can generally be minimized by employing well-known collective communication patterns [70, 71, 72, 73, 74]. For example, in the exact SSA case, routers only need to forward events that have a lower time stamp than any of the events that have been received already. Furthermore, we can adapt existing partitioning solutions [46] that analyze dependencies between reactions and map closely coupled subnetworks onto the same or neighboring SPEs to reduce overhead. Nevertheless, communication and synchronization in every time step may become a bottleneck. As such, we can envision decoupled approaches that, based on the observation that species values only increment/decrement slightly in each step, aim to overlap communication of species updates with computation of the next iteration. This can significantly increase throughput but will affect the exactness of computations that are performed while species updates are being performed. We are currently in the process of evaluating accuracy and performance of such an approach.

Next to graph-based methods, geometric clustering has been a relatively recent development. In these approaches, spatial information is introduced and systems are described as reaction-diffusion models [75]. For example, in the next subvolume method (NSM) [76], the system is divided into equal subvolumes that are individually simulated by an SSA to determine local reaction or diffusion events. The NSM then repeatedly executes the event with the lowest time stamp, where only those subvolumes affected by the event in the previous step are resimulated. This method has already been successfully parallelized on traditional clusters [48]. We can note that a two-dimensional partitioning in the NSM naturally maps to the 2D SSSoC topology. As such, we can adapt the techniques presented in [48] in such a way that SPEs only need to exchange updates with their immediate neighbors when executing diffusion events[7]. However, a research challenge will be how to optimally synchronize local time stamps and maintain a global time and event

[7]Other diffusion models can be realized by designating entries in the vector tables as external updates for specific other SPEs.

update order across SPEs, requiring implementation of distributed time synchronization protocols [77] within the routers. This will incur overhead for broadcast of time stamps in every iteration. To reduce potential bottlenecks, we can adapt existing temporal decoupling techniques [48]. However, accurate decoupling requires support for rollbacks with associated large memory requirements. We can instead envision a limited, inexact decoupling where routers advance time and update state if all configured neighbors have reached the same t. This effectively limits the window that each SPE can run ahead to one time step, where inaccuracies without a rollback mechanism arise from diffusion updates that arrive earlier than the already computed local event. Again, we will study tradeoffs between accuracy loss and performance gains of such an approach. Results of work on partitioned simulation are expected to lead to novel distributed simulation methods that will also have immediate applicability on traditional cluster or grid architectures. Overall, we envision an SSSoC network in which SPEs and routers can be individually configured for different network simulation modes, different SSA variants and different simulated network topologies.

6 Summary and Conclusion

We have presented an architecture of the stochastic simulation system-on-chip (SSSoC) that is capable of simulating gene regulatory networks and other biochemical reaction systems. The architecture implements both exact and approximate stochastic simulation algorithms (SSAs) via a scalable array of stochastic processing elements (SPEs) communicating through routers placed in a mesh topology.

As part of our research agenda forward, we have studied different types of stochastic simulation algorithms and evaluated how each of them will perform when implemented in custom hardware. For performance comparison among different implementations, we carried out a theoretical analysis of latency and throughput and compared them by varying hardware configurations and network sizes.

For exact SSAs, FRM-based hardware outperforms DM-based hardware to varying extents, and results from our throughput analysis show that, for small network sizes, throughput ranges from 10 Msteps/s to 1280 Msteps/s for a chip with 128 SPEs operating at 500 MHz, and for larger network sizes, the peak throughput reaches up to $64/M$ Gsteps/s with M being the number of reactions in the network. For approximate SSAs, we particularly investigated into a hardware implementation of the enhanced τ-leap method and formulated an analytical expression of the latency for a single leap step.

We performed software simulations on both artificial and real networks, and evaluated the performance gains of our analytical hardware models. For artificial networks, the performance gains varied extensively, and assuming a single SPE with varying levels of concurrency inside, speed-ups of 7x–241x for the exact method and speed-ups of 4x–32x for the τ-leap method were achieved depending on network size and dependency. For real biological networks of intracellular viral infection, HSR, and MAPK cascade, speed-ups of 7x–9x for the exact method and 5x–8x for the τ-leap method were achieved.

Alternatively to the τ-leap in custom hardware, we also sketched a specialized, microcoded processor that is flexible enough to implement current and future approximate methods. Furthermore, we briefly mentioned how hybrid SSAs can be realized by appropriately orchestrating the dedicated exact unit and the programmable approximate unit. Lastly, we explored how multiple SPEs interconnected in a network-on-chip fashion can be leveraged to support two different modes of simulation—standalone simulation and partitioned simulation. Exploiting coarse-grain SSA parallelism, we can either assign the same network in multiple SPEs or partition a large network across multiple SPEs based on some partitioning method such as graph-based method or geometric clustering.

In summary, with the promising results of our approach identified in this report, we have taken the first step toward co-design of algorithms and architectures for a dedicated SSSoC design. We believe the work and some of the envisioned ideas presented in this report can serve as a cornerstone for the future research going forward.

References

[1] F. CRICK, "Central dogma of molecular biology," *Nature*, vol. 227, no. 5258, pp. 561–563, Aug 1970. [Online]. Available: http://dx.doi.org/10.1038/227561a0

[2] L. Hunter, "Life and its molecules: A brief introduction," *AI Magazine*, vol. 25, no. 1, pp. 9–22, 2004.

[3] D. Thieffry, "From global expression data to gene networks," *BioEssays*, vol. 21, no. 11, pp. 895–899, 1999. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1521-1878(199911)21:11⟨895::AID-BIES1⟩3.0.CO;2-F

[4] W. Loomis and P. Sternberg, "Genetic networks," *Science*, vol. 269, no. 5224, p. 649, 1995. [Online]. Available: http://www.sciencemag.org/content/269/5224/649.short

[5] X. Cai and X. Wang, "Stochastic modeling and simulation of gene networks - a review of the state-of-the-art research on stochastic simulations," *Signal Processing Magazine, IEEE*, vol. 24, no. 1, pp. 27 –36, 2007.

[6] R. Albert, "Boolean modeling of genetic regulatory networks," *Lec. Notes Phys. SpringerVerlag Berlin Heidelberg*, vol. 650, pp. 459–481, 2004.

[7] D. T. Gillespie, "A rigorous derivation of the chemical master equation," *Physica A: Statistical Mechanics and its Applications*, vol. 188, no. 1-3, pp. 404–425, Sep. 1992. [Online]. Available: http://dx.doi.org/10.1016/0378-4371(92)90283-V

[8] D. T. Gillespie, "The chemical Langevin equation," *The Journal of Chemical Physics*, vol. 113, no. 1, pp. 297–306, 2000. [Online]. Available: http://link.aip.org/link/?JCP/113/297/1

[9] H. McAdams and A. Arkin, "Stochastic mechanisms in geneexpression," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 94, no. 3, pp. 814–819, 1997. [Online]. Available: http://www.pnas.org/content/94/3/814.abstract

[10] H. de Jong, "Modeling and simulation of genetic regulatory systems: A literature review," *Journal of Computational Biology*, vol. 9, no. 1, pp. 67–103, 2002. [Online]. Available: http://www.liebertonline.com/doi/abs/10.1089/10665270252833208

[11] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977. [Online]. Available: http://pubs.acs.org/doi/abs/10.1021/j100540a008

[12] 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, 2000. [Online]. Available: http://pubs.acs.org/doi/abs/10.1021/jp993732q

[13] T. G. Kurtz, "Strong approximation theorems for density dependent markov chains," *Stochastic Processes and their Applications*, vol. 6, no. 3, pp. 223 – 240, 1978. [Online]. Available: http://www.sciencedirect.com/science/article/B6V1B-45FCSKW-46/2/7684142af0702406b0f7f7b824911b26

[14] Y. Cao, H. Li, and L. Petzold, "Efficient formulation of the stochastic simulation algorithm for chemically reacting systems," *The Journal of Chemical Physics*, vol. 121, no. 9, pp. 4059–4067, 2004. [Online]. Available: http://link.aip.org/link/?JCP/121/4059/1

[15] 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, 1976. [Online]. Available: http://www.sciencedirect.com/science/article/B6WHY-4DD1NC9-CP/2/43ade5f11fb949602b3a2abdbbb29f0e

[16] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova, "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior," *Computational Biology and Chemistry*, vol. 30, no. 1, pp. 39 – 49, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/B73G2-4HNYM88-7/2/e8c9107565958b269665c1a4f2054850

[17] D. T. Gillespie, "Approximate accelerated stochastic simulation of chemically reacting systems," *The Journal of Chemical Physics*, vol. 115, no. 4, pp. 1716–1733, 2001. [Online]. Available: http://link.aip.org/link/?JCP/115/1716/1

[18] D. T. Gillespie and L. R. Petzold, "Improved leap-size selection for accelerated stochastic simulation," *The Journal of Chemical Physics*, vol. 119, no. 16, pp. 8229–8234, 2003. [Online]. Available: http://link.aip.org/link/?JCP/119/8229/1

[19] A. Chatterjee, D. G. Vlachos, and M. A. Katsoulakis, "Binomial distribution based tau-leap

accelerated stochastic simulation," *The Journal of Chemical Physics*, vol. 122, no. 2, p. 024112, 2005. [Online]. Available: http://link.aip.org/link/?JCP/122/024112/1

[20] Y. Cao, D. T. Gillespie, and L. R. Petzold, "Avoiding negative populations in explicit poisson tau-leaping," *The Journal of Chemical Physics*, vol. 123, no. 5, p. 054104, 2005. [Online]. Available: http://link.aip.org/link/?JCP/123/054104/1

[21] Y. Cao, D. T. Gillespie, and L. R. Petzold, "Efficient step size selection for the tau-leaping simulation method," *The Journal of Chemical Physics*, vol. 124, no. 4, p. 044109, 2006. [Online]. Available: http://link.aip.org/link/?JCP/124/044109/1

[22] X. Cai and Z. Xu, "K-leap method for accelerating stochastic simulation of coupled chemical reactions," *The Journal of Chemical Physics*, vol. 126, no. 7, p. 074102, 2007. [Online]. Available: http://link.aip.org/link/?JCP/126/074102/1

[23] J. Puchalka and A. M. Kierzek, "Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks," *Biophysical Journal*, vol. 86, no. 3, pp. 1357 – 1372, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/B94RW-4V2SGSG-8/2/e0713617e41d13a60f3189f0a92c5e14

[24] L. A. Harris and P. Clancy, "A "partitioned leaping" approach for multiscale modeling of chemical reaction dynamics," *The Journal of Chemical Physics*, vol. 125, no. 14, p. 144107, 2006. [Online]. Available: http://link.aip.org/link/?JCP/125/144107/1

[25] Y. Cao, D. Gillespie, and L. Petzold, "Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems," *J. Comput. Phys.*, vol. 206, pp. 395–411, July 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2004.12.014

[26] K.-H. Chiam, C. M. Tan, V. Bhargava, and G. Rajagopal, "Hybrid simulations of stochastic reaction-diffusion processes for modeling intracellular signaling pathways," *Phys. Rev. E*, vol. 74, no. 5, p. 051910, Nov 2006.

[27] Y. Cao, D. T. Gillespie, and L. R. Petzold, "The slow-scale stochastic simulation algorithm," *The Journal of Chemical Physics*, vol. 122, no. 1, p. 014116, 2005. [Online]. Available: http://link.aip.org/link/?JCP/122/014116/1

[28] J. Goutsias, "Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems," *The Journal of Chemical Physics*, vol. 122, no. 18, p. 184102, 2005. [Online]. Available: http://link.aip.org/link/?JCP/122/184102/1

[29] C. V. Rao and A. P. Arkin, "Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm," *The Journal of Chemical Physics*, vol. 118, no. 11, pp. 4999–5010, 2003. [Online]. Available: http://link.aip.org/link/?JCP/118/4999/1

[30] H. Salis and Y. N. Kaznessis, "An equation-free probabilistic steady-state approximation: Dynamic application to the stochastic simulation of biochemical reaction networks," *The Journal of Chemical Physics*, vol. 123, no. 21, p. 214106, 2005. [Online]. Available: http://link.aip.org/link/?JCP/123/214106/1

[31] A. Samant and D. G. Vlachos, "Overcoming stiffness in stochastic simulation stemming from partial equilibrium: A multiscale monte carlo algorithm," *The Journal of Chemical Physics*, vol. 123, no. 14, p. 144114, 2005. [Online]. Available: http://link.aip.org/link/?JCP/123/144114/1

[32] A. Samant, B. Ogunnaike, and D. Vlachos, "A hybrid multiscale monte carlo algorithm (hymsmc) to cope with disparity in time scales and species populations in intracellular networks," *BMC Bioinformatics*, vol. 8, no. 1, p. 175, 2007. [Online]. Available: http://www.biomedcentral.com/1471-2105/8/175

[33] X. Cai and X. Wang, "Stochastic modeling and simulation of gene networks - a review of the state-of-the-art research on stochastic simulations," *Signal Processing Magazine, IEEE*, vol. 24, no. 1, pp. 27 –36, 2007.

[34] E. L. Haseltine and J. B. Rawlings, "Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics," *The Journal of Chemical Physics*, vol. 117, no. 15, pp. 6959–6969, 2002. [Online]. Available: http://link.aip.org/link/?JCP/117/6959/1

[35] R. SRIVASTAVA, L. YOU, J. SUMMERS, and J. YIN, "Stochastic vs. deterministic modeling of intracellular viral kinetics," *Journal of Theoretical Biology*, vol. 218, no. 3, pp. 309 – 321, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/B6WMD-46YBBVF-G/2/b15fc67b5f30555e51c51019c97e92d5

[36] H. Li, Y. Cao, L. R. Petzold, and D. T. Gillespie, "Algorithms and software for stochastic simulation of biochemical reacting systems," *Biotechnology Progress*, vol. 24, no. 1, pp. 56–61, 2008. [Online]. Available: http://dx.doi.org/10.1021/bp070255h

[37] A. M. Kierzek, "STOCKS: STOChastic Kinetic Simulations of biochemical systems with Gillespie algorithm," *Bioinformatics*, vol. 18, no. 3, pp. 470–481, 2002. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/18/3/470.abstract

[38] D. Endy and R. Brent, "Modelling cellular behaviour," *Nature*, vol. 409, no. 6818, pp. 391–395, Jan 2001. [Online]. Available: http://dx.doi.org/10.1038/35053181

[39] J. F. Keane, C. Bradley, and C. Ebeling, "A compiled accelerator for biological cell signaling simulations," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, ser. FPGA '04. New York, NY, USA: ACM, 2004, pp. 233–241. [Online]. Available: http://doi.acm.org/10.1145/968280.968313

[40] H. Salis, V. Sotiropoulos, and Y. Kaznessis, "Multiscale hy3s: Hybrid stochastic simulation for supercomputers," *BMC Bioinformatics*, vol. 7, no. 1, p. 93, 2006. [Online]. Available: http://www.biomedcentral.com/1471-2105/7/93

[41] T. Tian and K. Burrage, "Parallel implementation of stochastic simulation for large-scale cellular processes," in *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, ser. HPCASIA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 621–. [Online]. Available: http://dx.doi.org/10.1109/HPCASIA.2005.67

[42] K. Burrage, P. M. Burrage, S. J. Jeffrey, T. Pickett, R. B. Sidje, and T. Tian, "A grid implementation of chemical kinetic simulation methods in genetic regulation," *Conference on Advanced Computing, Grid Applications and eResearch (APAC03)*, pp. 1–8, 2003.

[43] J. M. McCollum, G. D. Peterson, C. D. Cox, and M. L. Simpson, "Accelerating gene regulatory network modeling using grid-based simulation," *SIMULATION*, vol. 80, no. 4-5, pp. 231–241, 2004. [Online]. Available: http://sim.sagepub.com/content/80/4-5/231.abstract

[44] D. Jenkins and G. Peterson, "GPU Accelerated Stochastic Simulation," *Symposium on Application Accelerators in High Performance Computing (SAAHPC)*, July 2010.

[45] H. Li and L. Petzold, "Efficient Parallelization of the Stochastic Simulation Algorithm for Chemically Reacting Systems On the Graphics Processing Unit," *International Journal of High Performance Computing Applications*, vol. 24, no. 2, pp. 107–116, 2010. [Online]. Available: http://hpc.sagepub.com/content/24/2/107.abstract

[46] N. Jun-qing, Z. Hao-ran, C. Jiu-sheng, M. Meng, and W. Xu-fa, "A distributed-based stochastic simulation algorithm for large biochemical reaction networks," in *Bioinformatics and Biomedical Engineering, 2007. ICBBE 2007. The 1st International Conference on*, 2007, pp. 502 –505.

[47] M. Schwehm, "Parallel stochastic simulation of whole-cell models," *International Conference on Architecture of Computing Systems (ARCS)*, 2002.

[48] M. Jeschke, A. Park, R. Ewald, R. Fujimoto, and A. M. Uhrmacher, "Parallel and distributed spatial simulation of chemical reactions," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 51–59. [Online]. Available: http://dx.doi.org/10.1109/PADS.2008.20

[49] M. Yoshimi, Y. Osana, T. Fukushima, and H. Amano, "Stochastic Simulation for Biochemical Reactions on FPGA," *International Conference on Field Programmable Logic and Application (FPL)*, 2004.

[50] M. Yoshimi, Y. Osana, Y. Iwaoka, Y. Nishikawa, T. Kojima, A. Funahashi, N. Hiroi, Y. Shibata, N. Iwanaga, H. Kitano, and H. Amano, "An FPGA Implementation of High Throughput Stochastic

Simulator for Large-Scale Biochemical Systems," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, 2006, pp. 1 –6.

[51] M. Yoshiini, Y. Iwaoka, Y. Nishikawa, T. Kojima, Y. Osana, Y. Shibata, N. Iwanaga, H. Yamada, H. Kitano, A. Funahashi, N. Hiroi, and H. Amano, "FPGA Implementation of a Data-Driven Stochastic Biochemical Simulator with the Next Reaction Method," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 254 –259.

[52] M. Hucka, A. Finney, B. Bornstein, S. Keating, B. Shapiro, J. Matthews, B. Kovitz, M. Schilstra, A. Funahashi, J. Doyle, and H. Kitano, "Evolving a lingua franca and associated software infrastructure for computational systems biology: the systems biology markup language (sbml) project," *Systems Biology, IEE Proceedings*, vol. 1, no. 1, pp. 41 –53, 2004.

[53] P. Coussy and A. Morawiec, *High-Level Synthesis: from Algorithm to Digital Circuit*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[54] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[55] A. Gerstlauer, C. Haubelt, A. Pimentel, T. Stefanov, D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 10, pp. 1517 –1530, 2009.

[56] S. Paul, N. Jayakumar, and S. Khatri, "A fast hardware approach for approximate, efficient logarithm and antilogarithm computations," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 2, pp. 269 –277, 2009.

[57] S. Oberman and M. Flynn, "Design issues in division and other floating-point operations," *Computers, IEEE Transactions on*, vol. 46, no. 2, pp. 154 –161, Feb. 1997.

[58] K. Yeager, "The mips r10000 superscalar microprocessor," *Micro, IEEE*, vol. 16, no. 2, pp. 28 –41, Apr. 1996.

[59] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim, "Reduced latency ieee floating-point standard adder architectures," *Computer Arithmetic, IEEE Symposium on*, vol. 0, p. 35, 1999.

[60] L.-K. Wang, M. A. Erle, C. Tsen, E. M. Schwarz, and M. J. Schulte, "A survey of hardware designs for decimal arithmetic," *IBM Journal of Research and Development*, vol. 54, no. 2, pp. 8:1 –8:15, 2010.

[61] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[62] S. Obermann and M. Flynn, "Division algorithms and implementations," *Computers, IEEE Transactions on*, vol. 46, no. 8, pp. 833 –854, Aug. 1997.

[63] A. Akkas, "A combined interval and floating-point comparator/selector," in *Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on*, 2002, pp. 208 – 217.

[64]

[65] H. Kurata, H. El-Samad, T.-M. Yi, M. Khammash, and J. Doyle, "Feedback regulation of the heat shock response in e. coli," in *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, vol. 1, 2001, pp. 837 –842 vol.1.

[66] D. E. Knuth, *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. [Online]. Available: http://portal.acm.org/citation.cfm?id=270146

[67] B. J. Bornstein, S. M. Keating, A. Jouraku, and M. Hucka, "Libsbml: an api library for sbml," *Bioinformatics*, vol. 24, no. 6, pp. 880–881, 2008. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/24/6/880.abstract

[68] A. Levchenko, J. Bruck, and P. W. Sternberg, "Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties," *Proceedings of the National Academy of Sciences*, vol. 97, no. 11, pp. 5818–5823, 2000. [Online]. Available: http://www.pnas.org/content/97/11/5818.abstract

[69] P. Ballarini, R. Guido, T. Mazza, and D. Prandi, "Taming the complexity of biological pathways through parallel computing," *Briefings in Bioinformatics*, vol. 10, no. 3, pp. 278–288, 2009. [Online]. Available: http://bib.oxfordjournals.org/content/10/3/278.abstract

[70] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, 2007. [Online]. Available: http://dx.doi.org/10.1002/cpe.1206

[71] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[72] A. Jantsch and H. Tenhunen, Eds., *Networks on chip*. Hingham, MA, USA: Kluwer Academic Publishers, 2003.

[73] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a novel on-chip communication network," in *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, 2004, p. 15.

[74] M. Moadeli, W. Vanderbauwhede, and A. Shahrabi, "Quarc: A novel network-on-chip architecture," in *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, 2008, pp. 705 –712.

[75] K. Takahashi, S. N. V. Arjunan, and M. Tomita, "Space in systems biology of signaling pathways - towards intracellular molecular crowding in silico," *FEBS Letters*, vol. 579, no. 8, pp. 1783 – 1788, 2005, systems Biology. [Online]. Available: http://www.sciencedirect.com/science/article/B6T36-4FG4SNS-4/2/985b8f5ba22f012850d8b21571560681

[76] J. Elf and M. Ehrenberg, "Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases," *Systems Biology, IEE Proceedings*, vol. 1, no. 2, pp. 230 – 236, 2004.

[77] V. K. Garg, *Elements of distributed computing*. New York, NY, USA: John Wiley & Sons, Inc., 2002.