FPGA IMPLEMENTATION OF A DATA-DRIVEN STOCHASTIC BIOCHEMICAL SIMULATOR WITH THE NEXT REACTION METHOD

Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa
Toshinori Kojima, Yasunori Osana
Keio University
Yokohama, Japan
email: bio@am.ics.keio.ac.jp

Akira Funahashi, Noriko Hiroi
Kitano Symbiotic Systems Project,
ERATO-SORST, JST
Tokyo, Japan

Yuichiro Shibata, Naoki Iwanaga, Hideki Yamada
Nagasaki University
Nagasaki, Japan

Hiroaki Kitano
Kitano Symbiotic Systems Project,
ERATO-SORST, JST
Tokyo, Japan

Hideharu Amano
Keio University
Yokohama, Japan

ABSTRACT

This paper introduces a scalable FPGA implementation of a stochastic simulation algorithm (SSA) called the Next Reaction Method. There are some hardware approaches of SSAs that obtained high-throughput on reconfigurable devices such as FPGAs, but these works lacked in scalability. The design of this work can accommodate to the increasing size of target biochemical models, or to make use of increasing capacity of FPGAs. Interconnection network between arithmetic circuits and multiple simulation circuits aims to perform a data-driven multi-threading simulation. Approximately 8 times speedup was obtained compared to an execution on Xeon 2.80GHz.

1. INTRODUCTION

The emergence of an academic field called a systems biology has brought out a new challenge for both computer scientists and biologists, which is to simulate cellular systems using computational resources. Refinements of simulation algorithms and modeling techniques therefore have been continuous attempts in this field.

A stochastic biochemical simulation algorithm (SSA)[1] is a variation of the Monte-Carlo methods, known for its large number of calculation involved. Consequently, the stochastic approach could only be applied to extremely small testing models, and it was difficult to simulate large scale models with complex reaction systems.

A significant performance enhancement of general-purpose microprocessors since late in the '90s were one of the main causes for the stochastic approach to be reappraised as a feasible method to simulate large-scale biochemical models, and many refinements were made to the simulation algorithms. Following these trends, some hardware approaches of stochastic simulators on reconfigurable devices began to appear around 2004[2][3]. These works suggest tangibility to achieve 10 to 100 times performance improvement compared to running stochastic simulation on general-purpose microprocessors, while requiring much lower development cost than dedicated hardware.

Recent biochemical simulation softwares adopt an algorithm called the Next Reaction Method(NRM). It is widely known as an SSA with the finest scalability to the size of target models[4]. This work addresses an FPGA implementation of the NRM, which has not yet been reported.

In this paper, we propose a new framework of implementing NRM by connecting several calculation units with an interconnection network. This design aims to maintain a scalability towards size of biochemical models, while possessing flexibility to each target models and circuit sizes. As a prototype implementation, we designed the interconnection using multiplexers, and investigated performance and scalability of the design.

2. STOCHASTIC BIOCHEMICAL SIMULATION

2.1. Stochastic Simulation Algorithm (SSA)

Gillespie proposed stochastic modeling techniques of chemically reacting systems, the aim of which is to obtain "state" variations of a model from moment to moment[1]. A biochemical model is defined as a list of reactions, and the model's state as a quantity of each species that appears in these reactions. Thus, stochastic simulation algorithm (SSA) is a method to calculate the quantity from time to time. An example of a biochemical model which has N reactions is

1-4244-1060-6/07/$25.00 ©2007 IEEE.

defined as in (1a)-(1c).

$$R_1: \quad S_1 + S_2 \xrightarrow{k_1} S_3 \tag{1a}$$

$$R_2: \quad S_2 + S_3 \xrightarrow{k_2} S_4 \tag{1b}$$

$$\vdots$$

$$R_N: \quad S_N + S_1 \xrightarrow{k_N} S_2 \tag{1c}$$

$S_1, S_2, \cdots$ in equations above represent chemical species, whose numbers are integers. Species in the left-hand side are called "reactants", and ones in the right-hand side are "products". Each reactant in reaction R_j has event probability k_j to bring about a chemical reaction. After initial numbers of each species are given, it is ready to execute a whole process of one *computational cycle*, which is to obtain the time change of the model. However, because the algorithm being a variation of the Monte Carlo method, it requires many trials of computational cycles to obtain accurate results.

Since the Gillespie's First Reaction Method (FRM) and the Direct Method (DM) had been proposed[1], several improved versions of SSA were presented[4][5]. One notable proposal was made in 2000 by Gibson and Bruck, who presented a new algorithm called the Next Reaction Method (NRM). It reduced the time complexity from $O(N)$ of the original version to $O(\log(N))$, while mathematically "proving the statistical equivalence of the simulation results". NRM is applied to representative software biochemical simulators such as E-Cell3[6]. The detail of the algorithms are described in the next section.

2.1.1. First Reaction Method

The idea of SSA is to obtain the state-change of the model through a repetition of a process to select a reaction that is "most likely to occur in the next reaction cycle".

In this algorithm, the following steps are included in one *reaction cycle*. First, the chosen reaction of FRM has the smallest τ_j, predicted time of occurrence among all reactions in the system[1]. Then, a predicted time of occurrence for each reaction (2) is obtained by (3).

$$\overrightarrow{\tau} = (\tau_1, \cdots, \tau_N) \tag{2}$$

$$\tau_j = \ln(1/r)/a_j \tag{3}$$

Value r is a uniform random number between 0 and 1. a_j is called "a propensity", which is a multiple of an event probability k_j and a combination number of all the reactants in R_j.

2.1.2. Next Reaction Method

NRM obtains $\overrightarrow{\tau}$ according to (3) just like FRM, but two new ideas are introduced to reduce time complexity[4]: Indexed Priority Queue(IPQ) and Dependence Graph(DG). The calculation steps in one reaction cycle are as follows. Once $\overrightarrow{\tau}$ is calculated, a set of value τ_j and its reaction ID j are stored in a heap tree called an IPQ. With this modification, NRM only requires a calculation of τ_μ for reaction R_μ that occurred in one reaction cycle without recalculation of whole $\overrightarrow{\tau}$. The root node of IPQ points to the next reaction and its time of occurrence. Afterward, (4) modifies several values in $\overrightarrow{\tau}$.

$$\tau_{j,\text{new}} = a_{j,\text{old}}/a_{j,\text{new}}(\tau_{j,\text{old}} - \tau_\mu) + \tau_\mu \tag{4}$$

Meanwhile, each biochemical reaction may be depend among each other; a change in quantity of certain species due to one biochemical reaction may affect the others. Thus, all predicted time of reactions τ_j that are influenced by current occurrence needs to be modified. In order to clarify these causal relationships, NRM uses a list called a DG. For each reaction, DG enumerates other reactions whose related species' number would be modified. For instance, DG of R_1(1a) is given as (5).

$$DG(R_1) = \{R_2, R_3, R_N\} \tag{5}$$

Finally, the heap tree is updated to maintain its order. The update is necessary whenever value τ_j has been changed with (3) or (4).

Table 1 shows a comparison of operational processes in FRM and NRM in a reaction cycle. In NRM, Gibson and Bruck also introduced a barometer D for representing a complexity of a biochemical model. Assume a value d_j that represents a *dependency* of reaction R_j, which is a total number of reactions that will be updated due to the occurence of R_j. This is equal to a number of executing (3) or (4) when R_j occurs. The barometer is an average number $D(\ll N)$ of d_j in the model. Number of calculation increases according to the model size for FRM, while NRM is proportional to $\log(N)$. This table indicates higher scalability of NRM compared to FRM.

In this paper, the NRM circuit design was evaluated with a model D4S($D = 4$ System) defined in (1a)-(1c) by changing its N.

Table 1. Number of calculations and time complexity of FRM and NRM

	FRM	NRM
Eq. 3	N	1
Eq. 4	0	$d_\mu - 1$
updating IPQ	0	d_μ
Order	$O(N)$	$O(\log(N))$

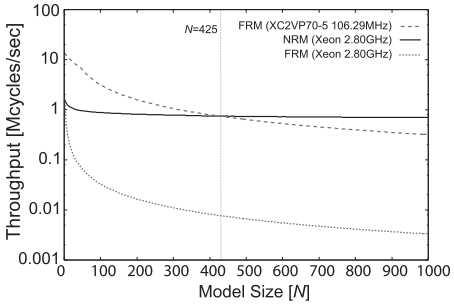

Fig. 1. Drop-off of the throughput versus model size

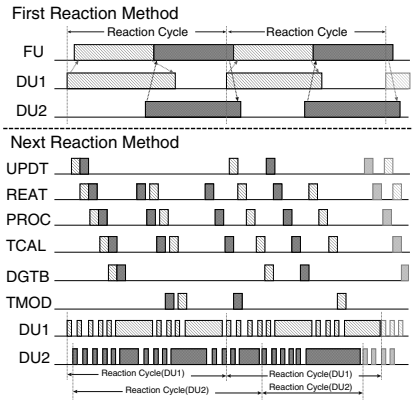

Fig. 2. Usage of calculation units for FRM and NRM

2.2. Stochastic Biochemical Simulators on FPGA

2.2.1. Related work

Several challenges have been made to design a stochastic biochemical simulator on FPGA since 2004. Keane *et al.* and Salwinski *et al.* both successfully achieved approximately 20 times speedup compared to general-purpose microprocessors [2][7]. However, both of their works are based on more approximated version of Gillespie's algorithm, and calculation steps were also simplified. For example, they convert floating-point to integer values to perform high speed computation. Thus, simulations on these platforms may require many more computational cycles to obtain the same level of accuracy with software-based simulators.

2.2.2. The FRM implementation on an FPGA

We have been implementing and evaluating stochastic biochemical simulators since 2004 [8][3]. In 2006, a circuit was designed with two simulation threads that time-share one single-precision floating point computational unit. Pipeline of the computational unit can receive consecutive input data to achieve high throughput [3]. And without using approximated stochastic algorithm, this implementation achieved more than 80 times speedup compared to execution on Xeon 2.80GHz by running six threads in parallel to simulate a model with 1000 reactions ($N = 1000$).

Cao *et al.* has already compared computational time of NRM and DM, which is known to be more computationally-efficient than FRM[5]. To investigate more detail, we wrote execution programs of FRM and NRM in C++, and evaluated throughput versus models size for both algorithms using D4S. Here, we define a term "throughput" as an execution time of one reaction cycle. The results are shown in Fig.1, together with the FPGA execution result of FRM. According to these results, throughput degradation of the FPGA implementation of FRM is more prominent than NRM execution on Xeon as the model size increases. Advantage of the two turns back at a point of $N = 425$, and NRM move out ahead by about three times at $N = 1000$. This implies that calculation cost of FRM on an FPGA is purely disadvantageous compared NRM on microprocessors, considering that NRM is proved to produce equivalent results with FRM.

Discussions above suggested the possibility of FPGA implementation of NRM to achieve higher throughput for stochastic biochemical simulations. Operating frequency of FPGAs are generally dozen times lower than that of microprocessors, therefore it is difficult to obtain high throughput with a single task execution without degrading numerical precision. Consequently, we followed the policy of running multiple threads as in our FRM implementation, by arranging multiple thread execution circuits with small number of computational units. In addition to achieve high throughput, we aim to investigate a scalable design toward size of a target model, or number of threads running in parallel.

3. DESIGN CONCEPT

A schematic figure of calculation steps in FRM and NRM per a reaction cycle is illustrated in Fig.2. Unlike FRM that calculates $\overrightarrow{\tau}$ by repetitively accessing the same calculation for N times, NRM performs several arithmetic operations and update of heap trees according to occurrences of reactions. To carry out a seamless operation, a circuit was designed to perform a data-driven multi-threading simulation unlike previous works with statically scheduled data-flow.

Calculation units for NRM generally occupy a large circuit area due to having logarithmic arithmetic units. However, number of logarithmic calculation in one cycle is very small. Thus, it is effective to share the unit among multiple simulation threads in terms of area efficiency and throughput.

Consequently, the modules are divided into two groups. The first group of modules need to be prepared for each simulation thread, while the other group is shared among the multiple threads. These would be connected via some communication networks. A schematic figure of calculation

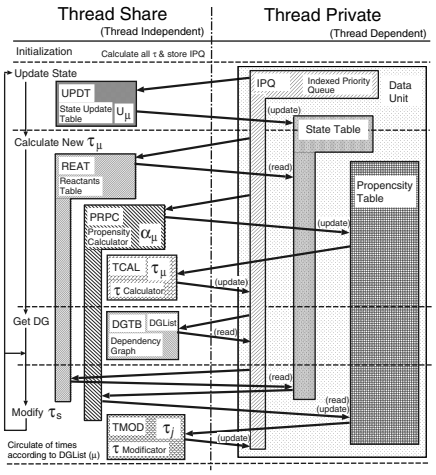

Fig. 3. Computational flow of the Next Reaction Method

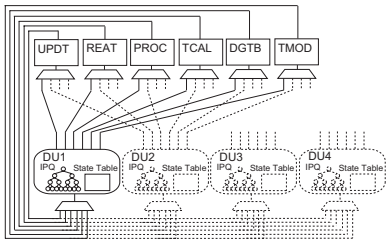

Fig. 4. Connection Diagram of the NRM circuit

steps per simulation cycle of NRM is shown in Fig.3. A simulation cycle starts from an "Update State" stage, followed by the next three stages. Each step should be proceeded sequentially as shown in Fig.3. "modify-τ" stage is repeated for $d_\mu - 1$ times per cycle, so calculation steps differ among reactions R_μ that occurred. Three blocks on the right-hand side of Fig.3 are arrays to store variables and intermediate data of the simulation, so these blocks should be prepared for each simulation thread. A set of the three blocks is called a "Thread Private Unit (TPU)" in the following sections. On the other hand, six blocks in the left-hand side of the figure are units that perform calculation based on data of TPUs, and retrieve the result. Since all simulation threads can use the same calculation units, they are called "Thread Share Units (TSUs)" in the following sections.

Fig.4 illustrates a configuration with four sets of TPUs each of which is connected to TSU with a multiplexer. By selecting an appropriate interconnection network and number of TPUs, it becomes tangible to configure a circuit design according to FPGA area and model size. This paper evaluates performance and area with a prototype implementation of the NRM circuit, followed by an investigation of scalability through an FPGA-implementation of NRM circuit with variable number of TPUs and TSU sets connected with a multiplexer.

4. IMPLEMENTATION

4.1. TPU and TSU

A prototype of TPU and TSU was implemented based on the design described in Section 3. All modules were written in Verilog-HDL, and synthesis, placement and routing were done by Xilinx's ISE8.2i.

Target device of the design is Virtex-II Pro (XC2VP70-6) on a ReCSiP-2 board[3], a biochemical simulation oriented platform. Single-precision floating-point arithmetic units were from Xilinx's LogiCORE floating-point. As a storage for variables in each unit, BlockRAMs on the Virtex-II Pro were utilized, whose entry size is 32bit $\times 1024$ words. Maximum number of biochemical reactions supported by this implementation is 1024, which is sufficient for existing stochastic models.

Table 2 is a rough estimate of area and operating frequency of each unit. Area of TCAL in TSU is large, because it owns a logarithmic arithmetic unit in order to calculate (3). Random numbers required in the same equation are generated with M-sequence random number generator, and logarithmic values are obtained with second order interpolation. PRPC and TMOD are calculation units for propensity and τ_j modification with (4). REAT, UPDT and DGTB are tables for storing constant values: species IDs of reactants in each reaction, state update vectors, and a Dependency Graph. Components of a TSU have pipeline pitches whose size are shown as "flit" in Table 2.

A TPU has three arrays in BlockRAMs, as shown in Fig.3. It also has controllers to communicate with each TSU based on the algorithm of NRM. There are two controllers in the TPU for external and internal use. The external one handles data transfer between TSUs, and the internal one is in charge of updating IPQ and reading data required in each calculation. It should be noted that current implementation does not perform a continuous data transfer, that is, once a data sending request is accepted, the next request is not issued before retrieving the calculation result. This data transfer method will make latency longer, but it does not require any output buffers so that the area is kept small.

Table 2. Resource utilization and operating frequency of TPU and TSU

	TPU	TSU				
	DU	REAT	TCAL	PRPC	UPDT& DGTB	TMOD
Slices	640	0	3894	961	0	169
BRAMs	8	2	6	2	3	4
Mult	0	0	13	8	0	0
Freq.	139	122	132	132	115	126
In. width	64	10	64	74	10	64
flit	1	1	1	1	1	2
Latency	-	1	21	11	1	27

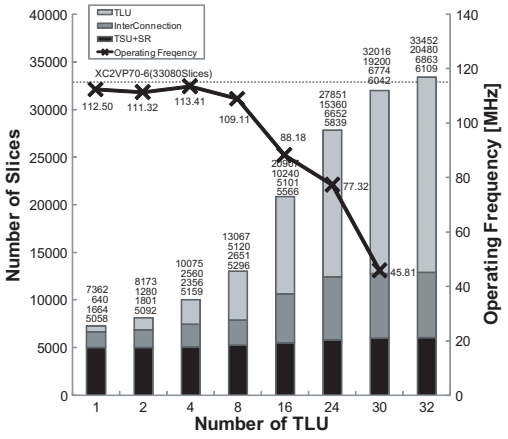

Fig. 5. Area and operating frequency

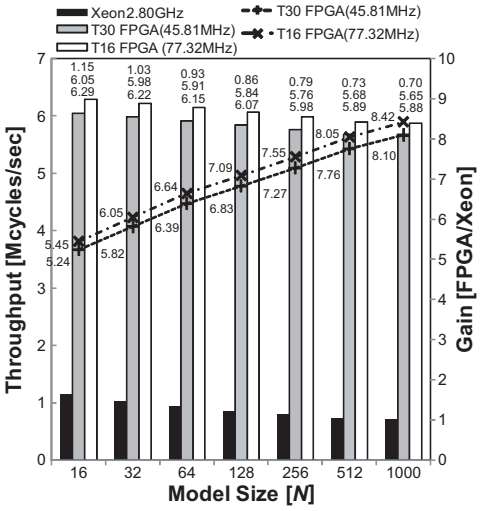

Fig. 6. Throughput and its gain

4.2. Interconnect

An interconnection network between TPUs and TSUs can be categorized into two components: an input network and an output network to/from the TSU. The input network especially requires a mechanism to appropriately transfer sending requests from multiple TPUs to the corresponding TSUs. For this work, T-input multiplexers (MUX) are adopted as the interconnect, where T represents a number of TPUs. Each TSU has its own MUX, data bandwidth and functions of which are correspondent to the TSU.

Each MUX receives data transfer request (REQ) signal and data from multiple TPUs, then returns acknowledge (ACK) signal to a TPU, and outputs data to the output line. Each MUX has a buffer to store one flit of input data. Thus, when an MUX is connected to a TSU whose transfer data size is more than two flits, an acknowledged TPU can dominate the MUX for the equal number of clock cycles with the number of flits. Other TPUs in need to use the same TSU wait until the end of current data transfer. Current implementation has six TSUs, the input network of which are connected with six different MUXs. Table 2 shows data size and number of flits for each MUX. This restriction mechanism of the input networks forms up the output data stream, so there is no case when multiple TSUs try to send results to one TPU. Thus, no arbitration mechanism is required in the output networks.

Area and latency of MUXs increase as T becomes larger. Their interrelationship will be discussed in Section 5.

5. EVALUATION

5.1. Area evaluation

Fig.5 shows an area and maximum operating frequency with different numbers of TPU (T). Current implementation can accommodate up to 30 TPUs on Xilinx's XC2VP70-6. From the figure, number of used BlockRAMs increases by 8 as number of T is incremented. Increasing rate of the BlockRAM (2.43%) is larger than that of the area (1.87%), but this is not the main reason of limiting the maximum number of T, since TSUs dominate many slices in the target FPGA.

Slices of TPUs increase linearly as T becomes large. This causes the increase of slices for MUXs, but its rate is gradual compared to that of TPUs since they own an output buffer only for one flit data. However, fan-out to the output buffer of MUXs also increases, which degrades maximum operating frequency.

Fig.5 also indicates that critical paths lies in a 64-bit integer-float converting module when $T \leq 8$, and in a TPU-TMOD MUX when $T \geq 16$.

5.2. Performance evaluation

Fig.6 shows throughput measured through RTL simulations for different model sizes between $N = 16$ and $N = 1000$ of D4S model (as defined in (1a)-(1c)). Gain in throughput and performance based on execution on Xeon 2.80GHz are also evaluated.

Execution of 16 and 30 threads achieved approximately 5.2 to 8.4 times throughput compared to that of Xeon. The result also indicates an advantage of the design for simulating models with larger N. In case when D is common and N differs, the difference of calculation time is only caused by the update of a heap tree. Since branch penalties on Xeon microprocessor is relatively large because it takes much longer time for updating the heap tree, while the TPU completes data reading, comparison and exchange in only 3 clock cycles. Critical path of simulation circuits when $T \geq 16$ lies in an MUX between TPU and TSU. Consequently, the throughput of $T = 30$ is below that of $T = 16$.

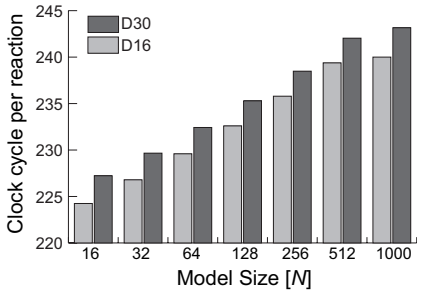

Fig. 7. Average clock cycles per reaction cycle versus number of TPU

Based on the analysis above, an improved interconnection is expected to suppress the degradation of the operating frequency. One idea is to replace the multiplexers with a hierarchical bus structure.

Data flow rate within the interconnect would significantly affect the throughput in such structure, therefore we also evaluated an average clock cycles per one reaction cycle for different model sizes N in case of $T = 16$ and $T = 30$. The result is shown in Fig.7. The figure tells that the increase of the clock cycles were small and follows $O(\log(N))$. In case of D4S model, the number of calculation and update of heap tree is same among all model sizes. Thus, Fig.7 corresponds to the difference in time required to update the heap tree in case of the same T and different N, and difference of waiting time to the multiplexer in case of the same N and different T. Clock cycles are larger for $T = 30$ than $T = 16$ in any case as small as 5 clock cycles. This is because data is not frequently sent over the interconnection, and the length is 2 flits at most. Thus, replacement of the interconnect from multiplexers to the hierarchical bus structure would possibly minimize throughput degradation due to the increase of waiting time in data transfer.

These evaluation results indicate scalability of the current NRM circuit design to the increasing model size, especially in contrast with execution on Xeon processor. Feasibility of achieving higher throughput is also suggested by modifying the structure of the interconnection network.

6. CONCLUSION AND FUTURE WORK

This paper described an FPGA-based design of a biochemical simulation circuit for performing a stochastic simulation based on the Next Reaction Method, and evaluated its prototype implementation. The circuit was designed to achieve high throughput by allowing multiple simulation threads running in parallel. Every module in the circuit is categorized into a group that should be prepared for each simulation thread and a group that are shared among multiple threads. Currently, their interconnection is designed with a multiplexer, and approximately 5.2 to 8.4 times higher throughput was obtained compared to execution on Xeon 2.80 GHz. Some investigation results are given to suggest a feasibility of a higher throughput design by selecting appropriate interconnection network.

As a future work, we are planning to improve throughput based on the current structure. Methodology of data transfer will also be modified from current ping-pong transmission to a mechanism that tolerates continuous requests. Furthermore, we will analyze utilization of each arithmetic unit and data transfer rate with several biochemical models, and carry out more investigation of a suitable design for the interconnection network with higher throughput and scalability.

ACKNOWLEDGMENT

This work is supported by VLSI Design and Education Center(VDEC), The University of Tokyo with the collaboration with Cadence Corporation.

7. REFERENCES

[1] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of Computational Physics*, vol. 22, pp. 403–434, 1976.

[2] J. F. Keane, C. Bradley, and C. Ebeling, "A compiled accelerator for biological cell signaling simulations," in *The 12th International Symposium on Field-Programmable Gate Arrays(FPGA)*, Feb. 2004, pp. 233–241.

[3] 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 *The 16th International Conference on Field Programmable Logic and Applications*, Aug. 2006, pp. 227–232.

[4] M. A.Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *Journal of Physical Chemistry A*, vol. 104, no. 9, pp. 1876–1889, 2000.

[5] Y. Cao, H. Li, and L. Petzold, "Efficient formulation of the stochastic simulation algorithm for chemically recting systems," *Journal of Chemical Physics*, vol. 121, no. 9, pp. 4059–4067, 2004.

[6] K. Takahashi, K. Yugi, K. Hashimoto, Y. Yamada, C. J. F. Pickett, and M. Tomita, "A multi-algorithm, multi-timescale method for cell simulation," *Bioinformatics*, vol. 20, no. 4, pp. 538–546, Mar. 2004.

[7] L. Salwinski and D. Eisenberg, "In silico simulation of biological network dynamics," *Nature Biotechnology*, vol. 22, no. 8, pp. 1017–1019, Aug. 2004.

[8] M. Yoshimi, Y. Osana, T. Fukushima, and H. Amano, "Stochastic simulation for biochemical reactions on FPGA," in *The 14th International Conference on Field Programmable Logic and Applications*, ser. Lecture Notes in Computer Science, vol. 3203. Springer, Aug. 2004, pp. 105–114.