A Study on Monte-Carlo Biochemical Simulation for Field-Programmable Gate Arrays

Masato Yoshimi

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Science for Open and Environmental Systems
Graduate School of Science and Technology
Keio University

March 2009

Preface

Since 1970s, many biologists and computer scientists have been studying various approach methodologies to simulate chain chemical reactions inside living cells using computers. These studies were supported by rapid progress of computer performance and operational cost. Stochastic simulation method has become increasingly important for various study fields in the systems biology, which is a strategic research area that aims to analyze life processes as a system through simulations of biochemical models. Life processes often exhibit stochastic phenomena. Stochastic Biochemical Simulation Algorithm, which is abbreviated as SSA, is a method to calculate time changes of molecular numbers in a stochastic biochemical model. SSA requires vast number of repetitive computation, because it originates in a Monte-Carlo simulation methodology. Therefore, various algorithms have been developed to reduce the amount of calculation, and there are all many ongoing studies on the development of new computer system to achieve high-throughput.

Since 2000s, the baseline performance improvement procedure for microprocessor evolved from acceleration with high operational frequency to parallel operation using multi-core processor. Under such circumstance, a new type of computational systems is being focused for allocating a part of scientific operations to dedicated hardware in order to achieve both low-cost and high-performance. For example, Graphical Processing Units (GPUs) and other many-core processors in equipped in large-scale PC-clusters for the aim of reducing operational cost. Especially, a Field Programmable Gate Array is anticipated as an alternative for small- to middle-class PC-clusters with less than dozens of PCs, and various researches are now in progress. Dedicated systems with FPGAs allow high-speed or high-throughput computation by well-scheduled dataflow that fills up arithmetic pipelines solving the algorithm. However, just like other dedicated hardware, high-performance is achieved with FPGAs when algorithms have high loop-level parallelism, and conditional branches in the loop may degrade performance drastically. Performance improvement obtained by use of an FPGA greatly varies by the characteristics of target algorithm, because FPGAs have strict limitation of data size, dataflow and parallelism to achieve high throughput.

This work studies implementation and evaluation of two different SSAs on an FPGA, which aims to achieve high-performance compared to execution on microprocessors. The goal of this work is to clarify the relationship between algorithm, hardware architecture, and performance according to data size based on the evaluation results.

First SSA is called First Reaction Method (FRM). Processes in the algorithm is simple, and has high degree of loop- and data-level parallelism. Thus, the hardware design was fixed, and data flow was statically scheduled to enhance performance by consecutively injecting data into deep pipelines of floating point units. Arithmetic operations were configured faithfully to the original algorithm by single-precision floating-point data. We validated that the implemented hardware can treat small-scale models well. The design achieved more than 80-fold throughput compared to software execution on Xeon 2.80 GHz, with large-scale biochemical systems for up to 1023 reactions.

Second SSA is called Next Reaction Method (NRM). The algorithm adopts two distinctive data structures: a binary tree in an Indexed Priority Que (IPQ) and a Dependency Graph (DG). They are used for improving computational efficiency over FRM for large-scale models, but processing

time dynamically varies because of the nature of their data structures. Main concept of the design for executing NRM was to allow multi-thread execution. However, since it is highly difficult to achieve high throughput by naively exploiting data-level parallelism, the design adopted a data-driven methodology. The modules in implemented hardware are linked with an interconnection network. By modifying the network among modules, hardware design can be flexibly tuned to perform well on the target FPGA device. We found that the bus-based interconnection network achieves reasonable performance. It was evaluated with different number of threads, and through the result analysis we studied a methodology to reduce waiting time. The design achieved approximately 4.2 to 5.4 times higher throughput compared to execution on Core 2 Quad Q6600 2.40GHz.

And finally, the performance of two implementations for various size of biochemical model is discussed to lead to future development of scientific application using FPGAs.

Acknowledgments

I would like to thank all the people who supported me to accomplish this thesis.

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Hideharu Amano for providing me this precious study opportunity since I was a bachelor student in his laboratory. He has always given me many beneficial advices and suggestions which were the stimulus to pursue my research. His great experience guided me to enlarge my knowledge on computer architecture fields.

I would like to thank all of my doctoral committee members, Professor Kotaro Oka, Associate Professor Michita Imai, and Associate Professor Hiroaki Saito for their careful reviews and comments to my thesis. I also would like to thank Professor Yoshikazu Yamamoto and Associate Professor Nobuyuki Yamasaki for their keen technical comments on this work.

This work is supported by countless researchers around the world.

First of all, I received generous support from members of our research group. I especially would like to appreciate my seniors, Dr. Akira Funahashi (Keio University) and Associate Professor Yuichiro Shibata (Nagasaki University). I also thank to members of Shibata Laboratory, Naoki Iwanaga, Hideki Yamada and Tomoya Ishimori, and our common co-researchers, Dr. Noriko Hiroi and Dr. Hiroaki Kitano.

I would like to thank Dr. Yasunori Osana (Seikei University). He is my trailblazer as a researcher in cross-disciplinary studies.

I am grateful to Yuri Nishikawa for her careful English support and vivid inspirations. Discussions with her on multi- and many-core processing hardware have been beneficial for this work.

I also thank to members of our research group (BIO and ASAP), Tomonori Fukushima (Toshiba Semiconductor Company), Shizuto Fukuda (SCEI), Yow Iwaoka (HITACHI), Toshinori Kojima (Panasonic), Hirokazu Morishita and Akihiro -Vegeta- Shitara. I received daily inspirations and encouragements from the profitable discussions with them.

I owe my deep thanks to assistant Professor Michihiro Koibuchi (National Institute of Informatics) provided me with many insightful technical discussions and comments.

I would like to appreciate my seniors, Dr. Akira -Yomi- Tsuji (NEC), Dr. Ko-nosuke Watanabe, Dr. Tomohiro -Terry- Otsuka, and member of PDARCH, Tomotaka -Amuro- Miyashiro (Sony). I also thank to Dr. Yasuki -Chuckey- Tanabe, Yoshinori Adachi (Toshiba Semiconductor Company), Dr. Satoshi -TSUTSU- Tsutsumi (HITACHI) and Dr. Yohei -GWA- Hasegawa (Toshiba).

I would like to extend many thanks to my colleagues: especially Dr. Hiroki Matsutani and Dr. Wang Daihan. I also greatly thank Dr. Vasutan Tunbunheng for sharing pleasures and sorrows with me as research associates of GCOE program. I wish to thank all the members of Amano Laboratory for their daily supports during my laboratory life.

I also wish to thank all the members of Anzai-Imai Laboratory and Yamasaki Laboratory, especially Kenta Ishii, Satoru Satake and Hirotaka Osawa.

A part of this research was supported by Japan Society for the Promotion of Science (JSPS) for the research fellowship (DC1). The project was also supported by Grant Number R01EB007511 from the National Institute Of Biomedical Imaging And Bioengineering. I would like to thank VLSI Design and Education Center (VDEC), the University of Tokyo for providing CAD tool supports. The CAD tools which are used in this work have been supported by VDEC in collaboration with Synopsys, Inc., Cadence Design Systems, Inc., and Mentor Graphics, Inc.

Finally, I am grateful to three women in my family. First of all, I could not sustain this research without my grandmother Eiko Kawakami's unselfish willingness to provide me with visible and invisible support. My senior sister Chisato Yoshimi supported and encouraged me to make improvements of my unsound and uncertain life.

This thesis is a tribute to my mother, Setsuko Yoshimi, who passed away in 2007. I hereby pledge to her that I will be a passionate educator and a genuine researcher to follow in her footsteps.

Masato Yoshimi

Yokohama, Japan
AM 02:23 13th February 2009

Contents

List of Tables

List of Figures

Abbreviations and Acronyms

E.coli *Escherichia coli*
ASIC Application Specific Integrated Circuit
DG Dependency Graph
DM Direct Method
DNA Deoxyribonucleic Acid
FPGA Field-Programmable Gate Array
FRM First Reaction Method
HSR Heat-Shock Response
IPQ Indexed Priority Queue
LCS Lenear Chain System
NRM Next Reaction Method
ODE Ordinary Differential Equations
ODM Optimized Direct Method
ReCSiP Reconfigurable Cell Simulation Platform
SBML Systems Biology Markup Language
SSA Stochastic Biochemical Simulation Algorithm
STOCHKIT Stochastic Simulation Kit
STOCKS STOChastic Kinetic Simulation
TIS Totally Independent System

Chapter 1

Introduction

1.1 Motivation

Since 1970s, there have been numerous attempts to replicate a sequence of chemical reactions using computers. It motivated various studies to calculate time-evolution of biochemical models which involve different types of chemical species reacting in a different form.

With rapid performance improvement and cost-reduction of computers, biologists acquired an easy access to computational resources in order to process of quantitative data gathered from experiment such as genome sequence of various life activities and metabolizing systems in a cell.

In 2000s, the Systems Biology became an active area of research as an interdisciplinary region of biology and computer science. Its aim is to create mathematical models of biochemical phenomena, which would encourage and stimulate the systematical understanding to them. Further advance of the study is anticipated by establishment of standardized environments and languages for computational modeling of biochemical systems, such as SBML [1]. Biochemical experiments not only require time and expense, but also involve ethical concerns. Therefore, biochemical simulation environment is both a necessity and challenge to biochemical researchers.

Varieties of software-based biochemical simulators have been exploited during the past few decades: from small-scale kinetic simulators such KINSIM [2] and GEPASI [3], to whole-cell simulators represented by E-CELL [4] and The Virtual Cell [5, 6, 7]. They allow users to explore the dynamics of the system by tracing trajectories of the metabolite concentrations in time-series, and also to generate a sequence of simulations with different combinations of kinetic parameters to seek for optimal sets of experimental results.

While traditional ordinary-differential equation (ODE) based simulation mentioned above is used for metabolic simulations, stochastic modeling technique for chemical reactions was developed by Gillespie in 1976 [8]. It is called Stochastic biochemical Simulation Algorithm (which is abbreviated as SSA), and various improvements on the algorithm are being studied. From the late 1990s, software-based simulators applying stochastic approach have been developed. For example, STOCKS [9] simulates cell growth and its division, and STOCHSIM [10] can deal with the conformational change of molecules.

However software-based simulators show serious performance limitations, for their platform being general microprocessors. For example, parameter estimation is a task with serious calculation cost that even a parallel system of high-throughput microprocessors such as PC/WS clusters and grid computers have deficiency in their performance at the present day. Performance of these computational resources may be also degraded as the scale of analysis objects increase, for intensive communication and synchronization due to parallel dependencies across species that commit to numerous reactions in the model. Moreover, as stochastic biochemical simulation is a kind of Monte-Carlo method, it requires vast time of circulation of a simulation trial to get the statistically-significant

number of result. While studies for both algorithm and simulation environment make advance in acceleration simulation, other acceleration techniques are also desired.

Defects of large-scale computational resources do not only lie in performance. They are also unrealistic for individual biologists to obtain in terms of their system size, cost, and power consumption. Dedicated hardware utilizing ASIC is a possible solution for achieving high-throughput with small hardware size, but it is also accompanied with extremely high development cost and poor capability toward variousness of biochemical reactions. Although it is difficult to accommodate its progressively and flexibility by dedicated hardware, biochemical simulation involves various levels of high parallelism. While reconfigurable systems such as FPGA can process by hardware operation directly, its hardware module is loaded by other modules generated for the target algorithm. Another advantage is an initial cost. As dedicated hardware used in scientific application is not produced in large amount, FPGA may be a reasonable solution to solve such problems. Especially, SSA is a challenging application to accelerate by specific hardware like FPGA and other multi- or many-core hardware.

Against such background, our research group has focused on a development of an FPGA-based biochemical simulator called ReCSiP [11, 12, 13]. In this thesis, stochastic biochemical simulators are implemented and evaluated. The main objective of this study is to show that the performance of FPGA obtains some-fold higher performance compared to recent microprocessors. Additionally, approaches to solve previous problems by exploiting parallelism are discussed based on quantitative evaluation results.

1.2 Objective and Contribution

In this thesis, two types of hardware for stochastic biochemical simulation are implemented and evaluated. The preliminary goals of implemented hardware are that: (1) performance should outperform severalfold throughput compared to the execution on recent microprocessor for simulation of practical stochastic biochemical model, (2) they improve performance of related works by other research group and our previous works. Even though their implementation achieved high performance, they expressed several problems that made them off from practical use.

Representative requirements for simulator hardware module are as follows:

1. It runs exact same algorithm with original SSAs using floating-point arithmetics,
2. It has a capability to simulate large scale biochemical model,
3. It exploits effective utilization for logic resource in FPGA.

To attend these requirements, two SSAs, First Reaction Method (FRM) and Next Reaction Method (NRM), are implemented and evaluated on an FPGA. Both of them have various levels of parallelism, and they are exploited on hardware modules. In processes in the algorithm is simple, and has high degree of loop- and data-level parallelism. On the other hand, NRM adopts two distinctive data structures: a binary tree in an Indexed Priority Que (IPQ) and a Dependency Graph (DG) to reduce time complexity of FRM. In the case of NRM, thread-level parallelism is only a potential to improve throughput.

1.3 Thesis Organization

This thesis organization is illustrated in Fig. 1.1. Chapter 2 of this thesis explains a fundamental stochastic modeling technique of chemical reacting systems, simulation algorithms, and actual software simulators. Chapter 3 describes the structure of an FPGA, reconfigurable systems for scientific

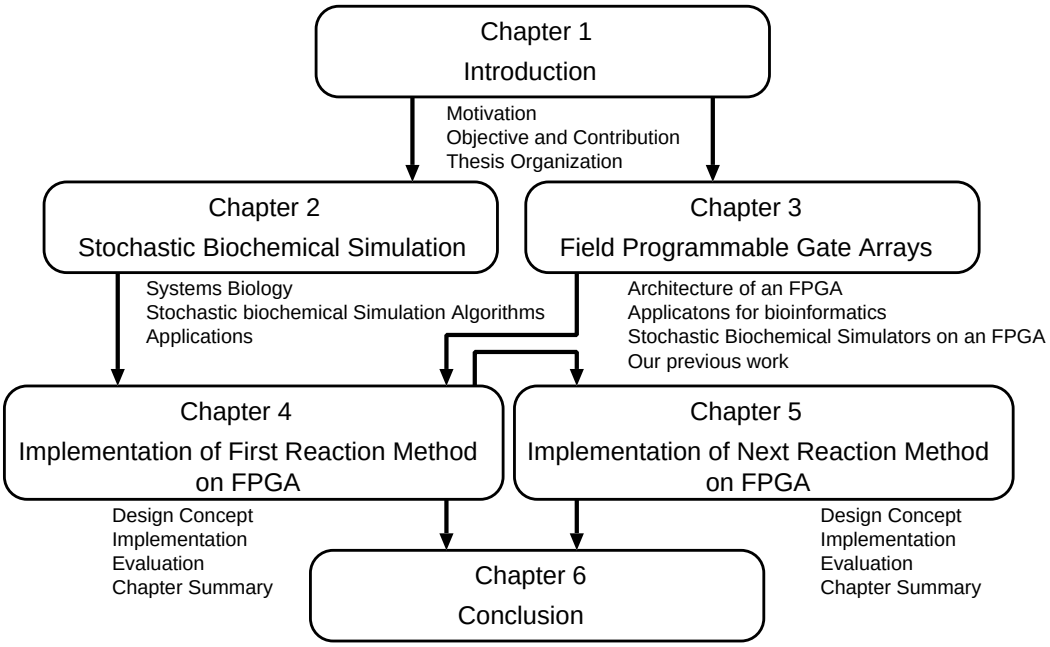


Fig. 1.1: Thesis organization

computing, recent applications for FPGA-based stochastic biochemical simulation as related work, and our previous work which suggests motivation and objective of this thesis. Chapter 4 describes the implementation and evaluation of FPGA-based stochastic biochemical simulator by First Reaction Method, followed by thorough discussion of its limitations and problems. Chapter 5 shows the implementation and evaluation of it by Next Reaction Method. Finally, Chapter 6 concludes this work.

Chapter 2

Stochastic Biochemical Simulation

2.1 Overview

Stochastic approach for biochemical simulation was originally proposed by Gillespie in 1976[8], which is a modeling technique for chemical reaction in well-stirred spaces. Gillespie modeled chemical reacting system as a list of reactions and chemical species, and presented a type of Monte-Carlo simulation algorithm to depict the time-evolution of populations with some example models. Although the simulation algorithm can calculate exact behavior that obeys to the definition of the model, it was limited to simulation of small scale models because it requires vast calculation time to acquire statistically-significant amount of simulation result. SSA has conventionally been intended for small scale biochemical models, now there are attempts to make various modifications to its simulation algorithms to accelerate and enable simulations for large scale models.

According to the fast progress of performance and cost reduction of computer, the stochastic approach has been received attention from around 2000. Since then, many researchers have developed various SSA to improve computation efficiency and to introduce specific features.

In this chapter, the System Biology which is an emerging academic field between biology and computer science is introduced in Section 2.2. Section 2.3 explains the calculation scheme of various SSAs which have been developed. This section describes the definition of the biochemical model and a recent developing stochastic algorithm fusing ODE-based approaches. Section 2.4 introduces software biochemical simulators which includes SSA-based simulator and uses SSA as a part of a simulation.

2.2 Systems Biology

Systems biology is an emerging academic field that aims at integrating information of different levels to understand the structures and dynamics of biological systems, which hopes to develop an understandable model of various life phenomena. Its main concept is the integrative study of complex network structures of life phenomena described in reaction mechanisms, such as gene regulatory networks, metabolic cycles, and organizations of inter-cellular signaling.

Since the discovery of double-helix structure of DNA by Watson and Crick in 1953 [14], a rise of molecular biology has laid a conception to regard living systems as molecular machines consisting of protein substances and nucleic acids, which successfully generated information about allosteric structures and functions of genes and proteins. However, conventional biological methodologies were capable of obtaining merely simple properties of individual components in cells. For integrative understanding of a cellular system, it is required to perform analyses and experiments of its dynamics based on collection of experimental data. This approach involves the iterative formulation of testable hypotheses, experiments, and refinement of the biochemical models based on these data.

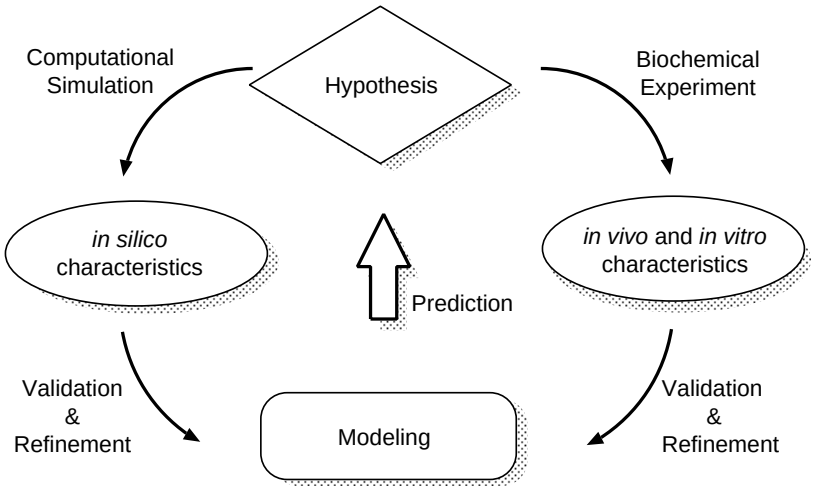


Fig. 2.1: Iterative process of model building

This iterative process draws two feedback loops: one is the conventional loop based on biochemical experiments (Fig. 2.1), and the other incorporates computational experiments, that is, *in silico* modeling and simulation. There are diverse fields where computer modeling and simulation are used, as in analysis of metabolic cycles described in ordinary differential functions and/or stochastic models, genetic regulation network written in Bayesian networks and Petri Nets. What is stated above all requires high-throughput computing systems for their huge problem sizes.

2.3 Stochastic Biochemical Simulation Algorithm

2.3.1 Overview

The theoretical derivation of stochastic biochemical simulation and a summary of recent various algorithms are introduced by Gillespie [15].

Stochastic simulation of chemical reaction proceeds in a unit of computation called “reaction cycle” as shown in Fig. 2.2. A reaction cycle is a process to (1) determine one reaction which is going to occur next and (2) update the state of the model.

The process of stochastic biochemical simulation includes vast amount of reaction cycle while each reaction cycle is consist of more simple arithmetic. Examples of simulation result which were reported by researchers indicates that a simulation has to circulate at least 10^5 reaction cycles. Especially, the HSR model in STOCHKIT which will be introduced later needs to simulate for more than 2^{32} reaction cycles. In general, thousands of simulation are required to statically analyze the behavior of the model.

In Section 2.3.1, a basis of a stochastic biochemical model which is the simulation target is explained. Scientists proposed various biochemical models with their simulation algorithm. After the explanation of some models which are used in the evaluation in this thesis, typical SSAs which have been proposed are introduced. And finally, four famous software simulators are introduced. Derivations of SSA and biochemical models are described in Appendix B.

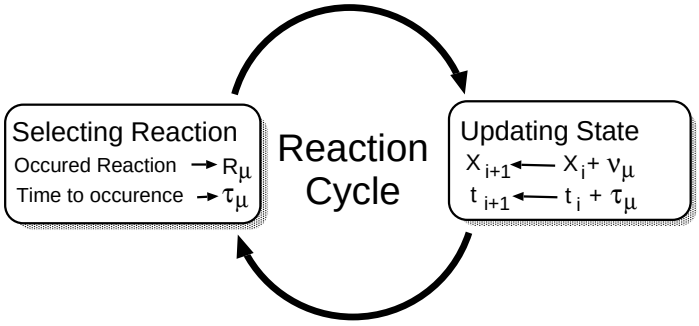


Fig. 2.2: Reaction cycle in SSA

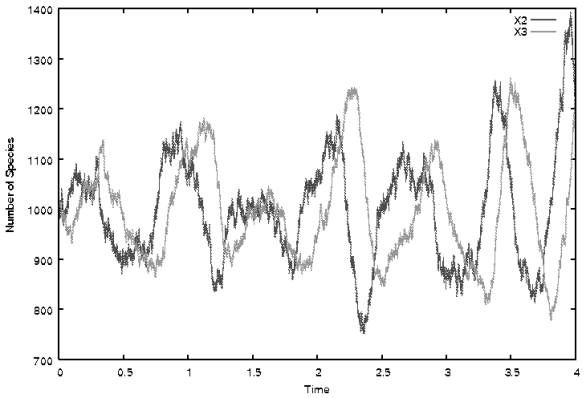


Fig. 2.3: Execution result for Lotka system(A) with random seed *A*

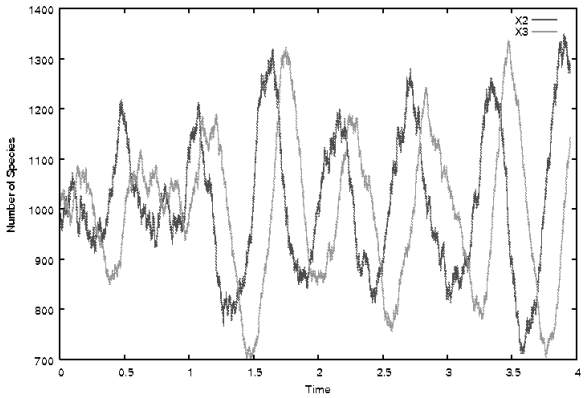


Fig. 2.4: Execution result for Lotka system(A) with random seed *B*

2.3.2 Stochastic biochemical model

(1) Definition and baseline of a biochemical model

A stochastic biochemical model consists of the list of reactions which is expressed in Eq. 2.1 as an example.

$$S_A + S_B \xrightarrow{c} S_C + S_D \tag{2.1}$$

S_x is a molecules, genes or other chemical species in the model. X_x is called "population", the actual number of S_x in the model. The couple of the list of all population $\boldsymbol{X}$ and simulation time t is a "state" of the model. The reaction in Eq. 2.1 is occurred when species S_A collides with species S_B, where the probability of occurrence of reaction c is given as "stochastic reaction rate constant". Species in the left-hand side of Eq. 2.1 is called "reactants", which causes the reaction. On the other hand, the right-hand side of Eq. 2.1 is called "products", which are generated by the result of the reaction. After the occurrence of the reaction, populations of reactants are decreased while population of products are increased. The reactants species are one or two, while the number of products is undecided. A reaction which includes only one reactants is called monomolecular reaction, and a reaction by two reactants is called bimolecular reaction.

Space complexity of most SSAs is $O(M)$, and the time complexity is up to $O(M)$, where M is the number of reaction defined in the simulation target model. As memory space is not required large for SSA, few dozens of kilo byte, studies for developing variant of SSA are entirely focused on the improvement of time complexity and throughput. As memory space of hardware accelerators is much limited compared to PCs, study for improvement of complexity is vital.

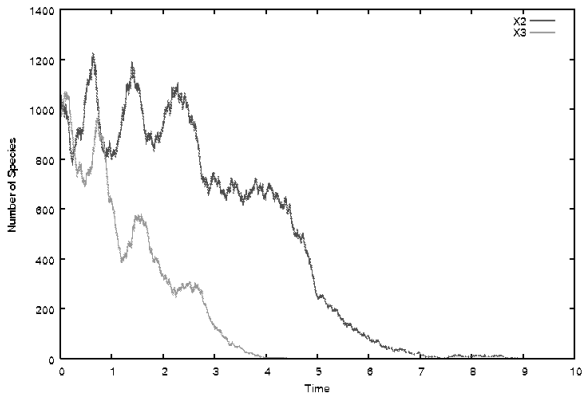


Fig. 2.5: Execution result for Lotka system(B) with random seed *A*

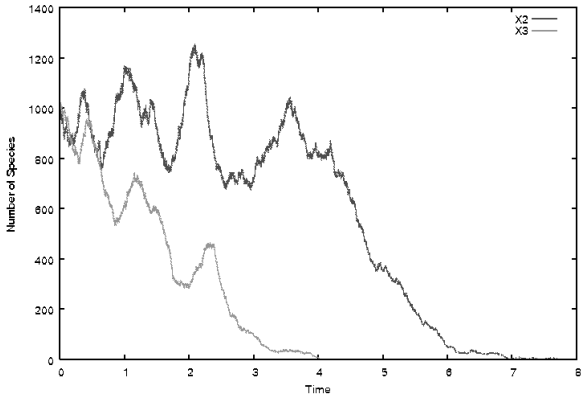


Fig. 2.6: Execution result for Lotka system(B) with random seed *B*

(2) Lotka system

Gillespie described following four biochemical models: Irreversible isomerization, Lotka system, Blusselator and Oregonator, as examples of suitable models to be simulated with SSAs [16]. As Lotka system depicts easy-to-follow trajectories of molecular number throughout the time evolution, this thesis would focus on this model as an example of stochastic biochemical model.

Lotka system developed by Volterra is a mathematical model of simple predator-prey relationship in the ecological system. Gillespie examined the theory for stochastic biochemical simulation using the stochastic form of Lotka system, which was a relatively small model that consists of four reactions by four molecular species. Table 2.1 shows the list of reactions of Lotka system in a stochastic form.

If species S_1 is assumed that its population would not decrease by R_1, populations of S_2 and S_3, which are X_2 and X_3, would oscillate around 1000, as shown in Fig. 2.3 and Fig. 2.4. These simulations begin from the initial state and parameters are summarized in Table 2.2. Fig. 2.3 and Fig. 2.4 are results of simulating 2×10^5 reaction cycles, because they do not attain the state which does not occur any reaction in the model. It means that the Lotka system with Table 2.2 is a steady state, populations of X_2 and X_3 oscillate around the 1000.

On the other hand, Fig. 2.5 and Fig. 2.6 show trajectories of S_2 and S_3 when the population of S_1 is decreased by occurrence of R_1. Same random seeds were used for Fig. 2.3 and Fig. 2.4, but only the assumption about S_1 differs. The difference between Fig. 2.5 and Fig. 2.6 is only a value of random seed. These simulations are executed until no more reaction occurs in the model or when it reaches equilibrium. Simulations are ended when no more reaction occurs next.

Table 2.1: Reactions in Lotka system

Reaction ID	Reactants		Products
R_1	$S_1 + S_2$	$\xrightarrow{c_1}$	$S_2 + S_2$
R_2	$S_2 + S_3$	$\xrightarrow{c_2}$	$S_3 + S_3$
R_3	S_2	$\xrightarrow{c_3}$	S_4
R_4	S_3	$\xrightarrow{c_4}$	S_4

(3) HSR : Heat-Shock Response

The Heat Shock Response model (which is abbreviated to HSR model) is based on results obtained in wet experiments. It models the mechanism of how the bacteria *Escherichia coli*(*E.coli*) responds to a temperature increases. When temperature is increased high enough to unhold the proteins in the cell, the *E.coli* displays complex behavior for protection. One of this behavior is that the heat shock sigma factor σ_{32} is generated very rapidly. According to rise in temperature, free σ_{32} molecule begin to bind to RNA polymerase (RNAP), and complex σ_{32} is generated by the reaction. RNAP begins the transcription of genes which encode a variety of chaperon enzymes. These chaperons take to refold or degrade unfolded proteins. Meanwhile, produced σ_{32} molecules are much more likely kept away by DNAK, which is one of chaperon enzymes.

HSR model is described both ODE-based deterministic model [17] and stochastic form of the model. Cao *et. al.* used a stochastic version of HSR model in which 61 types of reactions are defined by 28 species.

HSR is a typical model used for the analysis and evaluation of SSAs. Besides Cao's work mentioned in Section 2.3.8, Li adopted HSR as a test model for running on STOCHKIT [18].

(4) LCS: Linear Chain System

The Linear Chain System (LCS) is a hypothetical model used by Cao *et. al.* in evaluation of their proposed SSA [19]. The model contains M chain reactions with $M+1$ species as Eq. 2.2.

$$S_1 \rightarrow S_2 \rightarrow \cdots S_M \tag{2.2}$$

All reactions of LCS have uniform propensity function, $a_j(\boldsymbol{X}) = cX_i$, where $c = 1.0$. The initial state is $\boldsymbol{X} = (10000, 0, \cdots, 0)$. Although Cao used LCS with $M = 600$ and the final simulation time $T = 30$, the number of reaction for LCS can be expanded to examine the scalability of SSA. In this thesis, we use LCS to examine performance of SSA with various M.

(5) TIS: Totally Independent System

The Totally Independent System (TIS) is also a hypothetical model used by Cao [19]. This model contains M independent decaying processes as follows:

$$S_i \rightarrow \phi \qquad\qquad (i = 1, \cdots, M) \tag{2.3}$$

TIS is possibly the most loose and the simplest model. As same as LCS, Cao *et. al.* used TIS with $M = 600$. The propensity functions in TIS are same with $a_j(\boldsymbol{X}) = cX_i$, where $c = 1.0$. The initial state is stored as $\boldsymbol{X} = (1000, 1000, \cdots, 1000)$.

In this thesis, we also use TIS to examine performance of SSA with various M.

Table 2.2: Initial parameters in Lotka system

i	Initial Number of S_i (which is X_i)	Stochastic Reaction Rate Constant k_i
1	100000	0.0002
2	1000	0.01
3	1000	10.0
4	0	10.0

Table 2.3: Propensity Function : Equations to calculate propensity

Reaction	Reactants From	Calculation Scheme
R_1	S_a	$c_1 \times X_a$
R_2	$S_a + S_a$	$c_2 \times X_a \times (X_a - 1) \div 2$
R_3	$S_a + S_b$	$c_3 \times X_a \times X_b$

2.3.3 Common variables and operations in SSAs

Section 2.3.3 explains the definition of variables and operation which are commonly used in all of SSAs.

(1) Propensity

Propensity, a likelihood of reaction occurrence, is a very important parameter to determine the next reaction.

Table 2.3 shows schemes to calculate propensity depending on the forms of reactants. A reaction with a reactant like R_1 in Table 2.3 is called a monomolecular reaction, and a reaction within two reactants as R_2 and R_3 is called bimolecular reaction. Propensities for all reactions defined in the biochemical model is introduced according to the scheme of Table 2.3. It is said that propensity is obtained by multiplying the combination of reactants population, and its stochastic rate constant which is introduced in Appendix B. Propensity has to be recalculated when the number of the reactants population is updated.

(2) State-update vector

Each reaction defined in stochastic biochemical model has "state-update vector" $\boldsymbol{\nu_j}$ as shown in Eq. 2.4. The vector consists of M elements which increases and decreases population the reaction occurred.

$$\boldsymbol{\nu_j} = \{\nu_1, \cdots \nu_M\} \tag{2.4}$$

After the selection of a reaction, the state of the model is updated by adding state-update vector of occurred reaction $\boldsymbol{\nu_\mu}$ to the state of current model, as shown in Eq. 2.5.

$$\boldsymbol{X_{i+1}} \leftarrow \boldsymbol{X_i} + \boldsymbol{\nu_\mu} \tag{2.5}$$

State-update vector for each reaction is easily generated by form of the reaction. After all elements are set as 0, elements correspond to reactants are decreased and elements correspond to products are increased.

For example, state-update vectors of reactions in Lotka system are shown in Eq. 2.6-Eq. 2.9.

$$\boldsymbol{\nu_1} = \{-1, 1, 0, 0\} \tag{2.6}$$
$$\boldsymbol{\nu_2} = \{0, -1, 1, 0\} \tag{2.7}$$
$$\boldsymbol{\nu_3} = \{0, -1, 0, 1\} \tag{2.8}$$
$$\boldsymbol{\nu_4} = \{0, 0, -1, 1\} \tag{2.9}$$

(3) Handling simulation and termination condition

Although not strictly defined, following three conditions are substantial terminal conditions of SSAs:

1. Reaching certain pre-defined number of reaction cycle

2. Reaching certain pre-defined simulation time
3. Reaching certain state of the model satisfies a specific condition

2.3.4 FRM: First Reaction Method

First Reaction Method[8, 16], which is abbreviated as FRM, is one of an original SSA proposed by Gillespie in 1976. The computational procedure of FRM is as follows:

STEP1 Set initial populations of all species $\boldsymbol{X_0}$, a random seed r_0 and initial time $t = 0$.

STEP2 Obtain $\boldsymbol{\tau} = \{\tau_1, \cdots, \tau_M\}$ with Eq. 2.10 for all reactions $(R_j : j = 1, \cdots, M)$

$$\tau_j = \frac{1}{a_j} \ln\left(\frac{1}{r_j}\right) \tag{2.10}$$

- a_j is a propensity of reaction R_j calculated by populations of its reactants and reaction rate constant(See Table 2.3)
- r_j is a uniform random number(The range of r_j is $[0, 1)$).

STEP3 A reaction R_μ whose element of $\boldsymbol{\tau}$ is the minimum value, is selected as the reaction which occurs next.

$$\tau_\mu = \min\{\boldsymbol{\tau}\} \tag{2.11}$$
$$\mu = \text{Reaction ID}\left(\tau_\mu\right) \tag{2.12}$$

STEP4 Simulation time t is updated by adding t and τ_μ.

$$t = t + \tau_\mu \tag{2.13}$$

STEP5 State of the model is updated by state update vector $\boldsymbol{\nu_\mu}$.

$$\boldsymbol{X} \leftarrow \boldsymbol{X} + \boldsymbol{\nu_\mu} \tag{2.14}$$

STEP6 Calculation of a reaction cycle is completed. Return to Step.2

A simulation proceeds repeating Step.2 to Step.3 until it satisfies the termination conditions.

2.3.5 DM: Direct Method

Direct Method, which is abbreviated as DM, is a statically equivalent algorithm with FRM.

The algorithm of DM is similar with it of FRM. Step 2 and Step 3 in FRM are replaced following three schemes:

STEP2 Obtain propensity $\boldsymbol{a} = \{a_1, \cdots, a_M\}$ for all reaction $(R_j : j = 1, \cdots, M)$. Each of them is calculated by Table 2.3 with populations of reactants and reaction rate constant.

STEP3 Calculate sum of all element of $\boldsymbol{a}$, which is a_0.

$$a_0 = \sum_{j=1}^{M} a_j \tag{2.15}$$

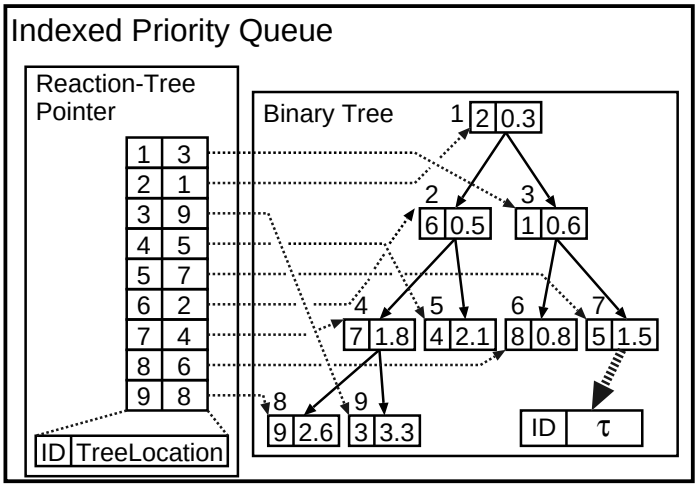


Fig. 2.7: A structure for Indexed Priority Queue

The time of the next reaction τ_μ and the index of the reaction μ are calculated by following two equations, respectively. They use two uniform random numbers r_1 and r_2, whose ranges are $[0, 1)$.

$$\tau_\mu = \frac{1}{a_0} \ln\left(\frac{1}{r_1}\right) \tag{2.16}$$

$$\sum_{j=1}^{\mu-1} a_j < r_2 a_0 \leq \sum_{j=1}^{\mu} a_j \tag{2.17}$$

2.3.6 NRM: Next Reaction Method

Computation time of SSAs, are proportional to the number of reaction M defined in the model (Both computational order is $O(M)$). In 2000, Gibson and Bruck proposed an algorithmic methodology to reduce computational order called Next Reaction Method or NRM, for simulating large-scale biochemical model [20].

NRM is based on FRM, but introduces two types of data structure called Indexed Priority Queue (IPQ) and Dependency Graph (DG).

In one reaction cycle of FRM, a reaction R_μ which has the minimum element of $\boldsymbol{\tau}$ is selected as a reaction which occurs next. Each element of $\boldsymbol{\tau}$ is regarded as the putative time of the next reaction. It means that putative times are unused, but are recalculated per cycle in vain.

IPQ reduces the time for searching the minimum value, and DG minimize the number of calculation Eq. 2.10. Computational order of NRM improves from $O(M)$ to $O(\log(M))$ while preserving statical equivalence with FRM. NRM is regarded that it has a potential for large scale simulation which includes large number of reaction in the the biochemical model.

(1) IPQ: Indexed Priority Queue

Fig. 2.7 shows a data structure of Indexed Priority Queue (IPQ) with two memories, a binary tree and a one dimensional array. The binary tree contains a set of reaction ID and its putative time (j, τ_j) in a node. τ_i in a parent node is always smaller than τ_{2i} and τ_{2i+1} in its children nodes. i is the node index of the tree. One dimensional array is a pointer table which points to a location of the node in the binary tree for each reaction ID. If the tree satisfies the condition, the smallest τ_j $(= \tau_\mu)$ is stored in the root node. The reaction ID μ is also obtained by the node from the root node.

Table 2.4: Operations for IPQ

ADD	Adding a node at the end of the tree. After that, it is compared and several nodes are exchanged in parental direction if needed.
UPDATE	Update putative time of a reaction τ_j and it is compared and several nodes are exchanged in parental direction if needed.
READ	Reading a node from the arbitrary location in the tree.

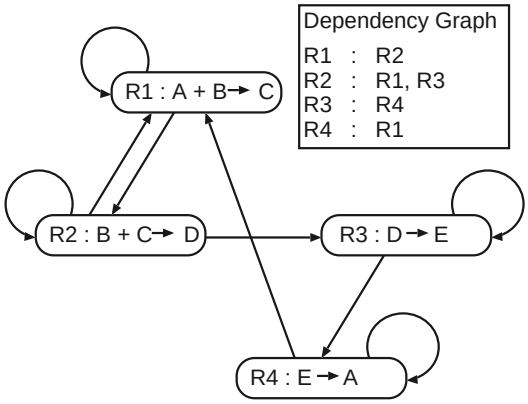


Fig. 2.8: A structure for Dependency Graph

The putative time of arbitrary reaction can be obtained by getting tree location of the reaction from pointer table, and reading the tree node according to the location.

IPQ needs to provide three operations as shown in Table 2.4. By exchanging nodes, variables in the pointer table are swapped. Number of nodes M, which is identical to the number of reactions defined in a biochemical model. The time complexity of NRM is $O(\log(M))$ because time to update the tree whose number of node in the binary tree is M, is domestic in a simulation for a biochemical model which has large M.

(2) DG: Dependency Graph

Updating populations alters putative times of a reaction. However, the influence of a reaction is restrictive for only a few reaction whose populations are updated by occurred reaction. So, FRM may include quite a large amount of wasteful computation because it recalculates all τ_j every reaction cycle.

Dependency Graph (DG) is a directed graph which indicates the relationship among reactions. Fig. 2.8 shows an example of DG. Four reactions are defined in the example, each of them is assigned at a vertex in DG. Each vertex has a loop edge to itself. Each reaction has edges to others when destination reaction has a species in source reaction as reactants. DG is a list of reaction which is updated when a reaction occurs.

For example, suppose R_2 in Fig. 2.8 is selected as a next reaction. The putative time of R_2 is recalculated according to Eq. 2.10 as same as FRM.

Populations of B, C and D are updated by R_2. According to occurrence of R_2, putative times have to be updated for reactions including these three species as its reactants. As vertex R_2 in DG has two

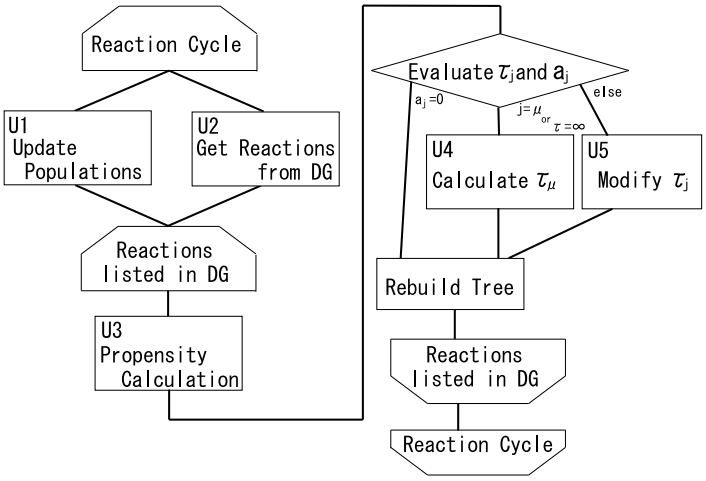


Fig. 2.9: Flowchart of NRM

edges for R_1 and R_3, putative times τ_1 and τ_3 are modified by Eq. 2.18 with updated state.

$$\tau_{j,\text{new}} = \frac{a_{j,\text{old}}}{a_{j,\text{new}}}\left(\tau_{j,\text{old}} - t\right) + t \tag{2.18}$$

Gibson *et. al.* reported that average number of edges from a vertex is $D = 4.2$ in the λ-phage model which consist of 51 reactions and 71 species [20]. D has generally no direct bearing on the size of the model M, because it means only a barometer of complexity among reactions in the model. Although value of D determines the average number of calculation of Eq. 2.18 in a reaction cycle, the computation time for updating the tree is more domestic at in whole simulation of NRM. After that, Cao *et. al.* computationally obtained that computational time does not depend only on the number of reaction N but also on an average number of modification of putative time D in biochemical models with around hundreds of reactions [19]. Their research outcome is introduced in Section 2.3.8.

(3) Algorithm

Fig. 2.9 shows the flowchart of calculation for a reaction cycle by NRM. The computational procedure of NRM is as follows :

STEP1 Set initial values for:

- model state $\boldsymbol{X}(\boldsymbol{t_0})$
- simulation time $t \leftarrow 0$
- DG

STEP2 Calculate τ_j for all reaction by Eq. 2.10 and store them into IPQ

STEP3 Select root node in the binary tree in IPQ as next reaction R_μ

- Update state($\boldsymbol{X}(\tau_\mu) \leftarrow \boldsymbol{X}(t) + \boldsymbol{\nu}_\mu$)
- Overwrite simulation time with the value of the root node ($t \leftarrow \tau_\mu$)

STEP4 New putative time for occurrence of R_μ is calculated by adding the result of Eq. 2.10 for τ_μ and current simulation time t, and update the time in the root node (UPDATE operation will be used).

Table 2.5: Features of FRM, DM and NRM

	FRM	DM	NRM
Random numbers used in a reaction cycle	M	2	1
The number of the most heavy function	$M \times \boldsymbol{\tau}$	$M \times \boldsymbol{a}$	$D \times$ UPDATE(IPQ)
Time complexity	$O(M)$	$O(M)$	$O(\log(M))$

STEP5 Get the list of reaction whose putative time τ_j has to be modified by occurrence of R_μ.

STEP6 Modify all reaction in the list by reading and writing binary tree and obtaining τ_j by Eq. 2.18.

STEP7 Calculation of a reaction cycle is completed. Return to Step 3.

2.3.7 Summarize of exact SSAs

FRM, DM and NRM is developed focusing on the computational efficiency. Features of these algorithms are summarized as Table 2.5. These algorithms are proved that their simulation results are statistically same. In time complexity, NRM is the most efficient algorithm while the calculation scheme is rather complex. Although computations of FRM is wastefulness, the procedure is the simplest and implemented easily. DM is known as an effective scheme to run on computers, and it is frequently used as the benchmark for measurement of computational performance.

2.3.8 ODM: Optimized Direct Method

Cao *et. al.* deeply investigated the computational efficiency of various SSAs using by scalable models, and proposed new optimized algorithm of DM called Optimized Direct Method (ODM) [19].

As domestic function in DM is calculation of propensity a_0, they tried to reduce the cost of this function. It includes three major function branches :

1. Calculating propensities $\boldsymbol{a}$
2. Accumulating a_j to obtain a_0
3. searching μ

ODM applies two optimizations for DM, (1) re-ordering reactions and (2) the dependency graph like NRM.

First optimization is the re-ordering of reactions. They focused on the fact that frequencies of occurrence reaction are biased in an actual large-scale model in which hundreds or thousands of reactions are defined. That is, as biochemical models include almost multi-scale parameters such as numbers of species and reaction rate constants, a few particular reactions are selected very frequently in a reaction cycle. For example, in HSR, they reported only six types of reactions occur for 90% of the whole a simulation, and 99% in case of 12 reactions. Therefore, they found that calculation time for searching μ could be reduced by re-defining the model by organizing the list of reaction in order of frequency. To count the number of occurrence of reaction to obtain the frequency, several simulations have to be run by some kind of SSA. According to results of simulation, reactions in the model is sorted in a descending order of frequency.

Calculations for re-ordered model reach the break instruction in Eq. 2.17 faster than calculations for pre-ordered model. Re-ordered HSR model could improve computation time for about 11.87% compared to default HSR model. In addition, it can also improved computation time about 25.12% compared to the worst HSR model which was sorted in ascending order of frequency.

Table 2.6: Execution time for LCS and TIS on 1.4GHz Pentium 4

	LCS [sec]	TIS [sec]
DM	2.13	5.39
NRM	1.07	2.63
ODM	0.86	3.72

Second feature is dependency graph to reduce the number of calculation for calculating propensities $\boldsymbol{a}$ and accumulating a_j to obtain a_0. Like NRM, it does not need to recalculate all propensity a_j every reaction cycle, but only necessary minimum number. Therefore a dependency graph can also be exploited to reduce the calculation. It can be realized by subtracting old $a_{j,\mathrm{old}}$s from a_0 and adding new $a_{j,\mathrm{new}}$s instead of accumulating a_j again every reaction cycle.

Cao *et. al.* reported execution time for LCS and TIS on 1.4 GHz Pentium 4 as shown in Table 2.6. Calculation cost of ODM is higher than that of NRM in case of $D \ll M$ and $S \approx M/2$, where S is the time to find μ, but actual biochemical systems are often multi-scale, so $S \ll M/2$ is approved. Thus, ODM is an effective algorithm for simulating many biochemical systems.

2.3.9 τ-leaping Method

τ-leap method, which was proposed by Gillespie in 2001, calculates behavior of biochemical model by a unit of finite minimal time to reduce calculation time [21]. Exact SSAs described above need to repeat calculation of a reaction which occurs next and update state and simulation time one by one. Hence, it becomes one of the major cause of vast calculation time in the case when many types of molecules are defined in the model. However, Gillespie argued that practical biochemical simulations do not require such complete record about the order of occurrence and time about every reaction. For example, when populations of a few species fluctuates, many reactions may be selected not to affect the behavior of the model. In a such case, the exact SSA will not compensate for the computational time. τ-leap method is developed reduce the computational cost by changing a unit of one loop from one reaction to one time step.

Suppose propensity vector $a_j(\boldsymbol{X})$ at the population vector $\boldsymbol{X}$. If increase or decrease of species is within the allowable range, value k_j, number of times that reaction R_j occurs between time $[t, t+\tau)$, is obtained from random number that is in the range of $P(a_j(\boldsymbol{X}), \tau)$, a probability density function for Poisson distribution. Every minimal timestep, $a_j(\boldsymbol{X})$, a probability for occurrence of reaction R_j, is calculated for all js, its number of occurrence is obtained from a Poisson random number, and the result is reflected to model status $\boldsymbol{X}$. Some errors can be observed to value of probabilities for occurrence of reactions, because it depends on system's state $\boldsymbol{x}$ which varies according to number of molecules in the system.

Gillespie compared the error rate of τ-leap method with other rigorous SSA algorithms with simulated system with four reactions. As the result, he found that reasonable approximation is possible by determining a value ϵ, an admissible error coefficient. ϵ varies according to the system, so current τ-leap Method cannot be applied for simulations of general systems. Now there are ongoing studies on determination methodology of ϵ and algorithms that accommodate to general systems.

The original τ-leap method is called explicit τ-leap method. It is still developing and challenging algorithm as an alternative of exact SSA, and Gillespie expects that it is going to be a promising approximation method for exact stochastic biochemical simulation. Actually, various modified algorithm of τ-leap method have been proposed, such as implicit τ-leap method [22] and trapezoidal τ-leap method [23].

2.4 Applications

2.4.1 Overview

Software developers and biologists have built various simulation software. Some of them follow proposed algorithm exactly, and some extend their unique features. This section introduces four representative software called STOCKS, STOCHSIM, E-cell version 3 and STOCHKIT around the stochastic biochemical simulation.

2.4.2 STOCKS

STOCKS(STOChastic Kinetic Simulation) [9] developed by Kierzek is a software for the stochastic kinetic simulation.

(1) Aspect

STOCKS is software which adds two main functions to Direct Method : (a) cell growth and division and (b) random pool.

First, STOCKS accommodates to simulate several cell generations. It introduces the scheme to treat the growth of cell volume with time and divide into two cells. Following two natures are assumed to simulate cell growth and division. (1) volume of a cell $V(t)$ is increased according to the time evolution and the doubled for a cell generation. (2) after cell division, a volume of each cell is halved to the initial value. These features are implemented by introducing function $V(t)$ with simulation time t as a variable. $V(t)$ is a function given by Eq. 2.19.

$$V(t) = 1 + \frac{t}{T} \tag{2.19}$$

T is value for a time of a generation of the cell. By introducing Eq. 2.19, it can express the linear increase of cell volume according to time evolution.

When the model introduces the cell generation, stochastic reaction rate constants c_j are updated by the value of $V(t)$ in each simulation cycle. The relationship between reaction rate constant c_j and cell volume V is explained in Appendix B. When $t = T$ which means $V(t) = 2$, it reaches the end of generation of a cell and cell division occurs. Stochastic reaction rate constants c_j is determined by Eq. 2.20.

$$c_j = \frac{k_j}{N_A V} \tag{2.20}$$

N_A is an Avogadro number. The second order reaction uses Eq. 2.21 instead of Eq. 2.20.

$$c_j = \frac{2k_j}{N_A V} \tag{2.21}$$

At the moment of cell division, volume V and values c_j are return to initial values and the species numbers are divided by 2. Numbers of species which model DNA are doubled before division to hold the populations after cell division. The technique to express cell growth and cell division as linear change of volume can approximate simulation results with model of procaryotic organism [24, 25].

Second, it introduces the “random pool” as population of chemical species. The population as random pool changes by every read operation. It can be useful to introduce cellular substances which are in dynamic equilibrium as a result of many competing processes. STOCKS introduce this feature for the RNA polymerase pool whose molecular number is obtained by Gaussian distribution with a specific mean and standard deviation before computing h_μ and a_μ value with Direct Method. The mean of the distribution can be set according to experimental estimates. Kierzek obviously explained that even though random pools can be justified in many cases, there is no mathematical proof.

(2) Implementation

STOCKS written in C++ language with an object-oriented programming. It is designed to suite background job on a UNIX operating system, because it requires long execution time of a few hours or even days. STOCKS uses Marsaglia's random number generator [26] whose cycle is 2^{144}.

STOCKS also provides following four utility programs to analyze simulation results.

1. Obtaining average trajectories within specified time interval
2. Adding or subtracting populations for two trajectories
3. Computing mean population
4. Fitting the linear function to the specified part of the trajectory

(3) Evaluation

Kierzek shows two examples to explain STOCKS's performances based on executions on a single Pentium III 800MHz processor running on Linux operating system. Both results are observed as same as original behavior of these models.

First is the kinetic model of procaryotic gene expression which was presented in Kierzek's previous paper[24]. It models the speed of protein synthesis and mRNA levels in the *lacZ* gene of *E.coli*. This model consist of eleven reactions by twelve chemical species. STOCKS runs 100 simulations this model for ten generations of the cell with a generation time of 2100 seconds. This execution takes about 22 hours CPU time.

Second is a model *lacZ* and *lacY* genes expression and enzymatic/transport activities of LacZ and LacY proteins. Kierzek focused on this example to investigate the applicability of STOCKS to models involving both enzymatic processes and small populations. A number of reactions whose magnitude is between 10^8 and 10^9 in a generation. An execution time is about 2.5 hours for one simulating generation, and about 90 hours for ten generations.

2.4.3 STOCHSIM

STOCHSIM [10, 27, 28] which was developed by Shimizu *et. al.* is a biochemical simulator that applies stochastic approach. The purpose of STOCHSIM is to overcome weakpoints of Gillespie's algorithm.

(1) Weakpoint of Gillespie's algorithm and its solution

SSAs proposed by Gillespie only focused on type of chemical species and their population. Like traditional ODE-based simulation method, it does not consider spatial distribution of chemical species in the model. This cannot accommodate to models which defines proteins that multiply-transforms due to isomerization activities and/or affinity bindings. The reason is that the status of each molecules may exhibit heterogeneous properties, and the model may be described in as many as several million reactions as its refinement proceeds. Hence, it is intractable to implement these features, because simulation time increases linearly to the number of reactions with both analytical and Gillespie's approaches.

STOCHSIM extend the format of molecule to accommodate to various 3-dimensional structure and features.

(2) Aspect

- **Definition of multistate molecule**
  STOCHSIM calculates the behavior of the model by basic monomolecular reaction and bimolecular reaction. Each molecule in the model is defined as the object called "Multistate molecules". It can store several molecular expressions in themselves. The variable in the object is used when the reaction probabilities are different depending on the state such as molecular interactions such as covalent modifications and conformational changes.

- **Definition of reaction probability and calculation time**
  STOCHSIM applies a simulation method of repeating calculation per constant minimal time step instead of obtaining simulation time based on a stochastic model. In the beginning of initialization of a simulation, reaction probabilities of all reactions are computed based on user-defined kinetic constant values, and are stored in an LUT. Concrete time step per one simulation cycle is adjusted so that it does not affect the user-define kinetic constant values per cycle.

  After these basic configuration is set, the simulation proceeds by repetition of simple computing processes which is to randomly select on unimolecular or bimolecular reaction. Unique approach of STOCKSIM is that users can define "pseudo-molecules" for the aim of simplification. Pseudo-molecules are virtual molecules which do not exist in real models, and is only used to simplify the description of mono- and bimolecular reactions. For example, when one out of the two reactant molecules was a pseudo-molecule, the reaction is treated as monomolecular. Quantity of pseudo-molecules represent occurrence frequency of monomolecular reactions.

- **Implementation**
  Original STOCHSIM was developed as a simulator run on Microsoft Windows operating system, and it provided GUI operating environment which was based on MS/MFC(Microsoft Foundation Class). Recent STOCHSIM is written in Standard C++ language, and its GUI operating environment written in Perl/Tk interface to execute on other operating systems such as UNIX, Linux, and MacOS.

  STOCHSIM can output numbers of molecules and their molecular state at every time interval. It can also display graphical snapshot for the state of the molecules and their distribution in the simulation space.

Algorithm

Simulations using STOCHSIM proceeds in a following manner. Simulation time is quantized to discontinuous minimal time steps. Per time step, one type of object among molecules in the system is randomly selected. Note that the molecule selected at this point cannot be a pseudo-molecule. Next, another object is selected among the system, and this time it could possibly be a pseudo-molecule. If two actual molecules are selected, it would cause a bimolecular reaction. If the pair is one actual molecule and a pseudo-molecule, STOCHSIM treats this as a monomolecular reaction.

A value of reaction possibility for the occurred reaction would be read from software-based LUT. Based on this value, a random number is generated, and the reaction is judged by comparing the random value and reaction possibility. If the reaction occurred as a result of the judge, the system is updated by modifying its populations and their status.

The following equation computes a probability of reaction occurrence after a selection of a molecule.

- If a molecule A and a pseudo-molecule are selected, a probability of A reacting in a monomolecular reaction is

$$P = \frac{k_1 \times n \times (n + n_0) \times \Delta t}{n_0} \tag{2.22}$$

- If two actual molecules are selected, a probability of them reacting is

$$P = \frac{k_2 \times n \times (n + n_0) \times \Delta T}{2 \times N_A \times V} \tag{2.23}$$

Variables in equations above indicate following factors.

- N_x : populations of species x in the model
- n_0 : population of pseudo-molecule in the model
- k_1 : Reaction rate constant of monomolecular reaction (s^{-1})
- k_2 : Reaction rate constant of bimolecular reaction ($M^{-1}s^{-1}$)
- Δt : Time step
- N_A : Avogadro number
- V : Volume of the model

Number of pseudo-molecules is determined by Eq. 2.24 where $[x]$ represents the closest natural number to x.

$$n_0 = \left[2 \times N_A \times V \times \frac{k_{1,\max}}{k_{2,\max}} \right] \tag{2.24}$$

In Eq. 2.24, k_1 and k_2 represent fastest reaction probabilities among all monomolecular reaction and bimolecular reaction, respectively.

Multistate molecules

In STOCHSIM, instances of molecular objects have an internal state. Enzyme activities and signal transmission of proteins are controlled by various factors such as covalent modifications, bindings of ligands and subunits, and protein conformational changes. They are modeled by internal state of molecular object instances. Precisely, status of a molecule is managed by a list of 1-bit flag. For example, there is a flag that determines whether an external ligand would bind to a membrane trafficking acceptor.

Developmental status

Version 1.4 of STOCHSIM was released in 2004 [27]. It supports realistic models compared to Gillespie's algorithm in terms of computation using small time step value or multistate molecules. Also it supports models that are affected by thermal dynamics. On the other hand, there is a strong demand for a computation environment with larger memory capacity and computation power to reduce simulation time. Original version 1.0 could treat a well-stirred solution reacting system, and version 1.2 supports cross-reaction among molecules in two adjacent lattice when the system space is portioned in a 2-D grid. This feature can be applied to simulating receptors closely-spaced on cell membranes [28]. Studies on expression methods of system space is still being refined to this day.

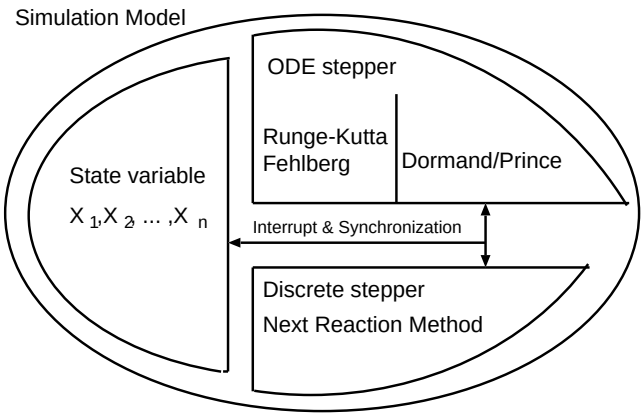


Fig. 2.10: A simulation model of E-Cell version 3

2.4.4 E-Cell version 3

E-Cell is a whole cell simulator which has been developed by Tomita laboratory in Keio University [4]. E-cell version 3 is an integrated cell simulator which adds stochastic simulation feature to traditional ODE-based simulation [29].

E-Cell 3 is an object-oriented cell simulator written in C++ language. It allocates each reaction which is defined in the simulation target model to the class called stepper, and can execute hybrid simulation of ODE- and stochastic approach. Each stepper can choose the quantitative changes of species from 3 types, ODE-based analytically-determined variation, discrete event and discrete time. Fig. 2.10 shows the simulation model in E-cell3.

E-Cell 3 examines interrupt signals from steppers in it according to progressing the simulation. When the interruption request comes from a stepper, the model synchronize all stepper and update its state.

ODE-based stepper can select a solver depending on required accuracy. E-Cell 3 provides two solvers based on explicit Runge-Kutta method: Fehlberg method to calculate the second order accurate solutions and Dormand/Prince method to calculate that of the forth-order.

ODE-based stepper calculates time-evolution of populations every time step. At that time, E-Cell 3 adjusts time interval in a step adaptively through comparing populations of pre- and post-time step. While step interval is shortened to cover loss of precision when variation of species reads rapid fluctuation, it is alternatively elongated to accelerate computational speed when the fluctuation velocity is low. A simulation proceeds by repeating synchronization of all stepper with the stepper which has the minimum simulation time.

On the other hand, a stochastic stepper calculates simulation by NRM. Mersenne-Twister [30] is adopted as a random number generator algorithm. While exact fluctuation of populations can be observed by a stochastic stepper, it is strictly required to synchronize among steppers. Therefore, each stochastic stepper has a parameter ξ as an error margin. In example of $\xi = 0.1$, synchronization is conducted when 10% of the population is exchanged.

To examine this algorithm, Takahashi *et. al.* [29] evaluated calculation speed by HSR with three different approaches : all reaction is allocated as ODE-based stepper, all reaction is allocated as stochastic stepper, and hybrid models. The hybrid model outperforms by 2.6 times compared to the ODE-only model, and it is also 351 times faster than the stochastic-only model. The cause that the hybrid model is faster than ODE-only model may be that stiff reaction is allocated to the stochastic stepper, according to their report.

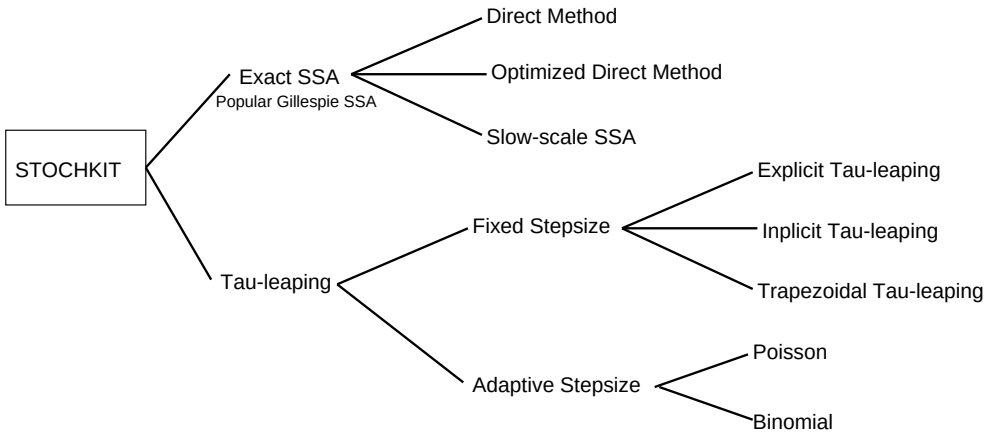


Fig. 2.11: Algorithms in STOCHKIT

2.4.5 STOCHKIT

STOCHKIT(Stochastic Simulation Kit) developed by Cao *et. al.* is a stochastic biochemical simulation framework written in C+ language [31]. Although it is now in beta version, various algorithms are provided. They are roughly divided into two choices, popular Gillespie SSA and τ-leap method as shown in Fig. 2.11. Popular Gillespie SSA includes Direct Method [16], Optimized Direct Method [19], and slow-scale SSA [32]. τ-leap method also adopts explicit τ-leap method [21], implicit τ-leap method [22], and trapezoidal τ-leap method [23].

At the beginning of a simulation, a biochemical model written in SBML is converted to an input file for STOCHKIT. A simulation process is executed according to the algorithm descripted in the setting file, and output result data is written per time interval. STOCHKIT provides not only simulation engine, but also some additional components to analyze results and high performance computation. STOCHKIT accommodates to parallel execution environment such as PC-cluster by utilizing its MPI interface.

Cao *et. al.* bundled some sample biochemical models for use on STOCHKIT. Especially, the HSR model for STOCHKIT is used in the evaluation benchmark in this thesis.

Chapter 3

Systems using Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are a commercially available silicon devices which provide instant manufacturing of logical functions with low prototype costs [33]. Fine-grain programmable logic blocks with Look-Up Tables (LUTs) allow users to customize large-scale circuit designs of their desire without any fabrication facility.

3.1 Architecture of an FPGA

FPGA is a programmable device whose basic unit is an LUT. An *n*-input *m*-output LUT consists a logic function memory for *n*-bit address and *m*-bit words. Majority of commercial FPGAs consist of 4-input 1-output LUT, which makes a memory with 4-bit address and 1-bit word. There are basically three types for the process technology of FPGAs:

- SRAM
  A volatile memory, which is applied to a majority of commercial FPGAs. These FPGAs must be configured each time the power is applied to the chip. Compared to other technologies, it requires relatively large chip area.

- Anti-fuse ROM
  An electrical device that are widely used to permanently program logic circuits. The chip area becomes relatively small using an anti-fuse, but their manufacture requires modification to the basic CMOS process, which is the major disadvantage.

- EEPROM
  A non-volatile storage chip, especially used in devices such as USB flash drives. It can be programmed and erased electrically in-circuit, unlike EPROM whose trapped charge needs to be removed from the floating gate when it is re-programmed. The disadvantage is that they consume about twice the chip area of EPROM transistors, and also require multiple voltage sources.

The programmable resource of Xilinx Virtex-II/Virtex-II Pro series, a core device of ReCSiP, are SRAM-based FPGAs. Its manufacture process is quite similar to that of CMOS, so that it benefits richly from the latest technology of circuit scale expansion.

The architecture of Xilinx FPGA is as in Fig. 3.1. It adopts Island-style, the most typical architecture. It consists of the following three blocks:

- Logic block

- Connection block

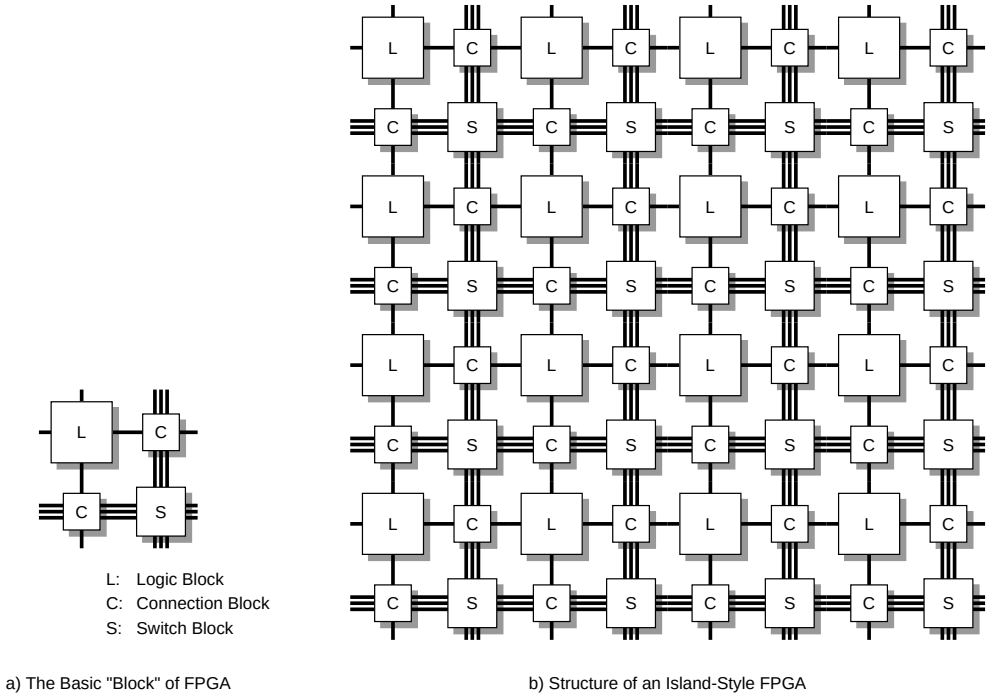


Fig. 3.1: Architecture of Island-style FPGA

- Switch block

Fig. 3.1(a) is the most basic unit, and a large-scale circuit shown in Fig. 3.1(b) is configured by laying out many of them. The uniformity of its structure fabricates the manufacturing of FPGAs with different logic size, offered simply by altering the number of logic blocks(L).

A logic block consists of an LUT and an interconnection to a neighboring connection block(C). A connection block is a programmable switch that is used to connect the pins of the neighboring logic blocks. At the intersections of horizontal and vertical routing channels is a switch block(S), which is a routing switch connecting wire segments in a channel segment to those in another. This routing architecture is the key for high flexibility of an FPGA.

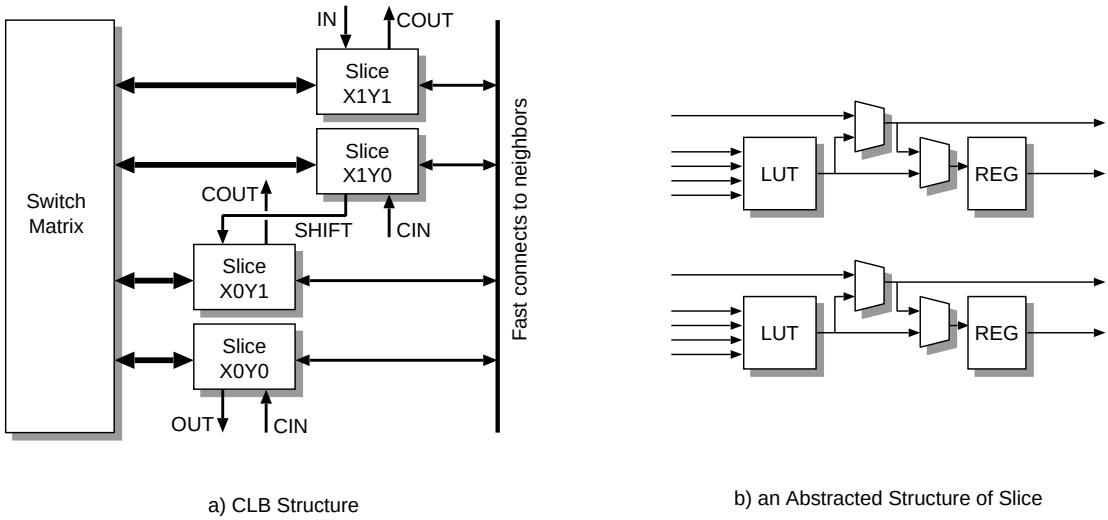
3.2 General architecture of Xilinx Virtex-II FPGA

This subsection provides descriptions of Xilinx Virtex-II architecture. This general architecture is common in the following families of Xilinx FPGAs: Virtex-II series [34] and Virtex-II Pro series [35].

3.2.1 Architecture of CLBs

The architecture of a logic block is shown in Fig. 3.2. A Configurable Logic Block (CLB) is depicted in Fig. 3.2 (a), which was described as a “logic block” in the previous subsection. One CLB consists of four blocks called slices. The architecture of a slice has 4-input LUT and two registers, as shown in Fig. 3.2 (b). A sequential circuit can be set up with the LUTs and a combinational circuit by the registers. A multiplexer between the LUT and the registers may directly bypass the output of LUTs, or configure a logical function of four input or more by connecting multiple LUTs in a cascade form.

Each slices also have dedicated resources for several commonly used circuit structures, as shown in Fig. 3.2 (a). For example, when an LUT is used as shift registers, these dedicated channels may



a) CLB Structure

b) an Abstracted Structure of Slice

Fig. 3.2: Architecture of a CLB of Virtex-II FPGA

eliminate circuit area of an FPGA and improve its maximum frequency. Then IN/OUT pins become a shift-chain which effectively consists a long shift registers between multiple LUTs and slices, while CIN/COUT pins connect carriers from full adders.

3.2.2 The overview of Virtex-II FPGA

The architecture of Virtex-II FPGA is shown in Fig. 3.3. Multiple I/O Blocks (IOBs) fit into either height of one row or width of one column. It adopts a programmable architecture that provides connectivity to various signaling standards, such as CMOS, LVTTL, SSTL, HSTL, and so on. It also equips registers for a DDR input and output.

The CLBs are arranged in a grid, and the interconnected channels are laid out in between. Some stripes of memory blocks called BlockRAMs and embedded 18x18-bit multipliers are longitudinally arranged among the CLBs. These hard macros of memories and multipliers prevents increase of circuit area and contributes to the improvement of its maximum frequency.

The architecture of Virtex-II Pro series is similar, with accession of a built-in PowerPC 405 processor and RocketIO multi-gigabit transceivers (MGTs). These components construct control microprocessors and high-speed serial communication interface on a single chip, to perform great advantage for applications with frequent I/O data transfer and high-speed process.

3.3 Applications in molecular dynamics

This subsection introduces some examples of high performance computing systems on FPGAs.

3.3.1 Advantages of FPGA-based architectures

Currently, dedicated computing systems are major solutions for accelerating operations in sci-tech applications, as represented in GRAPE [36]. However, they are fraught with low flexibility toward other operations and high development cost.

On the other hand, FPGAs consist with the advantages of dedicated hardware and the flexibility of software. Increasing logic capacity allows implementation of applications with floating-point operations, triggering the rise of commercial computers with FPGAs as coprocessor. The following

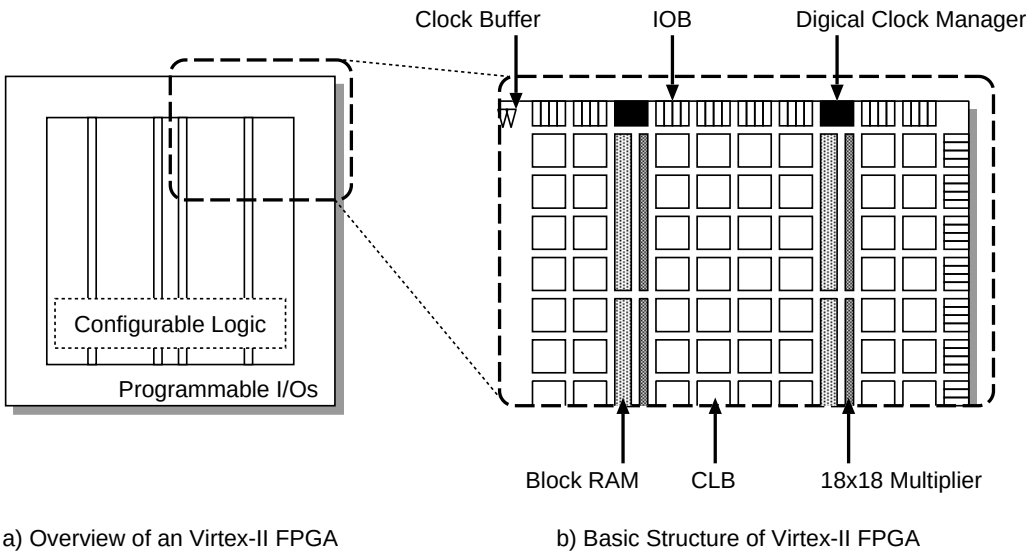


Fig. 3.3: Architecture of Virtex-II FPGA

subsections provide one of the examples of FPGA-based commercial computers and its application to bioinformatics.

3.3.2 PROGRAPE-3 (Riken/Tokyo Electron Device)

PROGRAPE (PROgrammable GRAPE), developed by Tokyo Electron Device, Chiba University and Riken (Fig. 3.4), is an accelerator for numerical simulations, especially for many-body problems such as intermundane gravity simulations. It is a derivative of GRAPE project; the crucial difference is that the computational pipeline originally on an LSI is implemented on FPGA chips in the case of PROGRAPE. The PROGRAPE board called Bioler-3 is a PCI card with four Xilinx Virtex-II Pro FPGA (XC2VP70-6), and is dedicated to calculate the interactions between particles while the host PC performs all other calculations. According to the recent study by the implementers, 236 GFlops is achieved for solving a forementioned problem [37, 38].

3.3.3 Molecular dynamics simulation

A research on a complex cubic structure and function of proteins based on molecular dynamics (MD) is a crucial part of computational chemistry, by simulating intermolecular forces between a protein and other molecules. MD simulations are one of the classical approaches to biochemistry and other studying fields. A recent study presented that an implementation of MD on an FPGA are effective compared to implementations on existing solutions as represented in parallel systems or special purpose hardware such as PC clusters or MD-GRAPE, in terms of acceleration and capability of problem size [39].

3.4 Related works : Stochastic Biochemical Simulator on an FPGA

3.4.1 Overview

Since 2004, there were several studies to implement and evaluate for stochastic biochemical simulator on FPGAs.

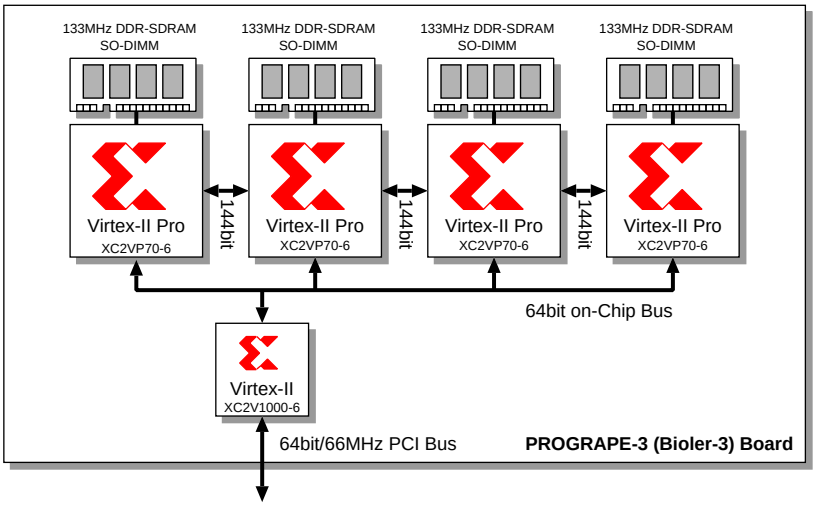


Fig. 3.4: Architecture of PROGRAPE-3 (Bioler-3)

1. Keane *et. al.* developed hardware generation system which loads biochemical model as an input [40], and reports its performance. Their algorithm is modified to run on an FPGA proposed by Salwinski and Eisenberg. Lok also discussed the possibility of FPGA accelerating stochastic biochemical simulation by appropriate modification of SSAs [41, 42].

2. Thurmon *et. al.* implemented an accelerator for a part of SSA, and evaluated it on a real FPGA-system [43].

3. This thesis is also based on a background by our researches in a part of ReCSiP project from 2004.

In this section, four studies mentioned above are introduced in detail, and discuss their performance estimation, advantages and disadvantages.

3.4.2 Keane's approach

Keane *et. al.* implemented and evaluated an FPGA-based stochastic biochemical simulator generation system [40]. They implemented a modified version of SSA to adjust to an execution on an FPGA. Additionally, research group of Lok and Salwinski also proposed a similar idea of Keane's approach [41, 42].

They focused that a number of the reaction occurence follows Poisson distribution with its propensity as a parameter derived from Gillespie's τ-leap method. The number of reaction in a time step is derived by Eq. 3.1.

$$S_1 + S_2 \xrightarrow{k} S_3 \Longrightarrow \text{Poisson}(k \cdot X_1 \cdot X_2) \tag{3.1}$$

Eq. 3.1 can be approximated to Eq. 3.2 because a number of reaction occurence is one at most by applying adequately small time step Δt.

$$S_1 + S_2 \xrightarrow{k} S_3|_{\Delta t} \Longrightarrow \text{Bernoulli}(k \cdot X_1 \cdot X_2 \cdot \Delta t) \tag{3.2}$$

Probability p of Bernoulli process is determined by multipling probability p_i of Bernoulli process as $p = p_0 \cdot p_1 \cdots p_i$. As multiplications are regarded as a logical AND, Eq. 3.2 can be rewritten as

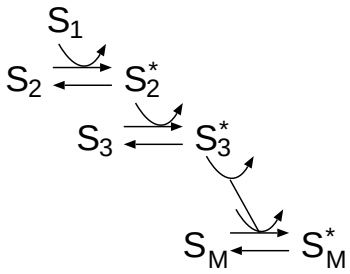


Fig. 3.5: Cascade model evalulated in Keane's system

Eq. 3.3.

$$\begin{aligned} P\left[S_1 + S_2 \xrightarrow{k} S_3\right]|_{\Delta t} &= P_0 \cdot P_1 \cdot P_2 \\ &= P[r_0 < k] \text{ AND } P[r_1 < X_1] \text{ AND } P[r_2 < X_2] \end{aligned} \tag{3.3}$$

In Eq. 3.3, variables r_i are uniform random numbers. Eq. 3.3 can be implemented on an FPGA as a combination of random number generator and counter for species. Furthermore, reactions in the model can be evaluated in parallel without floating-point calculation. An interval of time step is variable according to the number of reaction occured in recent time step.

Their simulator generator compiled the simulation hardware from common hardware modules which evaluates a reaction for 3 clock cycles and their interconnection by loading the model written in SBML. A simulator is configured so that the structure would contain the same number of reaction modules as reactions defined in the model.

The signaling cascade model is shown in Fig. 3.5. It can be extended easily when it is used to evaluate the performance of generated hardware. As shown in Table 3.1, the performance of Keane's FPGA-based system outperforms about 20 times compared to execution on 2.0 GHz Pentium4. However, the generation system has following problems. The logic resource of generated hardware requirement is propotional to the number of reaction defined in the model. Therefore, hardware resource for a model with up to about 120 reactions exceed the resouce of the largest FPGA of the time (Xilinx's Virtex-II XC2V8000). Moreover, as the unit of calculation is modified to time step from originally event-driven algorithm of SSA, the performance gain may not be easily applied to other biochemical models.

3.4.3 Thurmon's implementation

To deal with the problem of increasing logic resource, Thurmon *et. al.* proposed another approach to accelerate SSA using FPGA [43].

Table 3.1: Performance of Keane's system compared with the NRM on a 2.0GHz Pentium 4

Reactions	1	4	8	16	32
Species	2	8	16	32	64
Events/sec hline FPGA	2,711,405	3,052,482	4,695,500	7,572,711	7,494,402
Events/sec hline 2.0 GHz Pentium4	333,333	160,000	256,410	285,714	320,000
Gain	8.1	19.1	18.3	26.5	23.4

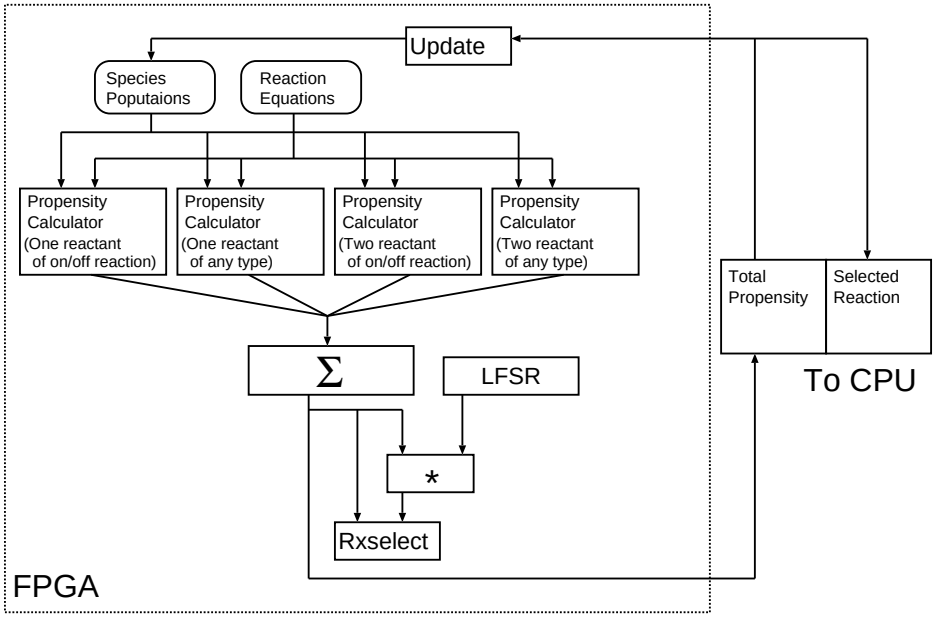


Fig. 3.6: Thurmon's hardware design

In a reaction cycle of SSA, the most frequently used operation is a propensity function, as shown in equations in Table 2.3. Thus, they off-loaded this part to an FPGA.

They implemented FPGA-based DM calculation system using pipelined multipliers and an accumulator. For hardware operation, they converted floating-point arithmetics to integers. They argues that lack of accuracy can be prevented by selecting appropriate bit-width for calculation.

Fig. 3.6 shows the design of Thurmon's implementation. A propensity is calculated by chosing a propensity calculator according to the shape of the reaction from population as input. All propensity are calculated by pipelined hardware modules to increase the throughput.

Thurmon *et.al.* evaluated their implementation on a memory-slot connected computing system equipped FPGA called Philchard [44]. They adopted two simulation models, Self-Regulated Model ($M = 14$) and Genomically Based Oscillation Model ($M = 16$) to compare its performance with a microprocessors. Table 3.2 shows execution time of the result of their implementation compared with four algorithms by Pentium III 1.0 GHz microprocessor. Although their implementation outperforms all algorithms on microprocessor, the performance is not so higher than performances with Pentium III, which is already categorized in earlier generation.

Causes of the supression of performance gain may be that as an FPGA on Pilchard, which uses Xilix's Virtex XCV1000E, is also earlier generation, efficient logic resource could not be provided to parallel execution. The total sum of propensity a_0 and selected reaction ID is transfered to host-PC to record the simulation. The data communication between FPGA and host-PC may also be a cause of supression for performance gain.

3.5 Our previous work

A study in this thesis began as a part of ReCSiP project, which stands for Reconfigurable Cell Simulation Platform [45]. The ReCSiP project which began in 2001 is aims to develop an accelerator for general purpose ODE-based simulation models.

In 2004, FPGA-based DM calculator was implemented and evaluated [46]. Although it was limited to Lotka system as it was preliminary implementation, the throughput was about 105.13 times

higher than execution on Athlon 2800+. Main difference from other related studies is that multiple simulations could be executed in parallel with pipelined floating-point arithmetic modules. However, hardware must be re-implemented when it runs other models. To addition of this problem, the logic resource for simulator expands according to the number of reaction defined in the model as same as Keane's implementation. In this section, the hardware structure of our previous implementation and its evaluation to establish problems.

3.5.1 Analysis of Lotka System

To implement SSA on an FPGA, algorithm of DM is analyzed when the Lotka system is executed. The Lotka system is well-suited for evaluation of Gillespie's algorithm implemented on FPGA because the Lotka model is relatively small model and pululations are fluctuated around the certain values.

Followings show a cycle of process in the Lotka system. It assumes that initialization has been done.

Step 1. h_j is stored in a value which is combination of populations of reactants in the current system for each reaction

$$h_1 = X_1 \cdot X_2 \tag{3.4}$$
$$h_2 = X_1 \cdot X_3 \tag{3.5}$$
$$h_3 = X_3 \tag{3.6}$$
$$h_4 = X_2 \tag{3.7}$$

Step 2. propensity a_j and the sum of them a_0 are substituted with multiplication of h_ν and c_ν, and the sum of a_ν

$$a_j = h_j \cdot c_j \qquad (j = 1, \cdots, 4) \tag{3.8}$$
$$a_0 = \sum_{j=1}^{4} a_j \tag{3.9}$$

Step 3. Two new random numbers r_1 and r_2 are generated. $1/\tau$ is calculated with multiplication of $1/\ln(1/r_1)$ and a_0. μ is determined by comparing a_j with $r_2 a_0$

$$\frac{1}{\tau} = \frac{a_0}{1/\ln(1/r_1)} \tag{3.10}$$

$$\sum_{j=1}^{\mu-1} a_j < r_2 a_0 \leq \sum_{j=1}^{\mu} a_j \tag{3.11}$$

Table 3.2: Performance of Thurmon's system compared to a 1.0GHz Pentium III

	Self-Regulated Model ($M = 11$)		Genomically Based Oscillation Model ($M = 14$)	
SSA	Exection Time	Speedup	Exection Time	Speedup
DM-HW	77.798	1.00	78.259	1.00
FRM	814.033	10.46	805.044	10.29
DM	225.114	2.89	230.558	2.95
NRM	174.656	2.24	252.125	3.22
ODM	109.410	1.41	118.948	1.52

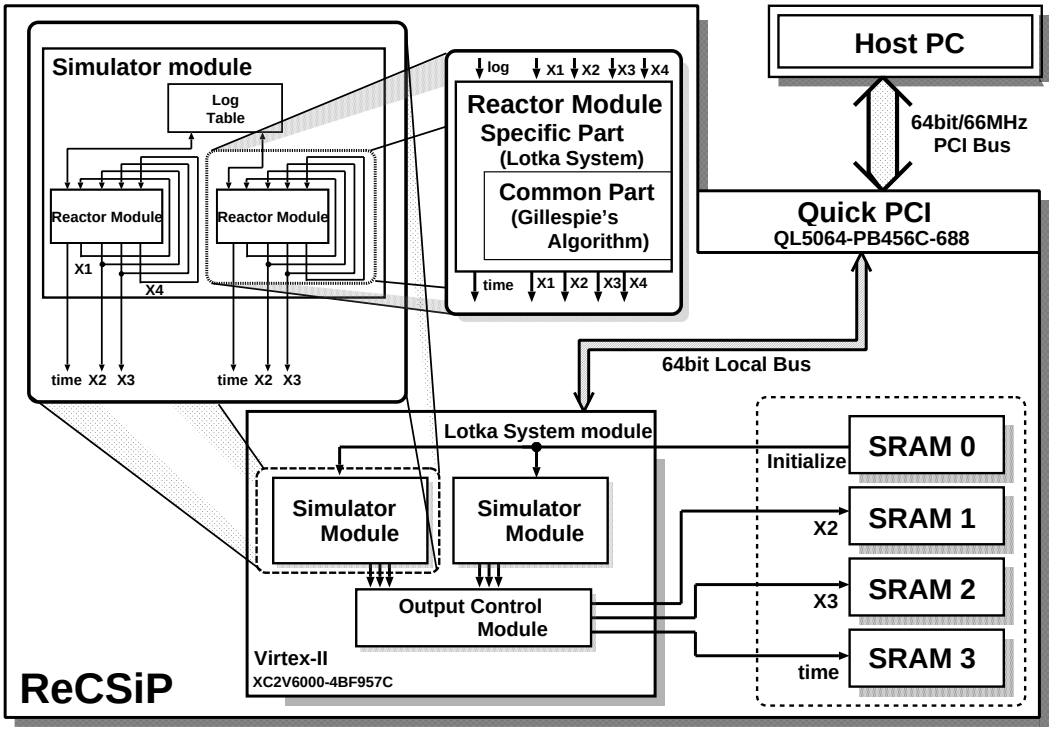


Fig. 3.7: Structure of the Lotka System Module

Step 4. Populations are modified by R_μ

Step 5. Modified populations $X'_1, \cdots, X'_4$ are referred by the next cycle of Step 1. Simulation time t is updated by adding τ.

Step 6. Return to Step 1.

3.5.2 Overview of the Simulator

Section 3.5.2 introduces design and implementation of the Lotka system for Direct Method on ReCSiP-board.

Fig. 3.7 shows a hardware design of DM simulator for the Lotka system. The reactor module, which performs Step 1, 2, 3, and 4 in the Section 3.5.1, is parallelized and controlled. The Lotka system module consists of two simulator modules and an output control module. Each simulator module has two reactor modules as the core of simulator.

A simulator module starts calculation when it receives the random seed and initial value of molecule numbers stored in the SRAM module. The results from two simulator modules (four reactor modules) are stored into the SRAM modules through the output control module.

3.5.3 Simulator Module and Reactor Module

Each simulator module simply consists of two reactor modules, which share a logarithmic table. A reactor module processes a cycle of the Lotka system with Gillespie's algorithm. In the first step of the process, the module receives the molecule numbers, $X_1, \cdots, X_4$. Then, it processes a cycle of the Lotka system with Gillespie's algorithm. Finally, it outputs τ and molecule numbers $X'_1, \cdots X'_4$ after the reaction. Molecule numbers $X'_1, \cdots X'_4$ are used in the next input of reactor module. As X'_2, X'_3 and

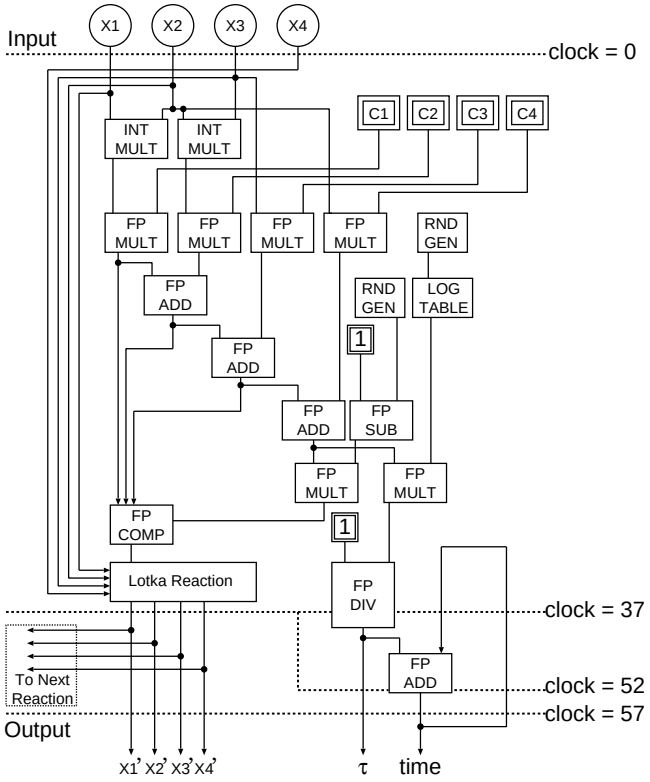


Fig. 3.8: Data-flows in Reactor Module

τ are required for the evaluation of the simulation result, they are transferred to the output control module.

The reactor module consists of two parts; the common part for Gillespie's algorithm, and the specific part for the Lotka system. By separating them, the common part can be used for the other reactor modules. That is, by replacing the reactor module in Verilog-HDL code level, other reactions can be simulated.

The common part calculates τ and μ, and also generates random numbers. The target-specific part processes the combinations of reactions, then increases or decreases numbers of molecules according to the result of calculated μ.

A reactor module has four kinds of major functional units, which are two 32bit integer multipliers, five single-precision FP adders, six single-precision multipliers, and a single-precision FP divider.

- Single-precision FP multiplier includes 18×18 bit dedicated multiplier blocks (distinct features of Virtex-II)
- Single-precision FP divider uses 2bit-base subtract-and-shift divider
- 32 bit LFSRs (Linear Feedback Shift Registers) are used for the random number generation with M-sequences

Fig. 3.8 shows the flow of calculation in the implemented reactor module. The reactor module has 37 pipeline stages. It takes $X_1, \cdots, X_4$ as inputs, then outputs populations $X'_1, \cdots X'_4$ and τ after a reaction through step 1, 2, 3, and 4 described before. The output takes 37 clocks for $X'_1, \cdots X'_4$, and 52 clocks for τ. So, it is possible to execute 37 independent simulation processes at a time. These

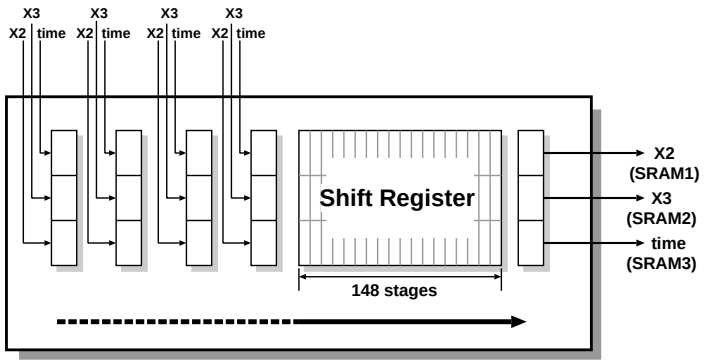


Fig. 3.9: Structure of Output Control Module

multiple concurrent executions are advantageous in this kind of stochastic simulation which returns an average of the iterative executions.

Values of $1/\ln(1/r_1)$, which are required to calculate τ, are stored in 32bit width tables whose depth are 2^{15}. The tables are implemented on Block RAM on Virtex-II. τ is derived with multiplication of the fetched number and a_0 (the sum of the reaction probability of each reaction a_ν).

The reactor module has 32bit-width 37 entries shift registers which store the total simulation times t. Each register updates value by adding previous t and τ at the end of a cycle.

3.5.4 Output Control Module

The Lotka system module has two simulation modules each of which has two reactor modules, and in a reactor module, 37 simulation processes are executable in parallel. Therefore, maximum 148 simulation processes are "on-the-fly" in the whole the Lotka system module. Total simulation time t, number of prey species X_2, and number of predator species X_3 are generated with the simulation of the Lotka system. These values are represented in 32bit integer or single-precision floating-point. As a result, four sets of three 32bit data is generated from the Lotka system module with each cycle.

Output control module manages data transfer from the simulator module to SRAM modules. The module works as follows.

- A set of three 32 bit data is transferred from reactor modules to SRAM modules with an arbitrary cycle interval
- FIFO buffer for temporary storage of output data is provided. It begins to store data from the reactor module in the specified cycle

Inputs of the output control module are four sets of three 32 bit data which are t, X_2 and X_3 per clock. The module transfers them to SRAM modules with an arbitrary interval (that is, interval must be nothing less than 5). Fig. 3.9 shows the structure of the output control module.

3.5.5 Evaluation

(1) Result of Synthesis and Place & Route

Table 3.3 and Table 3.4 show required resources and maximum operation frequency for the Lotka system module.

Modules are described in Verilog-HDL. Synthesis and place & route are done with Xilinx ISE6.1i. The target device is Xilinx's XC2V6000-4BF957C, which is equipped on the ReCSiP board.

Table 3.3: Resource Utilization

Slices	18x18 Multipliers	18kbit BlockRAM
26091 (77.21%)	120 (83.33%)	132 (91.67%)

Table 3.4: Performance

Frequency [MHz]	Throughput [cycle/sec]
76.25	304.88M

In order to optimize the clock frequency, this implementation is somehow specialized to the Lotka system. So, it can not be extended for larger scale systems without modifying the Lotka system directory in the current implementation.

Some simulation results are shown in Fig. 3.10 and Fig. 3.11. These are results after 500,000 cycles of executions with ten output intervals. The difference between them is caused only by the random seeds, and the conclusive simulation result of the Lotka system is obtained with taking an average of them by appropriate time interval.

(2) Accuracy Verification

The accuracy and performance of the simulator on ReCSiP is compared with a software simulator on common PCs.

Since the floating-point number arithmetic units on ReCSiP are not based on rounding algorithm in the IEEE standard, its influence must be examined. In calculation of τ, there is no error propagation because the output is not used as the input of the next cycle. However, the result of μ is used to determine what the next reaction occurs. In this case, the accumulation of rounding errors may cause a problem.

To examine the effect of rounding errors, the same simulator including an M-sequence generator written in C is executed with the same random seed. Fig. 3.12 and Fig. 3.13 show the result of software execution and result from ReCSiP board. They are measured after 100, 000 cycles of execution, and the output interval is 10. Trajectories of population X_2 and X_3 are similar both in software and in hardware execution. This shows that useful simulation results can be obtained from the hardware execution on ReCSiP board.

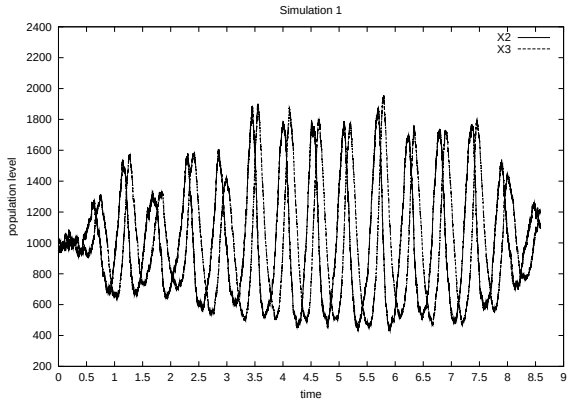


Fig. 3.10: Example of a result

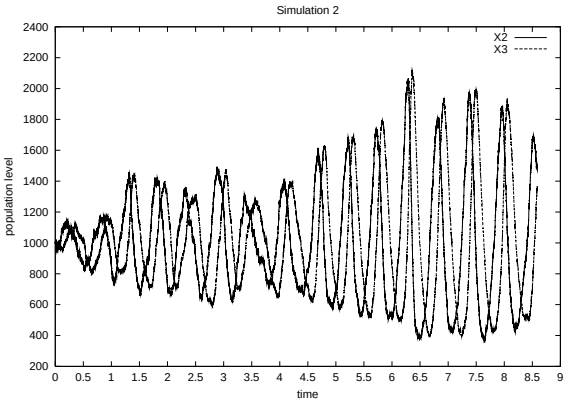


Fig. 3.11: Example of another result

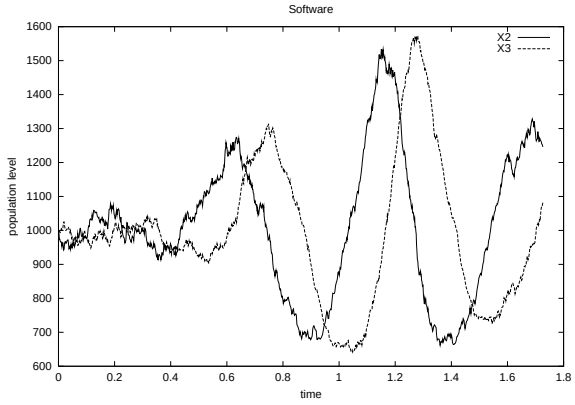


Fig. 3.12: Execution by C Code

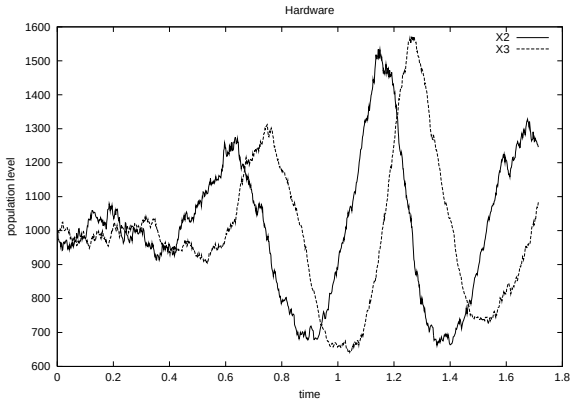


Fig. 3.13: Execution on Hardware

Table 3.5: Throughput of the Software

Processor	Memory	Environment	Run-time [μsec]	Throughput [cycle/sec]
AthlonXP2800+ 2.0 GHz	2 GB	Free BSD 4.8 +gcc2.95.3	172,226	2.90M
Xeon 2.8 GHz Dual(HT off)	4 GB	Linux2.4.21 +gcc2.95.3	214,219	2.33M
UltraSPARCIIIcu 1.2 GHz	4 GB	Solaris8 +gcc2.95.3	555,907	0.90M

(3) Performance Evaluation

Table 3.5 shows response times and performance for 500,000 cycles of the software simulation of the Lotka system. In the software execution, the program is compiled with -O3 option. Throughput S is derived from the equation $S = 0.5$[Mcycle]/run-time [μsec].

The implemented Lotka system simulator can generate a result at intervals of 37 clocks, and the maximum operation frequency is 76.25 MHz. Accordingly, by equation 76.25 [MHz]/37 = 2.06 [Mcycle/sec], the throughput of the single simulator is 2.06 [Mcycle/sec].

The implemented reactor module is 37-stage pipelined, and four reactor modules can be mounted (on a Lotka system module). Thus, this simulator is able to execute 148 simulations simultaneously in a clock. As the result, the maximum throughput of the Lotka system simulator is 304.88 [Mcycle/sec]. It is about 105 times faster than the software implementation on AthlonXP 2800+ (operating at 2.0 GHz).

ReCSiP is connected to a host PC with 64 bit/66 MHz PCI bus. If the Lotka system module transfers to SRAM modules by ten output intervals, total throughput of output data is 365.856 [MByte/sec]. That is, each simulator outputs three 32bit data, X_2, X_3, and t in every cycle, and 148 simulations are executing at a cycle. The performance of the Lotka system module is 304.88[Mcycle/sec]. If the output interval is assumed to be ten, data throughput becomes 365.856 [MByte/sec]. 64 bit/66 MHz PCI bus has adequate bandwidth to transfer the amount of data derived above.

Chapter 4

Implementation of First Reaction Method on an FPGA

4.1 Design concept to solve previous problems

4.1.1 Floating-point arithmetic

Any approximation and conversion are not preferred when a simulator is configured on an FPGA. Therefore, arithmetic operations are configured faithfully to the original algorithm by single-precision floating-point data. As floating-point arithmetic requires long clock cycles to acquire a result, it is difficult to achieve high-throughput compared to execution on microprocessors whose operating frequency is more than tenfold of an FPGA.

However, as shown in Section 3.5, SSA includes loop-level and thread-level parallelism. So, SSA can be overcome severalfold performance of a microprocessor by exploiting efficient use of pipelining and parallel execution.

4.1.2 Scalability for large-scale biochemical models

The simulator is required to simulate various biochemical models without compiling and reconfiguring hardware modules. Logic resource of the simulator is not so relevant with the size of models.

To approach this problem, simulation data is stored in embedded memories of an FPGA. For example, Xilinx's FPGA has distributed memories called BlockRAM which is 18bit $\times$ 1024words. Amount of data depends on the size of the model with different population, reaction rate constants and state-update vector. However, amount of logic resource can be kept stable by by storing these data into combination of BlockRAMs. Through reaction type and species IDs of its reactants are also stored into BlockRAMs, so re-compiling hardware module is not required even when target model changes.

4.1.3 Multi-thread execution

As the hardware structure is fixed, the clock cycles to calculate a reaction cycle can be obtained deterministically if model size is known.

As long as circuit size of the simulator does not exceed the capacity of an FPGA, multiple simulators can be configured for better performance. Simultaneous multiple simulations are legitimate for Monte-Carlo simulation because the biochemical model have to be simulated thousands of times with different random seed to acquire statistically-significant number of samples.

List 4.1: Core program in FRM

```
1  for(i=0;i<M;i++){
2    r = random();
3    p[i] = propensity(i, X);
4    tau[i] = (1/ln(r)) / a[i];
5  }
6  mu = min(tau);
7  update(mu, X);
8  t += tau[mu]];
```

Table 4.1: Variables in List. 4.1

r	uniform random number between $(0, 1]$
X	integer array for state vector $\boldsymbol{X} = \{X_0, \cdots X_N\}$
a	float array for propensity $\boldsymbol{a} = \{a_0, \cdots a_M\}$
tau	float array for putative time of each reaction $\boldsymbol{\tau} = \{\tau_0, \cdots \tau_M\}$
mu	reaction ID which occurs next

4.2 Acceleration concept for FRM-FPGA

This section depicts the implementation of FRM simulator module described in Section 4.1. First of all, a core part of FRM program-code is shown in List. 4.1.

List. 4.1 is a whole code in a reaction cycle of FRM, and its variables are shown in Table 4.1. As there are no data dependency between ith loop and $(i + 1)$th loop in List. 4.1, all loops can be executed in parallel. It means that loop calculation can be issued sequentially into the pipeline of a hardware module to calculate the operation inside the loop.

Suppose a operation in the loop takes p clock cycles, the time for the loop calculation is $(p \times M)$ clock cycles when the calculation advances sequentially. By exploiting pipelining technique, operation time is shortened to $(kp + M)$ clock cycles. Variable k is a pipeline pitch which is the acceptance interval of the hardware module.

When FRM is executed on hardware operation, operations in the loop may be implemented as pipelined hardware module or parallel execution on multiple hardware modules. These two approaches can be used simultaneously. We adopted the former approach, the pipelining technique. It means that a simulation process is executed in a hardware module which operates a calculation in the loop. We also adopted the latter approach as described before. Simultaneous multiple simulations are legitimate for parameter searches and also for Monte-Carlo methods.

Fig. 4.1 shows an example of data-flow of line 1 to 6 of List. 4.1 using three pipelined hardware modules: propensity calculator, τ-calculator and "getting-minimum". We assume these modules are all pipelined and their pitch is one clock cycle. The intermediate data, the model state which is a list of populations in this case, is stored in BlockRAM, and they are read in order according to reactants in each reactants. The read numbers of species are thrown into the propensity calculator module to obtain its propensity a_j by Table 2.3. The obtained a_j is directly input to the τ-calculator module. The getting-minimum module receives the output of the τ-calculator module to obtain the minimum value τ_μ and its index μ. Only getting-minimum module is not pipelined and it takes a pocket of time as comparing values sequentially thrown into the module. The hardware module with this structure can execute a calculation of the loop in List. 4.1 for $(kp + M)$ clock cycles.

After this section, details on actual implemented FRM hardware modules are explained.

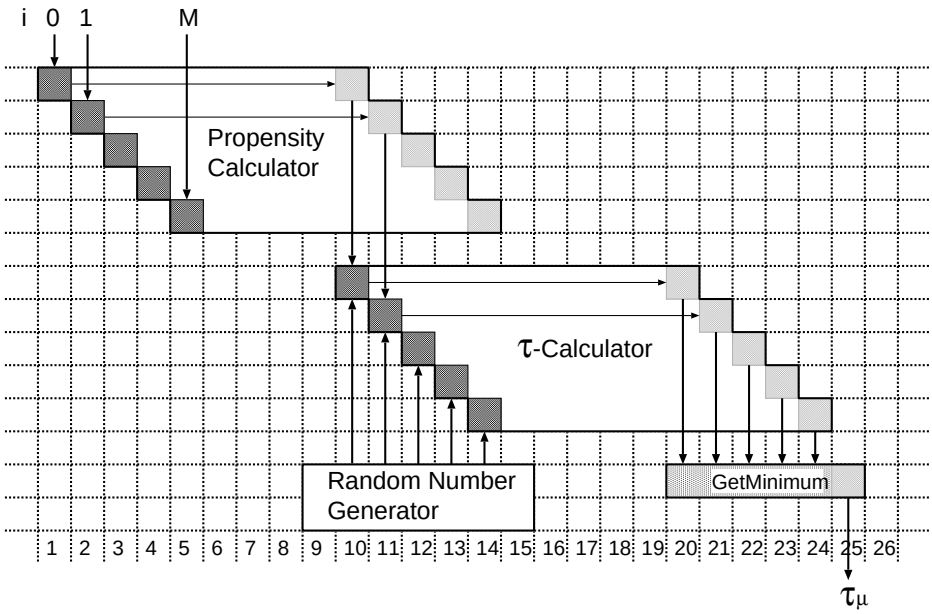


Fig. 4.1: Pipeline execution of List. 4.1

4.3 Implementation

4.3.1 A structure of FRM-FPGA

Fig. 4.2 gives the overview of the FRM simulator module called FRM-FPGA. It has three identical simulation units called FRM-UNITs that can operate in parallel. The number of FRM-UNITs is determined by the capacity of FPGA's logic resources. An FRM-UNIT has following three units: (1) a functional unit (FU) for grouping some pipelined FP operation units, (2) a data unit (DU) for intermediate simulation data storage, and (3) a controller unit (CU) controls data flow between DU and FU. A DU contains two independent datasets named (DS_A, DS_B) for different simulation tasks. Therefore, FRM-FPGA executes totally six independent simulations in parallel.

Fig. 4.3 shows an example of data-flow in a FRM-UNIT when simulation for a small biochemical model which defines five reactions($M = 5$) is executed.

A calculation for a reaction cycle is progressed as following scheme: Phase numbers at each head of the item in the list indicate the region in Fig. 4.3.

Phase 1. CU has a Block-RAM as a ROM (Read-only memory) to store the reactants species IDs of each reaction. ROM means that the data in the BlockRAM is not updated while a simulation is running. CU reads out the species IDs of reactants from R_1 in ascendant order, and outputs them to the DS_A. As the number of reactants is up to two, one clock cycle is enough to read data of a reaction using dual-port BlockRAM. After reading out the species ID of R_M, CU switches the destination dataset to DS_B from DS_A and reads again from R_1.

Phase 2. DU reads out populations of reactants from appointed DS according to their IDs transferred from CU. Populations are transferred to FU.

Phase 3. τ-unit in FU received populations calculates τ_js sequentially. μ-unit also searches τ_js to obtain τ_μ. This computation is similar scheme with Fig. 4.1.

Phase 4. μ and τ_μ are determined at next clock cycle after the moment when τ_N is thrown into μ-unit. FU transfers τ_μ and update-vector of μ to DU.

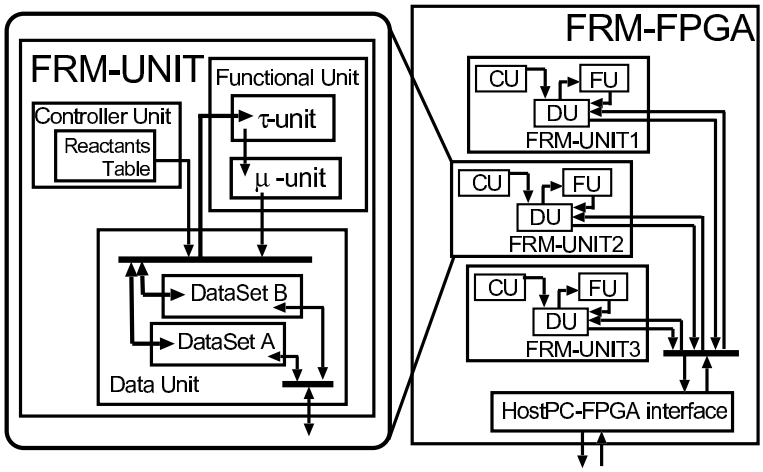


Fig. 4.2: Structure of FRM-FPGA

Phase 5. DU updates simulation time t and population $\boldsymbol{X}$ according to τ_μ and ν_μ from FU.

Phase 6. After both computation of updating $\mathrm{DS_A}$ and sending species IDs of reactants to $\mathrm{DS_B}$, CU returns the control to Phase 1.

The simulation result is adjudicated to send back to the host-PC at the end of every reaction cycle. In this implementation, simulation time t and all populations are transferred to memory in host-PC via DMA operation.

Hardware implementation of FRM-FPGA is written in Verilog-HDL. Details of each module are explained following subsections.

4.3.2 CU: Controller Unit

CU is a hardware module reads out reactants IDs of each reaction and transfers them to the FU to calculate τ_j at the beginning of the reaction cycle. A structure of CU given in Fig. 4.4 which is consisting of following components:

1. Reactants Table
2. Reaction Number: a register which is stored the number of reaction defined in the model
3. Reaction ID Counter: a counter which indicates a reaction ID to read out reactants
4. a flag to specify which data set is current process

A “Reactants Table” stores the Reactants of each reaction. As both theoretical maximum numbers of reaction and species are 1024 which can be expressed 10-bits, and reactants in a reaction is consisting of up to two species, Reactants Table is composed by 10 bits entry with 20bit words of a dual-port BlockRAM.

A “Reaction ID Counter” is used as an address to read out reactants from the reactants table. It is in charge of incrementing index μ after each reaction cycle. Both “Data Unit Selector” and “Reaction Number” are flags for dataset assignment. The whole unit of one CU serves as pointers to the species required in propensity calculation for reaction R_j. In the beginning of each reaction cycle, indices of species for all $R_1, \cdots R_M$ are read from the Reactants Table. This operation is repeated sequentially at every clock cycle until all the indices of species for M reactions are obtained, and they are given to the Data Unit (DU) together with the index of reaction. Then the value of the flag switches and

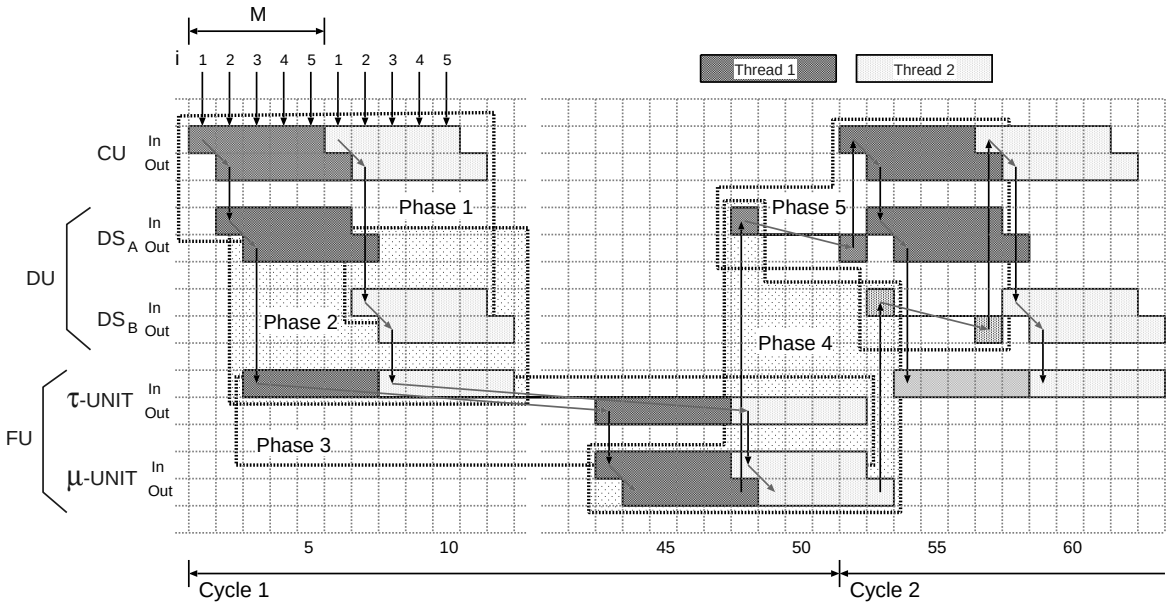


Fig. 4.3: An example of pipeline execution

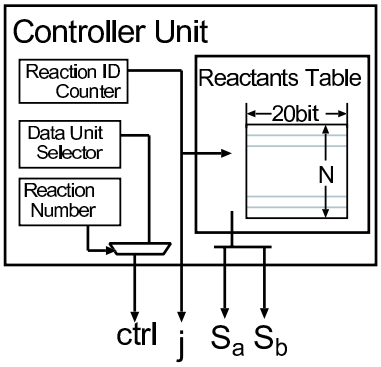


Fig. 4.4: Block diagram of Controller Unit

points the other dataset in DU. After updating all the number of species in DS_A, DU reads an index of species for the next reaction cycle.

Reactants of each reaction are stored into the Reactants Table. With the beginning of the reaction cycle, reactants of reactions are read out every clock cycle. Read reactants data is transferred to DU. After reading reactants from R_1 to R_M twice for $\mathrm{DS_A}$ and $\mathrm{DS_B}$, CU waits for the completion of state update of $\mathrm{DS_A}$. After that, CU starts the reading in next reaction cycle.

Reading reactants takes $2M$ clock cycles because two simulation processes are run at a time.

4.3.3 DU: Data Unit

Fig. 4.5 shows the structure of a DU. DU stores intermediate data for two simulation threads, each simulation thread includes as follows :

- population of species $\boldsymbol{X}$, which is consist of dual-port BlockRAM with 32-bit ×1024
- current simulation time t

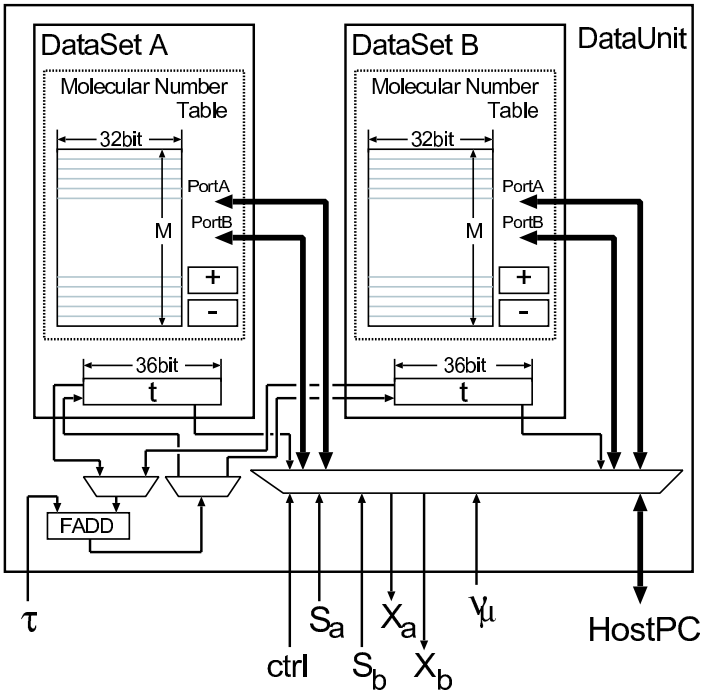


Fig. 4.5: Block diagram of Data Unit

DU has two functions: (1) reading population and (2) updating population. In reading population phase is indicated as “Phase 2” in Fig. 4.3. The scheme of this phase is as follows:

1. Receiving dataset assignment flag and reactants from CU
2. Reading out populations from assigned dataset
3. Transfering populations to FU

In updating population phase is indicated as “Phase 5” in Fig. 4.3. The scheme of this phase is as follows:

1. Receiving a state-update vector ν_μ and time gain τ_μ from the FU
2. Updating population according to the state-update vector ν_μ
3. Updating current simulation time t by a floating-point adder

It takes 1 clock cycle to read population from a data set module, and it also takes 4 clock cycles to update population after receiving $\boldsymbol{\nu_\mu}$ from μ-unit in FU. As simulation current time t is not used following reaction cycles, it is updated with the calculation of next reaction cycle.

4.3.4 FU: Functional Unit

An FU is an calculation unit to obtain τ_j from population of reactants in a reaction by floating-point(FP) operations. It consists of two components: a τ-unit and μ-unit. τ-unit calculates the putative time τ_j for the next occurrence of R_j. Meanwhile, μ-unit searches the minimum value and its index from $\boldsymbol{\tau}$. Fig. 4.6 and Fig. 4.7 are the structures of these units.

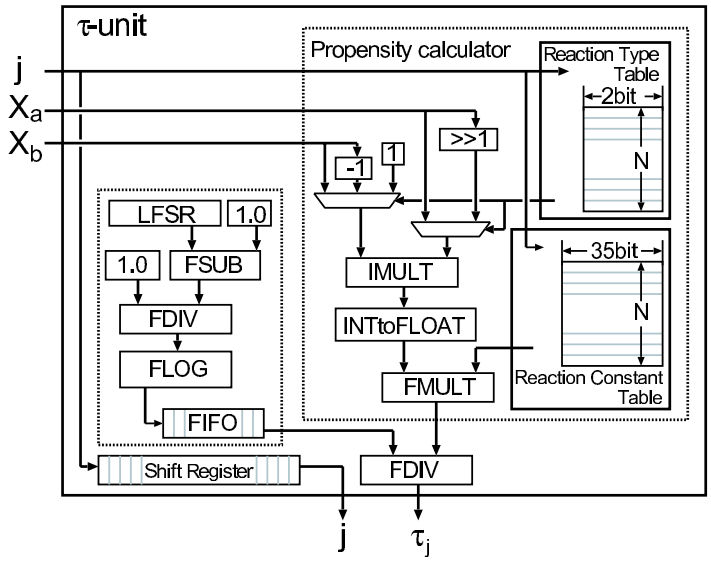


Fig. 4.6: Structure of τ-unit

(1) τ-unit

τ-unit calculates τ_j from reaction ID j and its molecular numbers (indicated as "Phase 3" in Fig. 4.3). It has a deeply pipelined structure for FP operations. The calculation is the same as Eq. 2.10.

τ-unit has two types of memory: a "Reaction Type Table" and a "Reaction Constant Table". Data in the former memory is used for selecting a calculation method (reaction type) for obtaining the propensity for reaction. Three methods are prepared to deal with target biochemical systems which are bimolecular reaction or under. The latter memory stores stochastic reaction rate constants c. Combinatorial number (e.g. $X_1 \times X_2$) becomes an integer value, since molecular numbers are given as integers. After combinatorial numbers are calculated by an internal integer multiplier (IMULT), they are transformed into FP format through an "INTtoFLOAT" transformation unit. Because there are some pipeline latency in floating operations, τ-unit utilizes a shift register for reading out the value j simultaneously with τ_j.

It takes 42 clock cycles to calculate τ_j after reading molecular numbers, but high throughput is obtained by filling up the pipelines in FU and executing two data sets in parallel.

The term $\ln(1/r)$ in Eq. 2.10 is independent from the calculation for propensity. Therefore, there is a FIFO which buffers values of $\ln(1/r)$ to cut down the calculation time. After dequeuing FIFO, τ-unit asserts a request flag to generate next random number to calculate $\ln(1/r)$ which is enqueued the FIFO. This scheme can use random cycle efficiently, while it makes the verification of the result easy.

The random number generator is required to generate 26-bit random bit every clock cycle. In this implementation, 167-bit of Lenear Feedback Shift Register (LFSR) is used for the random number generator with M-sequence. Even though RPG100 equipped on ReCSiP2-board, the physical random number generator, can also be used, it may be utilize to replace a part of the random sequence because the generation frequency is 250 kbps the maximum.

Important values in stochastic simulatoin are the state after certain simulation time or the margin of fluctuation for population. The pseudo random number generation method which does not show obvious linearity is said that it is sufficient for stochastic simulation. Various studies for random number generator on an FPGA make advance such as Merssenne-Twister algorithm and other hardware-dedicated algorithm [47, 48]. The random number generator in τ-unit can be superseded

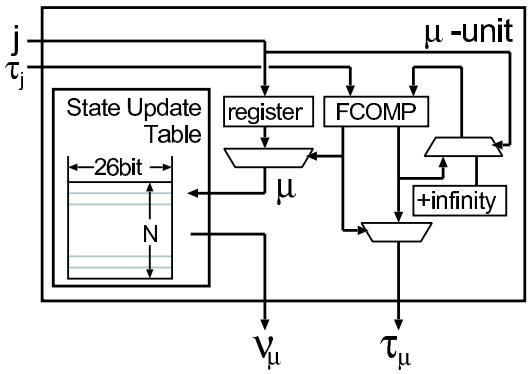


Fig. 4.7: Structure of μ-unit

Table 4.2: Logic resouces and maximum operating frequencies for FP arithmetic

	latency	Slices	Multiplier	RAM	[MHz]
Addr/Subtracter	6	663	-	-	134.6
Multiplier	8	246	4	-	196.6
Divider	17	1542	-	-	106.7
Comparator	1	48	-	-	119.2
Logarithmic function	32	2276	6	13	109.2

by hardware modules with other generation algorithms.

(2) μ-unit

μ-unit searches the minimum value from data stream inputed every clock cycle, and output the minimum value and its index of the stream.

μ-unit includes floating-point comparator (FCOMP) and a register to hold the reaction ID j, and "State Update Table" which is stored state-update vector ν_j for each reaction. FCOMP is a simple comparator which outputs smaller value and a flag which indicates the change of output value with the data output before one clock cycle from a couple of input value.

The following is the mechanism for searching the minimum value of τ_j within $\boldsymbol{\tau} = (\tau_1, \tau_2, \cdots, \tau_M)$. First, the μ-unit stores values of τ_j sequentially given from τ-unit. Then with an internal FP comparator (FCOMP), the minimum value τ_μ is selected between τ_1 and τ_M. Note that this operation takes M clock cycles. Finally, μ-unit outputs τ_μ and state-change vector ν_μ from to DU.

4.3.5 Floating-point arithmetic unit

This implementation utilizes an extended data format for floating-point values. It is based on the IEEE-754 standard for single-precision floating point, but has extended 3 bits for rounding. Thus, floating point values are not approximated within each operations, but maintains the "round" bits. Table 4.2 shows the logic resouces and maximum operating frequencies for floating-point arithmetic which are used in FRM-UNIT. The arithmetics are written in Verilog-HDL, the synthesis and place & route are done by Xilinx ISE-8.1i with the target device for XC2VP70-5 which is equipped on the RecSiP2-board.

The logarithmic unit in FU calculates logarithmic number $\ln(x)$ of the input x by second order interpolation. The random number generator is implemented as 167-bit LFSR (Linear Feedback

Shift Register), which is capable of producing 26-bit random number per clock cycle based on M-sequence.

FP format random number between $[0, 1)$ is generated by following procedure. First, a uniform random number r' between $[1, 2)$ is generated by combining a positive sign bit of 0, 127 as exponential bits and 26 random bits from LFSR. Finally, the random number is obtained by subtracting 1.0 from r'.

4.3.6 Computational time for a reaction cycle on FRM-UNIT

In this subsection, a compuational time for a reaction cycle on FPM-UNIT is explained. As shown in Fig. 4.3, Phase 1, Phase 2, and Phase 4 take $2M$ clock cycles, Phase 3 takes 42 clock cycles to output the result from input, and Phase 5 takes four clock cycles. Phase 1, 2, and 4 are overlapped to advance calculation. If M is less than 46, that is the total time of Phase 3 and phase 5, two simulation threads can be executed in an execution time of single simulation thread. Therefore, the case of $M \leq 46$, the computational time for a reaction cycle is $M + 46$ clock cycles. On the other hand, the case of $M \geq 47$, even though population update of $\mathrm{DS_A}$ is finished, calculation of next reaction cycle can not start till output reactants for $\mathrm{DS_B}$. In simulation for $M \geq 47$ biochemical system, the computational time for a reaction cycle is $2M + 2$ clock cycles including adjudication on the transfer time and state to host PC.

Although time for a reaction cycle is longer than only one simulation thread when M is larger then 47, the size of recent biochemical model for actual usecase may be between $M = 50$ and $M = 100$. As the rate of active and inactive of pipeline for FU are almost same in the simulation of such models, it has the advantage of execution by two threads execution to fill blank of the pipeline.

On the other hand, more than two threads may be executed in parallel when M is small. However, multithreading on FPGA can not be very effective to achieve high throughput compared to multiprocessor which can mark very high performance in very small model. Therefore, we fixed the number of thread executed on a FRM-UNIT.

It may be effective to cut down the calculation time by the FRM-UNIT executes one simulation thread when the main target of the biochemical model is larger than recent models.

4.4 Evaluation

4.4.1 Result of resource utilization

FRM-FPGA module was written in Verilog-HDL, synthesized, and place & routed with Xilinx ISE7.1i. The target device is XC2VP70-5FF1517 equipped on the ReCSiP2-board. Table 4.3 shows the component resources for the FRM-FPGA module and its submodules. This result indicates that the FPGA of ReCSiP2-board is capable for implementing FRM-FPGA with three FRM-UNITs, along with interfaces for connecting host PC and PCI bus. This FRM-FPGA can execute 6 ($= 2 \times 3$) simulations in parallel.

4.4.2 Performance evaluation

(1) Benchmark biochemical systems

Performance of FRM-FPGA was evaluated by three biochemical benchmark programs: LTS (Lotka System), TIS (Totally Independent System) and LCS (Linear Chain System). LTS [16] is a very small scale system typically chosen as benchmarks of stochastic simulators. TIS and LCS [19] are systems whose number of reactions N is variabled. Further details are described in each reference.

Table 4.3: Resource Utilization

Module	Slices	Mult.	Block RAM	Freq. [MHz]
FU	7360(22.24%)	26	8	106.43
DU	992 (2.99%)	-	8	106.32
CU	60 (0.18%)	-	2	143.88
FRM-UNIT (≃ FU+DU+CU)	8166(24.68%)	26	18	106.56
FRM-FPGA (≃ 3 FRM-UNITs)	24496(74.03%)	78	54	106.29

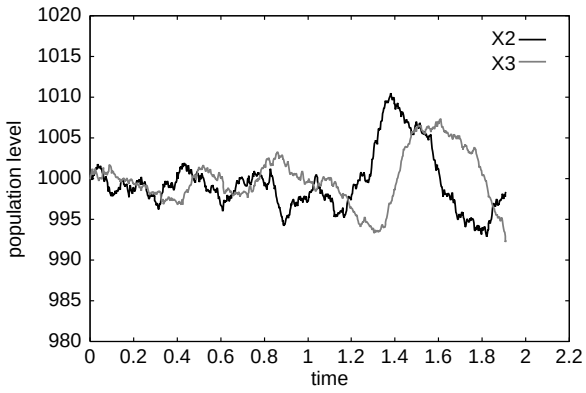


Fig. 4.8: Average populations on FRM-UNIT

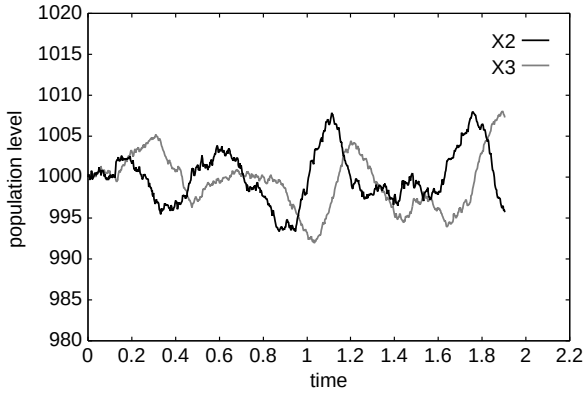


Fig. 4.9: Average populations on FRM-SW

Throughput of FRM-FPGA versus microprocessors with a program code in C++ (indicated as FRM-SW) was evaluated. The program adopts Mersenne-Twister algorithm for generation of random numbers.

It is quite difficult to determine the barometers for "accuracy" in stochastic simulation, but this paper defines the validity of simulation results of FRM-FPGA by smallness of disparity from FRM-SW simulation result. Fig. 4.8 and Fig. 4.9 represent the averaged trajectories of LTS of FRM-FPGA and FRM-SW, respectively. LTS is often observed as a steady state in ODE-based simulation, but values of X_2 and X_3 oscillate with stochastic simulators and each result depicts completely different trajectories (due to the use of different random generation algorithm and random seed). Yet, the trajectories of X_2 and X_3 after 1000-times execution stays within the range of 1000 ± 1.5% in both figures, indicating that FRM-FPGA generates reasonable results with regards to software execution.

(2) Performance evaluation

Throughput of FRM-FPGA was estimated by assuming operational frequency of 106.29 MHz, with regards to the result of placement and routing. FRM-SW was compiled with gcc3.3.5(-O3) at Xeon 2.80 GHz with 4.0 GB RAM, and the throughput was evaluated in Linux2.4.31 environment. Fig. 4.10 and Fig. 4.11 compare the throughput versus N.

Execution time of FRM-UNIT is proportional to system size is N (N + 46 clock cycles when $N \leq 44$, or $2N + 2$ when $N \geq 45$). This is because FRM-UNIT obtains propensity on a fixed circuit capable of up to the second order reaction. Also, deeply pipelined structure of FU may cover up the latency for FP operations, attaining high-throughput versus FRM-SW even with large-scale biochemical systems. As the result, because TIS is the simplest biochemical system definable,

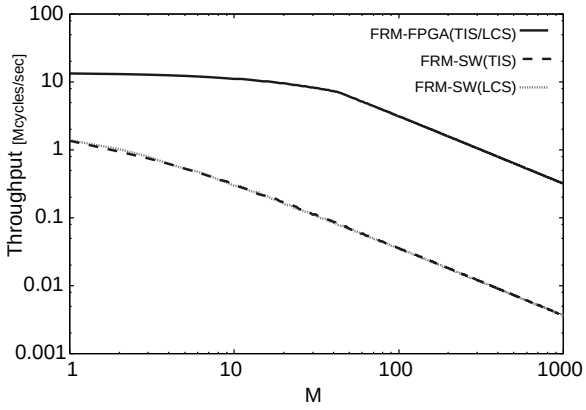


Fig. 4.10: Throughput gain

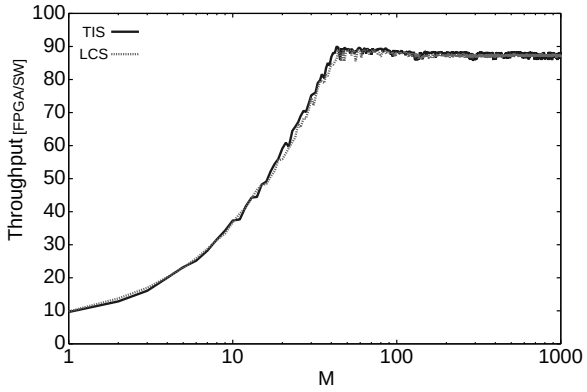


Fig. 4.11: Gain ratio

Fig. 4.11 shows that FRM-FPGA achieves about 80-fold speedup compared with software execution.

4.5 Chapter summary

In this chapter, we proposed a methodology for implementing an SSA algorithm called First Reaction Method (FRM). Processes in the algorithm is simple, and has high degree of loop- and data-level parallelism. Thus, the hardware design was fixed, and data flow was statically scheduled to enhance performance by consecutively injecting data into deep pipelines of floating point units. Arithmetic operations were configured faithfully to the original algorithm by single-precision floating-point data. We validated that the implemented hardware can treat small-scale models well.

Overall, the design achieved more than 80-fold throughput compared to software execution on Xeon 2.80 GHz, with large-scale biochemical systems for up to 1023 reactions.

Chapter 5

Implementation of Next Reaction Method on an FPGA

This chapter explains an implementation of NRM simulator module.

5.1 Design of NRM on an FPGA

5.1.1 Analysis of NRM

As a preliminary of implementation of NRM simulator module, performances of FRM and NRM in C++ language by microprocessors whose environment is shown in Table 5.1 (afterward, they are called FRM-SW and NRM-SW, respectively) are compared with the performance of FRM-FPGA discussed in Section 4.2 for various size of models. In this evaluation, we used virtual models in which n set of the Lotka system($M = 4, N = 4$) is defined. Fig. 5.1 shows results within the range of $M \leq 1024$ as realistic size of biochemical models. n sets of the Lotka model are described as nLotka in Fig. 5.1.

As shown in Fig. 5.1, FRM-FPGA outperforms throughput about 6.64 times in 1Lotka and about 40 times in more then 16Lotka, compared to the microprocessor. However, FRM becomes less efficient than NRM because FRM does not use the dependency graph to reduce the number of calculation in a reaction cycle. Therefore, FRM is the algorithm under no circumstance to simulate for models which defines large number of reactions such as more than 32Lotka. As results between FRM and NRM are proved statistically same, FRM-FPGA is hard to say that it is accelerator for large biochemical simulations.

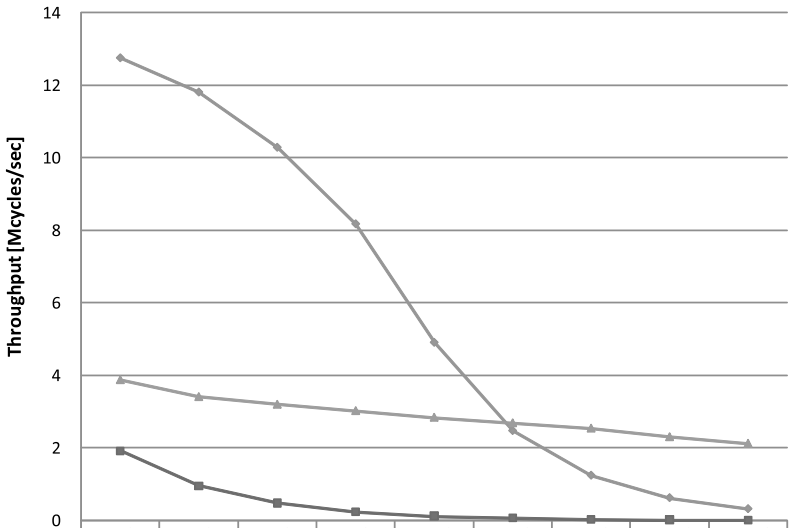
Recently, although some SSAs which are said that they are faster than NRM are proposed, NRM on an FPGA (which is written as NRM-FPGA afterward) is implemented to simulate large biochemical models because NRM indicates the most efficient time complexity.

Number of function calls were also obtained and are shown in Fig. 5.2. n sets of Heat-Shock Response (nHSR) Model (1HSR : $M = 61,\ N = 28$) was used, and set number of reactions within the range of $M \leq 1000$.

The profile was taken when 2×10^6 reaction cycles were executed. As we have already shown in

Table 5.1: Execution environment for C++ program code

CPU	Intel Core 2 Quad Q6600 (2.40 GHz)
Memory	3.5 GB (4.0 GB on board)
OS	Linux 2.6.22-14 (x86-32bit)
Compiler	gcc 4.1.3 (-O3) 1 core



	1Lotka (M=4)	2Lotka (M=8)	4Lotka (M=16)	8Lotka (M=32)	16Lotka (M=64)	32Lotka (M=128)	64Lotka (M=256)	128Lotka (M=512)	256Lotka (M=1024)
FRM-FPGA	12.755	11.810	10.286	8.176	4.906	2.472	1.241	0.622	0.319
FRM-SW	1.918	0.959	0.479	0.242	0.121	0.061	0.031	0.015	0.008
NRM-SW	3.876	3.409	3.198	3.021	2.839	2.685	2.538	2.300	2.120

Fig. 5.1: Calculation time and its breakout for the Lotka model in NRM

Fig. 5.2, calculation time displays mildly increases and is linear to the number of reactions M. The increase is due to the update of the binary tree in IPQ.

Computational time of NRM is also influenced by the length of list of DG. The number of executing innermost loop is linear to the number of reactions listed in DG. Also, data for frequently-occurred reactions in HSR model concentrates at the root node of the tree. Note that reactions are likely to occur when there is a big population of molecules involved in it or when the value of reaction constant is large. Propensity of such reactions are also large, and putative time is likely to be the nearest value from the present time. Thus, they are apt to stay at the root of the tree. Number of functions calls for US decreases as number of reaction increases. Thus, we can find that length of DG's list for HSR model is relatively short.

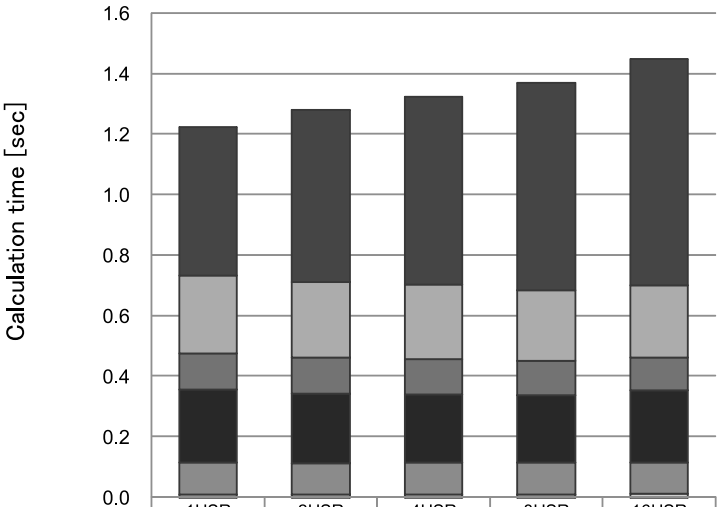
5.1.2 Analysis of NRM execution unit using an FPGA

As shown in Fig. 5.3, large proportion of the execution time is computation of reaction cycles. Memory regions allocated during runtime are as small as approximately $M\times$24 KBytes ($M\times$20 KBytes for storing intermediate populations and $M \times 4$ KBytes for parameters). Based on these characteristics, we will discuss the methodology for acceleration.

First, the whole computation is done on an FPGA, and no interaction will be performed with host PCs except for sending or receiving initial values and output results. Because computation of each reaction cycles simple enough to be done with an FPGA, communication overhead between a host PC would obviously be dominant among the execution time.

Second, throughput can be improved with multithreading by exploiting thread-level parallelism of NRM. Hereafter, we will refer a hardware module as a "thread module", which computes one thread of NRM according to the flowchart shown in Fig. 2.9 and stores its intermediate status. Capacity of BlockRAMs are large enough to store the whole simulation data, we will form a distributed memory system with different data types, and allow parallel access to them.

Throughput of mathematical operation is often improved by throwing data consecutively into deeply-pipelined arithmetic cores. In general, floating-point units consume large logic capacity, we



	1HSR (M=61)	2HSR (M=122)	4HSR (M=244)	8HSR (M=488)	16HSR (M=976)
IPQ update	0.490	0.568	0.621	0.686	0.749
Tau-modification(U5)	0.258	0.250	0.246	0.232	0.237
Tau-calculation(U4)	0.119	0.119	0.116	0.113	0.109
Propensity-calculation(U3)	0.243	0.231	0.226	0.222	0.238
State update(U1)	0.103	0.104	0.105	0.106	0.104
etc.	0.010	0.008	0.009	0.009	0.010
total	1.223	1.279	1.322	1.368	1.448

Fig. 5.2: Profiles for HSR model in NRM-SW

can streamline by allowing several thread modules share one arithmetic core. Hereafter, we will refer to these arithmetic cores and their I/Os as “shared modules”.

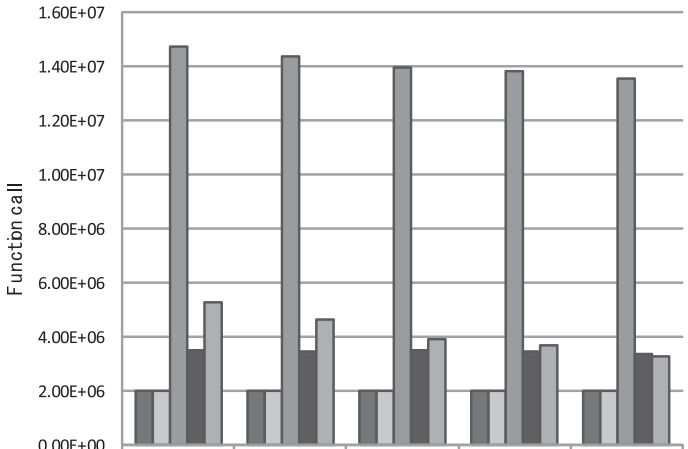
Types of arithmetic operation, its occurrence and timing vary among selected reactions. Thus, pipeline efficiency is degraded when timing of data transfers between modules are statically scheduled. Consequently, we can eliminate pipeline vacancy by interconnecting shared modules and thread module with some network mechanism, and appropriately sequencing requirements of each thread modules. The structure of interconnects are modifiable, so number of thread modules can be adjusted according to the capacity of FPGA platform.

Fig. 5.4 shows the interconnection of each modules. Each thread module has data necessary for one simulation thread of NRM. It appropriately transfers data to the shared modules as simulation proceeds. There are five different types of shared modules: modules with unmodified parameter tables (U1 and U2), and computation modules (U3, U4, and U5). They receive input as shown in Fig. 5.4 and return their results.

Main concern of this system is the principle of interconnection between modules. The simplest method is to connect modules with multiplexers (MUX), and we firstly tested the efficiency of this structure [49]. Its major problem was that multiplexers could not latch signals internally. Thus, input, control and output process had to be done in one clock cycle, which seriously prolonged the latency that it negated the efficiency obtained by multithreading.

Next, Network-on-Chip (NoC) method was adopted [50]. Latency was moderate with hierarchical network between shared and thread modules. However, output port was configured with Block-RAMs, and crossbars were used as switches. Moreover, due to the nature of NoCs, there were paths between modules without any communication. Overall, this type achieved poor performance over logic utilization.

Third structure we adopted is Distributor-Concentrator type which is described in Section 5.2.5. With original network structure, we achieved high performance over logic utilization.



	1HSR (M=61)	2HSR (M=122)	4HSR (M=244)	8HSR (M=488)	16HSR (M=976)
U1	2000000	2000000	2000000	2000000	2000000
U2	2000000	2000000	2000000	2000000	2000000
U3	14714327	14379002	13963897	13798527	13540956
U4	3487257	3442904	3520728	3452620	3369411
U5	5288292	4617244	3890173	3662927	3280242

Fig. 5.3: Number of function call in NRM

5.2 Implementation of NRM execution system

This section describes the detail of structures and implementation of thread modules, shared modules and interconnects in the NRM execution system. Each module was written in Verilog-HDL. Floating point arithmetic units and memories were designed by utilizing single-precision floating-point operation units and BlockRAMs produced by Xilinx's CORE generator.

5.2.1 Data transfer protocol between modules

Each module communicates by sending and receiving data packets. Each flit is 32bit wide, and one packet consists of a header flit and pay-load flits which is more than one flit long. Header flit stores the details of arithmetic processes and routing sequence. Pay-load flits contain simulation data which will be the input of shared modules. Source-routing mechanism is adopted. Note that superscription and interruption by trailing packets are avoided by several inter-module control signals and transfer control using FIFOs.

5.2.2 Structure of thread modules

Thread modules possess intermediate data of one simulation thread, and throw data to shared modules according to the sequence shown in Fig. 2.9.

Fig. 5.5 is a block diagram of thread modules. Intermediate data stored in the module are status of the biochemical model (time and population of molecules), IPQ and propensity. The memory was configured with Dual-port BlockRAM that stores up to 1024 words. This means that the system is capable of simulating a models whose parameters are $M = 1023, N = 1024$. Case of $M = 1024$ is incapable because IPQ is a binary tree and we cannot use a field whose address is zero. IPQ has a 10-bits×1024 index table which stores pointer of each reaction within a binary tree and a 42-bits×1024 binary tree that stores data in a pair of reaction ID number and putative time. IPQ has a controller in

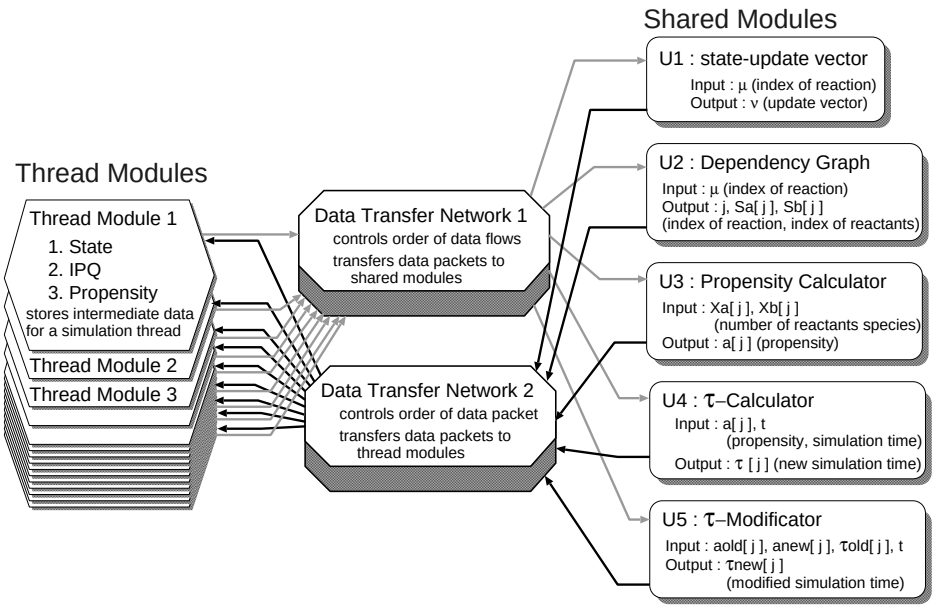


Fig. 5.4: Module connection diagram in NRM execution system

charge of reading data and updating values. Magnitude correlation of the binary tree is maintained by comparison and exchange nodes completed within five clock cycles.

Thread module has a receiver FIFO to store packets from other modules and a sending controller. It also has an external I/O port exclusive for communication with each BlockRAMs, and is used for writing initial values and reading output values.

5.2.3 Principles of thread modules

Packet controller in a thread module generates packets for control of arithmetic operations and communication, and sends data to shared modules. Received data are stored in FIFOs. Packet controller is an 8-state machine, which reads or writes value into FIFOs and memories and processes according to appropriate sequence.

Fig. 5.6 shows the relationships between state transition of packet controllers and sender or receiver packets in one reaction cycle. The ovals indicate the status of packet controller, and the solar-marks are status of sender packets. Input of each edge indicates the types of receiver packets or certain conditions. The calculation scheme is as follows:

1. Calculation of each reaction begins from “START”, and proceeds according to the flow chart shown in Fig. 5.6

2. U1 and U2 are processed in parallel, so reaction ID number is read from the root node of IPQ’s binary tree, and packets are generated and sent consecutively to U1 and U2

3. Packets sent to each shared modules have pay-loads whose values are shown in “Input:” in Fig. 2.9. Afterwards, the status becomes “IDLE” and, waits until data is pushed into a receiver FIFO.

4. Status becomes “FETCH” when data is pushed into the FIFO, and popped a packet.

5. Status transits to “Update State” if the source of the packet is U1, and model’s status (or populations) is updated according to the data-flits in pay-load.

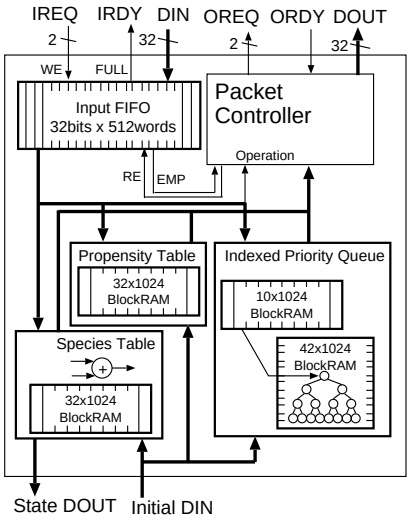


Fig. 5.5: Structure of the threaded module

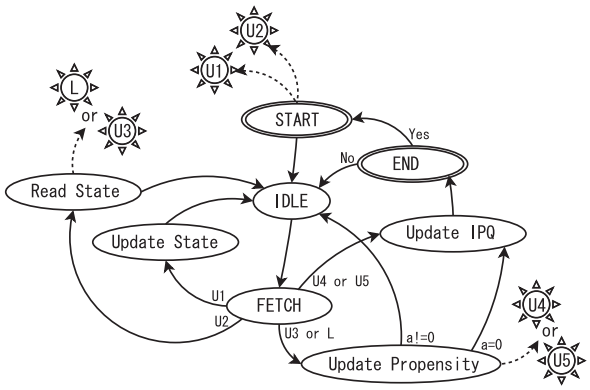


Fig. 5.6: State transition in the packet controller and send/receive packet in a reaction cycle

6. Likewise, if the packet is from “U2”, status becomes “Read State”. And sends packet to U3 if process in U1 is complete. If not, packet is sent to itself (L: loop packet), and waits until data for U3 is ready.

7. Data of U1 packet has a number of reactions(including evolution reactions) whose putative time must be updated, and identical number of packet is sent back as U2 packet.

8. If U3 or loop packet was read out, the status becomes “Update Propensity” and updates propensity of corresponding reaction. Note that process depends on values of $a_{j,\mathrm{old}}$, $a_{j,\mathrm{new}}$, $\tau_{j,\mathrm{old}}$, and selects whether the system should send packet to U4, U5, or update IPQ’s binary tree.

9. One simulation cycle ends when the system reaches certain simulation time or has gone through “Update IPQ”. At this point, reaction number at the root of IPQ’s binary tree is the reaction that is going to occur at the next cycle, and its putative time will be the next system time.

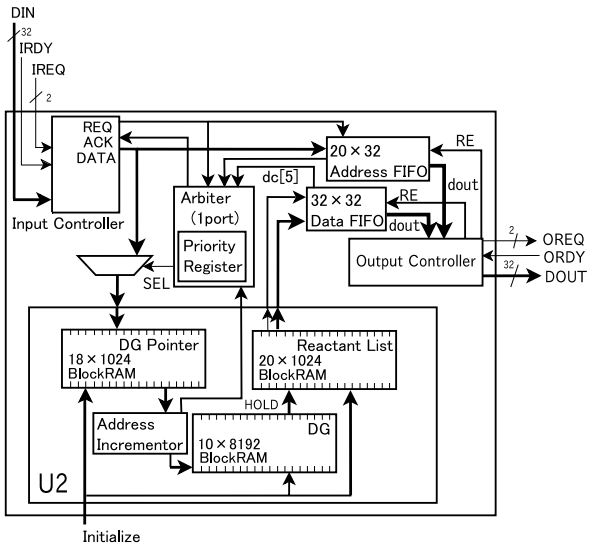


Fig. 5.7: Structure of shared module U2 (Dependency Graph) with a set of I/O port

5.2.4 Principles of shared modules

Shared module consists of I/O ports, arithmetic cores and an arbiter. As described in Section 5.1.2, there are five types of arithmetic cores: two "data-memory" types (U1 and U2) and three "computation" types (U3, U4, and U5). There are constantly only one set of I/O ports in a thread module, but its number of shared modules are variable according to the form of interconnect and the number of thread modules. Fig. 5.7 and Fig. 5.8 show examples of U2 with one set of I/O port and U4 with two sets. An Input port of a shared module stores receiver packet into its internal register, and requests a property to use an arithmetic core to the arbiter. An arbiter is in charge of selecting requests from input ports, and sends data of pay-loads from certain input to an arithmetic core. Arithmetic cores process input data according to Fig. 5.4. Output port stores header flit from the input port and output results from an arithmetic core into its internal FIFO. The FIFO generates a packet and sends back to the owner thread module.

"Data-memory" type of arithmetic cores receive reaction ID and generate a variable-length list of update status vector (U1) or ID number of reactant molecules (U2). As it is a two-dimensional array, arithmetic core is a two-stage hierarchical structure that connects data and pointer memories, as shown in Fig. 5.7. First, Reaction ID is used an address of pointer memory to read the number of data and header address of data memory. Based on this, a list is read from the data memory. Meanwhile, the arbiter does not allocate property to use the core to other modules. At the output port, one sending packet is generated only with one data in the list in order to reduce the waiting time due to collision in the interconnect.

On the other hand, "computation" type of arithmetic cores is pipelined, and can process consecutive requests per clock cycle. U4 shown in Fig. 5.8 performs a single-precision floating point computation based on Eq. 2.10. Random numbers in the range of $(0, 1)$ are generated with an M-sequence algorithm using a Linear Feedback Shift Register (LFSR). First, random numbers between $(1, 2)$ is generated from uniform random numbers, and then value 1.0 is subtracted. Logarithm numbers with base of natural logarithm "e" is calculated with two-dimensional interpolation. Random numbers are generated independently from data of biochemical models, so they can be stored in FIFOs and used when they are necessary. Random number generator modules are interchangeable as occasion arises. U3 and U5 were also implemented based on this guideline.

Size of input packet to U4 is three flits. Thus, pipeline utility is 33% at most when there is only

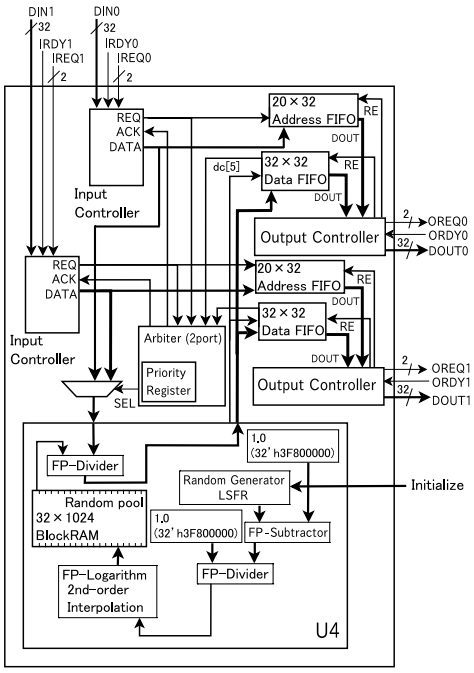


Fig. 5.8: Structure of shared module U4 (calculates τ) with two sets of I/O port

one set of I/O port. This is improved by simply adding more I/O ports so that waiting time at the input can be reduced. On the other hand, U1 and U2 are not pipeline-based, so waiting time cannot be improved by increasing the number of I/O ports. As shown in Fig. 2.9, U1 and U2 are only used once per reaction cycle. Consequently, waiting time at these inputs does not have large influence on this NRM system, but can be improved by configuring multiple sets of U1 and U2.

5.2.5 Principles of interconnections

Interconnection of NRM system connects thread modules and shared modules, and also controls packet transfer. There are two types of controller modules in the interconnect: a “Concentrator” which aggregates multiple inputs and controls their sequence, and a “Distributor” which sends data to appropriate output according to the information given by input header flit. Data flow of NRM system is rather simple; packets are sent from each thread module, and computation results are given from shared modules in return. Thus, these two modules play adequate roles in this interconnection system.

Fig. 5.9 shows block diagrams of a 4-port Concentrator and a Distributor. Number of ports in both modules are variable. Each input port owns 34-bit$\times$2 FIFOs, and suppresses wiring delay likely to be caused by multiple-stage interconnection.

As mentioned before, Concentrator is in charge of controlling sending requests from multiple input ports, and organizing them as a single output. Arbiters refers to controlling information from each input ports, and establishes a data transfer signal line to the output port. Distributor selects an output port based on routing information in the header flit, and send the packet.

5.2.6 Organization of NRM execution system

NRM execution system is configured by multiple thread modules and shared modules communicating via an interconnection. Fig. 5.10 shows an example with four thread modules.

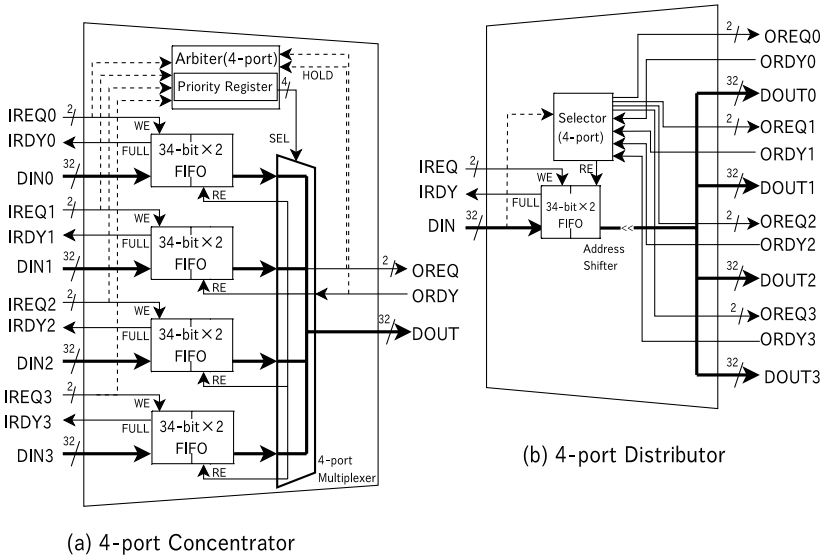


Fig. 5.9: Examples of 4-port interconnection modules

The system first writes parameters of a target biochemical model to shared modules, and initial values of each thread are stored into independent thread modules. Simulation proceeds by repeating the sequences described in Section 5.2.3. Intermediate status is dumped per time interval configured before the simulation.

Hereafter, an NRM execution system as shown in Fig. 5.10 with p thread modules and one set of output port is called a "Tp execution system". A pair of a Concentrator and a Distributor is used for connecting the shared modules. Analysis of packet congestion in Tp execution system is fairly easy, and the system can be regularly extended. This section evaluates area and performance of the system by modifying the value of p.

Tp execution system is a reasonable structure when the utility of each shared module is equally likely, but I/O ports of a shared module easily becomes a bottleneck when its utility is high. Fig. 5.3 shows that frequency in use of U3 is obviously high. Thus, we selected a T16 execution system whose degradation of operational frequency is in an allowable range and U3 distinctively being the communication bottleneck, and designed a T16C execution system whose portion of interconnection was modified. T16C system uses a U3C module (U3 module with four I/O ports) and each ports are connected with four threaded modules via 4-port Concentrator and Distributors. This design improves pipeline efficiency of U3's arithmetic core.

5.3 Evaluation

This section shows the evaluation of NRM execution system. The target system is an FPGA evaluation board designed by Tokyo Electron Device (TB-5V-LX110T-PCIEXP [51], On board FPGA: XC5VLX110T-FF1136). Synthesis, placement and routing was done by Xilinx ISE8.2i, and performance was evaluated by RTL simulation.

5.3.1 Module area

Table 5.2 shows areas and operating frequencies of implemented modules, Based on this, we will estimate the number of thread modules configurable on FPGAs. Thread module consumes approximately 2.1%(= 1283/61920 × 100) of LUTs, and 4.1% of BlockRAMs. Hence, the maximum num-

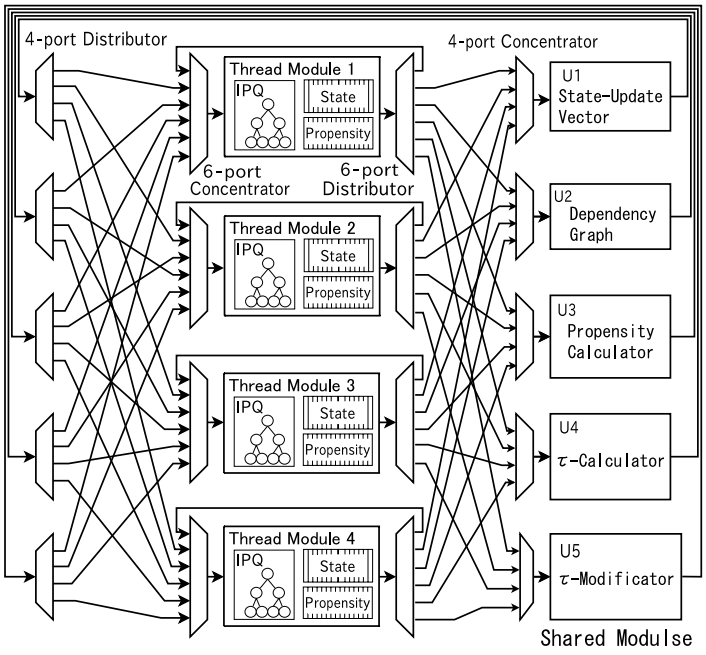


Fig. 5.10: Structure of NRM execution system with 4 threaded modules

Table 5.2: Area and operating frequency of each module

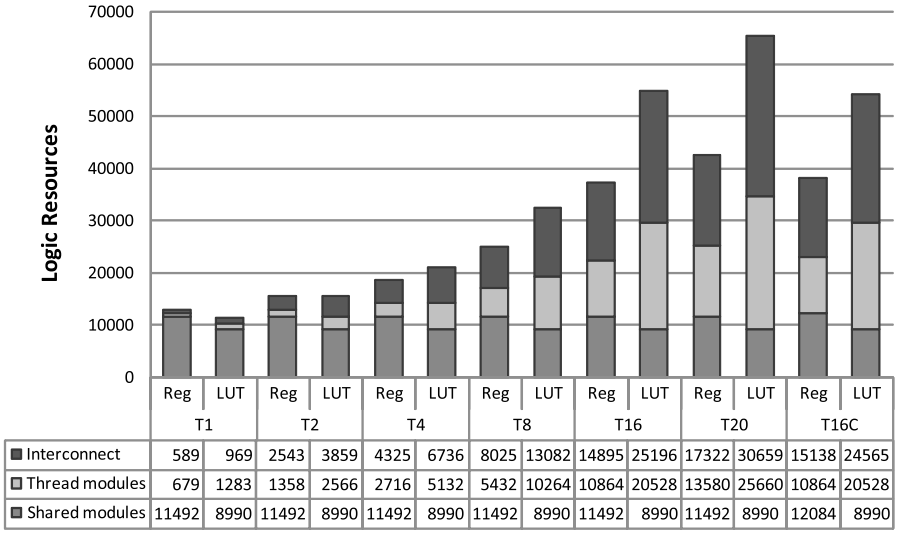
	Thread	U1	U2	U3	U4	U5	U3C
Registers	679	161	215	1028	7231	2857	1620
LUTs	1283	348	384	993	5154	2111	1774
BlockRAM/FIFO	6	5	7	4	6	2	9
DSP48Es	0	0	0	5	14	5	5
Max. Delay[ns]	6.40	5.44	5.42	5.73	5.84	4.29	5.73
Op. Freq.[MHz]	156.30	183.82	184.54	174.43	171.38	233.10	174.43

⋆ XC5VLX110T-FF1136 : Slice 69120 : LUTs 69120 : BlockRAM/FIFO 148 : DSP48E 64

ber of thread modules is limited by the number of BlockRAMs on a target FPGA. U1 and U2 uses many BlockRAMs for storing model parameters, but consumes less logic resource compared to other modules because their internal control logics are relatively simple. On the other hand, U3, U4 and U4 consumes considerable amount of logic resource because they mainly consist of floating-point arithmetic cores.

5.3.2 Area of NRM execution system

Area and operational frequency of entire NRM execution system are shown in and Table 5.3 and Fig. 5.11, respectively. As shown in Fig. 5.11, target FPGA is capable of holding T20 system. Table 5.3 indicates that systems under T8 are capable to keep their operational frequency high, and their critical paths are within thread modules. On the other hand, critical paths of T16, T20 and T16C execution system lies in Concentrators and Distributors because of their increasing number of ports. These systems consume many LUTs for their interconnections. For example, there is approximately 2.24% increase of slice registers between T16 and T16C, while number of LUT and operational frequency are equally likely. Computational type of shared modules can increase its pipeline efficiency without much increase of area by owning several sets of I/O ports like U3C,



	Reg	LUT	Reg	LUT	Reg	LUT	Reg	LUT	Reg	LUT	Reg	LUT	Reg	LUT
	T1		T2		T4		T8		T16		T20		T16C	
Interconnect	589	969	2543	3859	4325	6736	8025	13082	14895	25196	17322	30659	15138	24565
Thread modules	679	1283	1358	2566	2716	5132	5432	10264	10864	20528	13580	25660	10864	20528
Shared modules	11492	8990	11492	8990	11492	8990	11492	8990	11492	8990	11492	8990	12084	8990

Fig. 5.11: Resource utilization of NRM execution system

because thread modules connected to each I/O ports can be equally distributed.

5.3.3 Performance evaluation

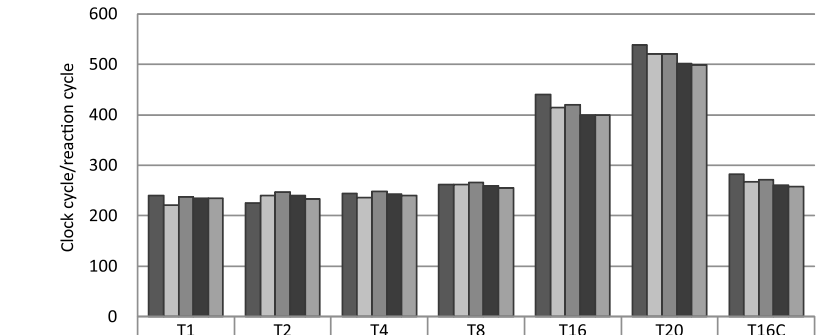
We will evaluate each NRM execution system with regards to number of thread modules and number of reactions. For this aim, an RTL simulation with n sets of HSR system (introduced in Section 5.1.1) were run for 50000 reaction cycles. Hereafter, we will refer to this model as nHSR. Fig. 5.12 shows average clock cycles for simulating one reaction cycle, and Fig. 5.13 is average waiting time for each type of data packets to send one flit to an adjacent module. Fig. 5.14 is pipeline efficiency of arithmetic cores in each shared module.

First, we will discuss the influence of thread modules and performance. As Fig. 5.12 indicates, number of clock cycles increase when number of thread module increases. T16 and T20 are distinctive cases; they are due to waiting time of U3 ad shown in Fig. 5.12. Pipeline efficiency of arithmetic cores is proportional to the number of thread modules as Fig. 5.14 indicates, but it saturates at 33.3% for U3 in case of T16 and T20. Operational rate of U3C with four sets of I/O ports is 100%, so T16C design was able to resolve the bottleneck at U3.

Next, we will discuss the relationships between number of reactions and clock cycles. As shown in Fig. 5.3, number of nodes in IPQ increases as number of HSR model increases. Meanwhile, it reduces the number of function calls for U3 and U5.

Table 5.3: Operation frequency of NRM execution system

Execution system	Operational frequency[MHz]
T1	156.30
T2	156.30
T4	156.30
T8	155.62
T16	141.30
T20	120.69
T16C	138.60



	T1	T2	T4	T8	T16	T20	T16C
1HSR(M=61)	239.68	224.51	244.21	262.21	441.06	538.07	281.74
2HSR(M=122)	220.63	239.54	235.98	261.81	415.21	520.46	267.35
4HSR(M=244)	237.28	247.34	248.68	265.76	419.34	520.64	271.45
8HSR(M=488)	233.96	239.43	242.71	258.42	399.76	501.43	261.11
16HSR(M=976)	234.55	233.67	239.67	255.23	399.86	499.64	257.24

Fig. 5.12: Average clock cycles to calculate a reaction cycle

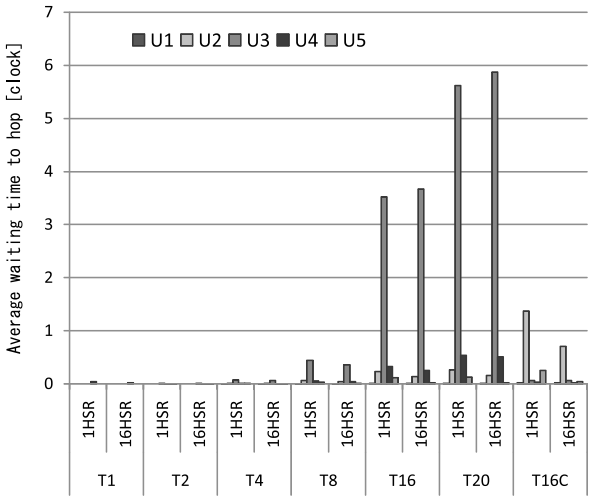


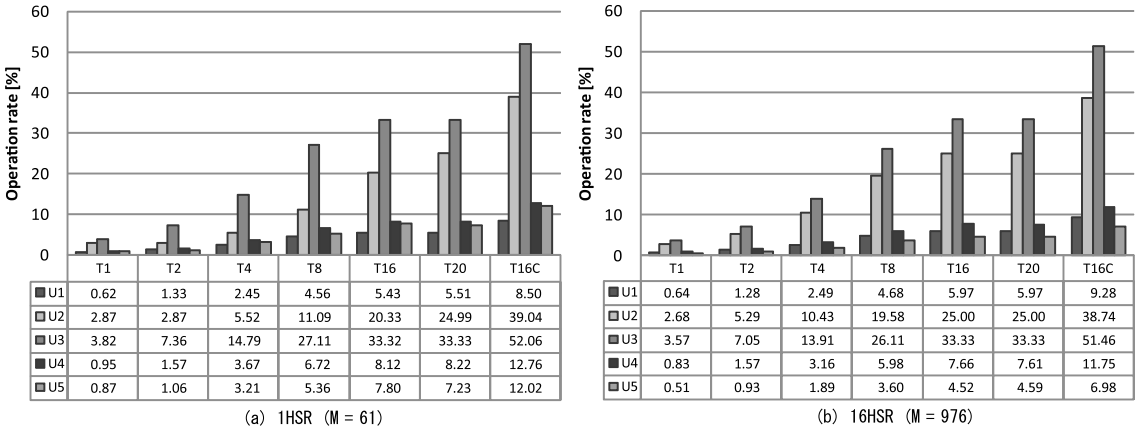
Fig. 5.13: Average waiting time to transfer for each packet

In addition to interruption and route distribution are not supported, each reaction cycle starts when all processes of previous reaction cycle is completed. Thus, whole computational performance is limited when only a portion of interconnection has a bottleneck. Consequently, it is our next work to investigate optimal interconnection by inspecting the tradeoff between pipeline efficiency and transfer waiting time.

5.3.4 Throughput

Fig. 5.15 shows throughput of each execution system and result of the NRM algorithm runs on a general-purpose processor whose execution environment is shown in Table 5.1. Throughput of DM by Stochkit is also indicated for reference. Throughput of FPGA system was obtained by dividing operational frequency with clock cycles required for simulating one reaction cycle (shown in Fig. 5.12) and multiplying the number of thread modules. The system was operated with 150 MHz, 135 MHz and 120 MHz for execution system smaller than T8, 16, and T16C, respectively.

There is a linear increase of throughput between T1 and T8. However, Fig. 5.15 indicates that throughput of T20 system is inferior to that of T16 due to the drop of operational frequency. On the



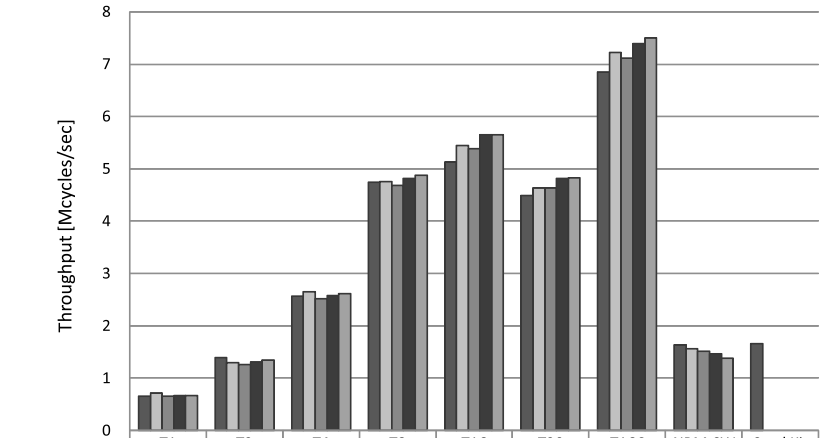
	T1	T2	T4	T8	T16	T20	T16C
U1	0.62	1.33	2.45	4.56	5.43	5.51	8.50
U2	2.87	2.87	5.52	11.09	20.33	24.99	39.04
U3	3.82	7.36	14.79	27.11	33.32	33.33	52.06
U4	0.95	1.57	3.67	6.72	8.12	8.22	12.76
U5	0.87	1.06	3.21	5.36	7.80	7.23	12.02

(a) 1HSR (M = 61)

	T1	T2	T4	T8	T16	T20	T16C
U1	0.64	1.28	2.49	4.68	5.97	5.97	9.28
U2	2.68	5.29	10.43	19.58	25.00	25.00	38.74
U3	3.57	7.05	13.91	26.11	33.33	33.33	51.46
U4	0.83	1.57	3.16	5.98	7.66	7.61	11.75
U5	0.51	0.93	1.89	3.60	4.52	4.59	6.98

(b) 16HSR (M = 976)

Fig. 5.14: Operation rate for each functional core (unit: %)



	T1	T2	T4	T8	T16	T20	T16C	NRM-SW	StockKit
1HSR(M=61)	0.652	1.392	2.560	4.748	5.126	4.486	6.854	1.636	1.661
2HSR(M=122)	0.708	1.290	2.649	4.755	5.445	4.638	7.223	1.564	
4HSR(M=244)	0.659	1.264	2.514	4.684	5.391	4.636	7.113	1.513	
8HSR(M=488)	0.668	1.306	2.576	4.817	5.655	4.814	7.395	1.462	
16HSR(M=976)	0.666	1.338	2.609	4.878	5.654	4.831	7.506	1.382	

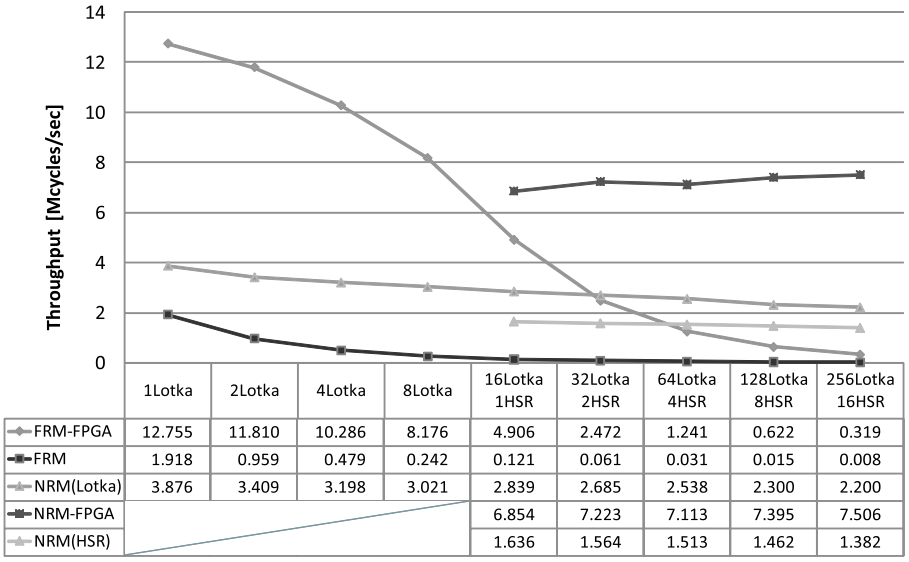
Fig. 5.15: Comparison for throughput (unit: Mcycles/sec)

other hand, T16C resolves the bottleneck of U3, as mentioned before.

In case of a microprocessor, Fig. 5.2 shows that access time for IPQ increases with the number of reactions. On the other hand, NRM tolerates the increase in number of reactions. For example, throughput is improved for HSR model with larger number of reactions. This may be because frequently occurred reactions in multiple HSR are concentrated around the root node, the rebuilding time is shorter than 1HSR.

FPGAs used in this work can configure two T8 execution system. They are independent and can be run in parallel, so this doubles the throughput and is better than T16C. This result suggests our next work, which is to improve the efficiency of single execution system, as well as availability of multiple execution systems on a single FPGA.

Best performance of single execution unit was provided by T16C, and approximately 4.2 to 5.4 time higher throughput was obtained compared to NRM run on the microprocessor. Performance gap between execution on microprocessors increase with number of reactions. These evaluation results proved that FPGA implementation presented in this work maintains a scalability for a biochemical model with up to 1023 reactions.



	1Lotka	2Lotka	4Lotka	8Lotka	16Lotka 1HSR	32Lotka 2HSR	64Lotka 4HSR	128Lotka 8HSR	256Lotka 16HSR
FRM-FPGA	12.755	11.810	10.286	8.176	4.906	2.472	1.241	0.622	0.319
FRM	1.918	0.959	0.479	0.242	0.121	0.061	0.031	0.015	0.008
NRM(Lotka)	3.876	3.409	3.198	3.021	2.839	2.685	2.538	2.300	2.200
NRM-FPGA					6.854	7.223	7.113	7.395	7.506
NRM(HSR)					1.636	1.564	1.513	1.462	1.382

Fig. 5.16: Throughput of descripted hardware (unit: Mcycles/sec)

5.4 Review

Fig. 5.16 shows performances of FRM and NRM run on both implemented hardware and software. The throughput was measured with Lotka system and HSR model with various model size. Fig. 5.16 contains performances of FRM-FPGA, FRM-SW and NRM-SW for Lotka systems, and performances of NRM-FPGA and NRM-SW for HSR models. Software programs are executed on environments shown in Table 5.1.

With software, as shown in Fig. 5.16, NRM-SW exhibits different performance curves for Lotka systems and HSR models depending on its forms. Although the number of reaction of 16Lotka ($M = 64$) and 1HSR ($M = 61$) is almost same, the throughput of 1Lotka is 30% higher than 1HSR. The average weighted degree, which was proposed by Cao [19] is about $D_{\text{Lotka}} = 3.0$ for Lotka systems and $D_{\text{HSR}} = 8.79$ for HSR model. The barometer D is calculated by Eq. 5.1:

$$D = \frac{\Sigma d_i k_i}{\Sigma k_i} \tag{5.1}$$

where d_i is the number of reactions affected by the ith reaction, and k_i is the number of times that the ith reaction is fired during a simulation. This difference is obviously caused by the number of calculation for modifying τ_j.

As explained in Section 5.1.1, the throughput of FRM-FPGA becomes inferior to NRM-SW when the number of reaction is larger than $M = 128$. However, NRM-FPGA always outperforms NRM-SW for approximately 4.2 to 5.4 times with HSR models. Additionally, the difference between NRM-FPGA and NRM-SW also expands according to the number of sets of HSR. On the other hand, FRM-FPGA may exceed NRM-FPGA when the size of the model is less than $M = 128$

Therefore, we perceived the condition to select the two algorithms when we apply FPGA for the stochastic biochemical simulation. That is, FRM-FPGA should be applied when the model is small, and performs well when number of reaction defined is less than $M = 128$. NRM-FPGA should be applied to larger models.

5.5 Chapter summary

This chapter presented an implementation of SSA execution system applying an algorithm called Next Reaction Method (NRM). The algorithm adopts two distinctive data structures: a binary tree in an Indexed Priority Que (IPQ) and a Dependency Graph (DG). They are used for improving computational efficiency over FRM for large-scale models, but processing time dynamically varies because of the nature of their data structures.

Main concept of the design for executing NRM was to allow multi-thread execution. However, since it is highly difficult to achieve high throughput by naively exploiting data-level parallelism, the design adopted a data-driven methodology. Every module in the circuit was categorized into two groups. The former is called a thread module, which owns its data structure (IPQ, DG and initial values) for one simulation thread and initiates a computation. The latter is called a shared module, which are parameter tables or arithmetic pipelines. The two modules are linked with an interconnection network. By modifying the network between two groups, hardware design can be flexibly tuned to perform well on the target FPGA device.

Finally, performance using three categories of network were evaluated: simple multiplexer, NoC, and modified multiplexers called Concentrator and Distributor that play similar roles to routers. We found that the third approach achieves best performance. It was evaluated with different number of threads, and through the result analysis we studied a methodology to reduce waiting time.

Overall, the design achieved approximately 4.2 to 5.4 times higher throughput compared to execution on Core 2 Quad Q6600 2.40GHz.

Chapter 6

Conclusion

6.1 Summary

This thesis presented implementations and evaluations of two SSA algorithms on an FPGA. We discussed the area, throughput and availability compared to their software execution.

Contribution of this work is to clarify the relationship between algorithm, hardware architecture, and performance according to data size based on the evaluation results. On designing hardware, we covered following three points:

1. It runs exact same algorithm with original SSAs using floating-point arithmetics,
2. It has a capability to simulate large scale biochemical model,
3. It exploits effective utilization for logic resource in FPGA.

To attend these requirements, we proposed methodologies for implementing following algorithms, and their performances were evaluated.

At first, we proposed a methodology for implementing an SSA algorithm called First Reaction Method (FRM). Processes in the algorithm is simple, and has high degree of loop- and data-level parallelism. Thus, the hardware design was fixed, and data flow was statically scheduled to enhance performance by consecutively injecting data into deep pipelines of floating point units. Arithmetic operations were configured faithfully to the original algorithm by single-precision floating-point data. We validated that the implemented hardware can treat small-scale models well. Overall, the design achieved more than 80-fold throughput compared to software execution on Xeon 2.80 GHz, with large-scale biochemical systems for up to 1023 reactions.

Secondly, this work presented an implementation of SSA execution system applying an algorithm called Next Reaction Method (NRM). The algorithm adopts two distinctive data structures: a binary tree in an Indexed Priority Que (IPQ) and a Dependency Graph (DG). They are used for improving computational efficiency over FRM for large-scale models, but processing time dynamically varies because of the nature of their data structures.

Main concept of the design for executing NRM was to allow multi-thread execution. However, since it is highly difficult to achieve high throughput by naively exploiting data-level parallelism, the design adopted a data-driven methodology. Every module in the circuit was categorized into two groups. The former is called a thread module, which owns its data structure (IPQ, DG and initial values) for one simulation thread and initiates a computation. The latter is called a shared module, which are parameter tables or arithmetic pipelines. The two modules are linked with an interconnection network. By modifying the network between two groups, hardware design can be flexibly tuned to perform well on the target FPGA device.

Finally, performance using three categories of network were evaluated: simple multiplexer, NoC, and modified multiplexers called Concentrator and Distributor that play similar roles to routers. We found that the third approach achieves best performance. It was evaluated with different number of threads, and through the result analysis we studied a methodology to reduce waiting time.

Overall, the design achieved approximately 4.2 to 5.4 times higher throughput compared to execution on Core 2 Quad Q6600 2.40GHz.

6.2 Outlook for the future

Computing systems using FPGAs will expand the future to achieve high-performance for many applications with lower operating costs. The price of an FPGA chip will degrade by volume efficiency. This is because FPGAs are now embedded in large-scale computing systems in some research institute, such as BEE2 [52] and commercial products like high-vision recorders and video capture cards. This means that the advantage of paying initial costs for introducing FPGA devices is becoming much higher than producing ASIC. Moreover, the operating cost is much lower than personal computers and other hardware like Cell/BE and GPU when applications are limited to rather restricted range.

This study showed that a middle-range FPGA can achieve severalfold higher throughput compared to a recent microprocessor. Although it is true that recent FPGAs can outperform performance of personal computers by 10 times in maximum, performance improvement by parallel processing on a chip can be expand linearly according to the capacity of the FPGA. Moreover, recent FPGAs are being developed to operate with operating frequency higher than 500 MHz. Recent microprocessors will not be able to improve performance simply by their number of core, because each core shares a cache with other cores.

Therefore, FPGA limps toward an alternative of a computational system using multiple microprocessors. By then, we may have a new approach for implementation; from traditional methodology to write programs which is adjusted to hardware, to selecting and generating hardware that best fit target application.

Bibliography

[1] Michael Hucka, Andrew Finney, Herbert M. Sauro, H. Bolouri, John C. Doyle, Hiroaki Kitano, Adam P. Arkin, Benjamin J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, Serge Dronov, Ernst Dieter Gilles, Martin Ginkel, Victoria Gor, Igor Goryanin, W. J. Hedley, T. Charles Hodgman, J. H. Hofmeyr, Peter J. Hunter, Nick S. Juty, J. L. Kasberger, Andreas Kremling, Ursula Kummer, Nicolas Le Novère, Leslie M. Loew, D. Lucio, Pedro Mendes, E. Minch, Eric Mjolsness, Yoichi Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, James C. Schaff, Bruce E. Shapiro, Thomas Simon Shimizu, Hugh D. Spence, Jörg Stelling, Koichi Takahashi, Masaru Tomita, J. Wagner, and J. Wang. The systems biology markup language(sbml): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19:524–531, 2003.

[2] Bruce A. Barshop, Richard F. Wrenn, and Carl Frieden. Analysis of numerical methods for computer simulation of kinetic processes: Development of kinsim — a flexible, portable system. *Analytical Biochemistry*, 130:134–145, 1983.

[3] Pedro Mendes. Gepasi: a software package for modeling the dynamics, steady states and control of biochemical and other systems. *Computer Applications in the Biosciences*, 9(5):563–571, Oct. 1993.

[4] Masaru Tomita, Kenta Hashimoto, Kouichi Takahashi, Thomas Simon Shimizu, Yuri Matsuzaki, Fumihiko Miyoshi, Kanako Saito, Sakura Tanida, Katsuyuki Yugi, J. Craig Venter, and Clyde A. HutchisonIII. E-cell: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84, Jan. 1999.

[5] James Schaff, Charles C. Fink, Boris Slepchenko, John H. Carson, and Leslie M. Loew. A general computational framework for modeling cellular struc ture and function. *Biophysical Journal*, 73:1135–1146, Sep. 1997.

[6] James Schaff and Leslie M. Loew. The virtual cell. In *Proceedings of Pacific Symposium on Biocomputing*, volume 4, pages 228–239, Jan. 1999.

[7] Leslie M. Loew and James Schaff. The virtual cell: a software environment for computational cell biology. *Trends in Biotechnology*, 19(10):401–406, Oct. 2001.

[8] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.

[9] Andrzej M. Kierzek. Stocks: Stochastic kinetic simulations of biochemical systems with gillespie algorithm. *Bioinformatics*, 18(3):470–481, Mar. 2002.

[10] Nicolas Le Novère and Thomas Simon Shimizu. Stochsim: modelling of stochastic biomolecular processes. *Bioinformatics*, 17(6):575–576, Jun. 2001.

[11] Yasunori Osana, Tomonori Fukushima, and Hideharu Amano. Implementation of recsip: a reconfigurable cell simulation platform. In *The 13th International Conference on Field Programmable Logic and Applications*, volume 2778 of *Lecture Notes in Computer Science*, pages 766–775. Springer, Sep. 2003.

[12] Yasunori Osana, Tomonori Fukushima, Masato Yoshimi, and Hideharu Amano. An FPGA-based acceleration method for metabolic simulation. *IEICE Trans. on Information and Systems*, E87-D(8):2029–2037, Aug. 2004.

[13] ReCSiP Project Team. ReCSiP project frontpage. http://recsip.org/.

[14] James Dewey Watson and Francis Harry Compton Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, Apr. 1953.

[15] Daniel T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, May. 2007.

[16] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2461, Dec. 1977.

[17] Hiroyuki Kurata, Hana El-Samad, Tau-Mu Yi, Mustafa Khammash, and John C. Doyle. Feedback regulation of the heat shock response in e. coli. In *Proceedings of the 40th IEEE Conference on Decision and Control*, pages 837–842, 2001.

[18] Hong Li, Yang Cao, Linda R. Petzold, and Daniel T. Gillespie. Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnology Progress*, 24(1):56–61, Sep. 2007.

[19] Yang Cao, Hong Li, and Linda Petzold. Efficient formulation of the stochastic simulation algorithm for chemically recting systems. *Journal of Chemical Physics*, 121(9):4059–4067, 2004.

[20] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.

[21] Daniel T. Gillespie. Approximate acclerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, 115(4):1717–1733, Jul. 2001.

[22] Muruhan Rathinam, Linda R. Petzold, and Yang Cao. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *Journal of Chemical Physics*, 119(24):12784–12794, 2003.

[23] Yang Cao and Linda R. Petzold. Trapezoidal tau-lepaing formula for the stochastic simulation of biochemical systems. In *Proceedings of Foundations of Systems Biology in Engineering (FOSBE 2005)*, pages 149–152, 2005.

[24] Andrzej M. Kierzek, Jolanta Zaim, and Piotr Zielenkiewicz. The effect of transcription and translation initiation frequencies on the stochastic fluctuations in prokaryotic gene expression. *Journal of Biological Chemistry*, 276(11):8165–8172, Mar. 2001.

[25] Adam Arkin, John Ross, and Harley H.McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected escherichia coli cells. *Genetics*, 149:1633–1648, Aug. 1998.

[26] George Marsaglia, Arif Zaman, and Wai Wan Tsang. Toward a universal random number generator. *Letters in Statistics and Probability*, 9(1):35–39, Jan. 1990.

[27] Nicolas Le Novère. Stochsim, a stochastic biochemical simulator. http://www.ebi.ac.uk/~lenov/stochsim.html.

[28] Bray Group: Computer Models of Bacterial Chemotaxis. Stochsim. http://www.pdn.cam.ac.uk/groups/comp-cell/StochSim.html, May. 2006.

[29] Kouichi Takahashi, Katsuyuki Yugi, Kenta Hashimoto, Yohei Yamada, Christopher J. F. Pickett, and Masaru Tomita. A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, 20(4):538–546, Mar. 2004.

[30] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, Jan. 1998.

[31] Yang Cao, Andrew Hall, Hong Li, Sotiria Lampoudi, and Linda Petzold. User's guide for stochkit. http://www.engineering.ucsb.edu/~cse/StochKit/StochKitUserGuide.pdf.

[32] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. The slow-scale stochastic simulation algorithm. *Journal of Chemical Physics*, 122(1):014116–1–18, 2005.

[33] Stephen D. Brown et al. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[34] Xilinx Inc. *Virtex-II Platform FPGA User Guide*, v2.0 edition, Mar. 2005.

[35] Xilinx Inc. *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*, v4.0 edition, Mar. 2005.

[36] Atsushi Kawai, Toshiyuki Fukushige, Jun ichiro Makino, and Makoto Taiji. Grape-5: A special-purpose computer for n-body simulations. *Publ. of the Astronomical Society of Japan*, 52:659–676, Aug. 2000.

[37] Naohito Nakasato and Tsuyoshi Hamada. Astrophysical hydrodynamics simulations on a reconfigurable system. In *Proceedings of the 13th IEEE Workshop on FPGAs for Custom Computing Machines*, pages 279–280, Apr. 2005.

[38] Tsuyoshi Hamada, Naohito Nakasato, and Toshikazu Ebisuzaki. A 236 gflops astrophysical simulation on a reconfigurable super-computer. In *Proceedings of IEEE/ACM SC 2005 Conference*, Nov. 2005.

[39] Yongfeng Gu, Tom Van Court, and Martin Herbordt. Accelerating molecular dynamics simulations with reconfigurable circuits. In *Proceedings of the 15th International Conference on Field-Programmable Logic and Applications*, pages 475–480, Aug. 2005.

[40] John F. Keane, Christpher Bradley, and Carl Ebeling. A compiled accelerator for biological cell signaling simulations. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 233–241, Feb. 2004.

[41] Larry Lok. The need for speed in stochastic simulation. *Nature Biotechnology*, 22(8):964–965, Aug. 2004.

[42] Lukasz Salwinski and David Eisenberg. In silico simulation of biological network dynamics. *Nature Biotechnology*, 22(8):1017–1019, Aug. 2004.

[43] Brandon Thurmon, James M. McCollum, Gregory D. Peterson, Chris D. Cox, Nagiza F. Samatova, Gary Sayler, and Michael L. Simpson. Accelerating exact stochastic simulation using reconfigurable computing. In *International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2005.

[44] Philip Heng Wai Leong, Monk Ping Leong, O. Y. H. Cheung, T. Tung, C. M. Kwok, M. Y. Wong, and Kin Hong Hong Lee. Pilchard – a reconfigurable computing platform with memory slot interface. In *Proceedings of the 9th IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 170–179, 2001.

[45] Yasunori Osana, Tomonori Fukushima, and Hideharu Amano. Implementation of recsip: a reconfigurable cell simulation platform. In *The 13th International Conference on Field Programmable Logic and Applications(FPL)*, pages 766–775, Sep. 2003.

[46] Masato Yoshimi, Yasunori Osana, Tomonori Fukushima, and Hideharu Amano. Stochastic simulation for biochemical reactions on FPGA. In *The 14th International Conference on Field Programmable Logic and Applications*, volume 3203 of *Lecture Notes in Computer Science*, pages 105–114. Springer, Aug. 2004.

[47] Takakazu Kurokawa. Hardware implemenrtation of mersenne twister using fpga. In *Proceedings of the 2001 International Technical Conference on Circuits/Systems, Computers and Communications*, pages 307–310, July 2001.

[48] Shiro Konuma and Shinichi Ichikawa. Design and evaluation of hardware pseudo-random number generator mt19937. *IEICE TRANSACTIONS on Information and Systems*, E88-D(12):2876–2879, Dec. 2005.

[49] Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hideki Yamada, Hiroaki Kitano, and Hideharu Amano. FPGA Implementation of a data-driven Stochastic Biochemical Simulator with the Next Reaction Method. In *The 17th International Conference on Field Programmable Logic and Applications(FPL'07)*, pages 254–259. IEEE, Aug. 2007.

[50] Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano, and Hideharu Amano. A framework for implementing a network-based stochastic biochemical simulator on an fpga. In *International Conference on Field-Programmable Technology(ICFPT'07)*, pages 193–200, Dec. 2007.

[51] Tokyo Electron Device. Virtex-5 LXT/SXT PCI Express Evaluation Platform Board. http://www.inrevium.jp/eng/x-fpga-board/hibiki.html.

[52] Chen Chang, John Wawrzynek, and Robert W. Brodersen. Bee2: A high-end reconfigurable computing system. *IEEE Design and Test of Computers*, 22(2):114–125, Mar. 2005.

[53] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE, New York, Aug. 1985.

Publications

Journal Papers

[1] Masato Yoshimi, Yuri Nishikawa, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, "Design and Evaluation of an FPGA-based Stochastic Biochemical Simulator for High-throughput Execution", *IPSJ transactions on Advanced Computers System*, Vol.1, No.3, pp.120-135, December, 2008. (In Japanese).

[2] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "A Biochemical Model Compiler for Simulations on an FPGA", *IEICE Transactions on Information and Systems*, Vol.J91-D, No.9, pp.2205-2216, September, 2008. (In Japanese).

[3] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "ReCSiP: An FPGA-based general-purpose biochemical simulator", *Electronics and Communications in Japan (Part II:Electronics)*, Vol.90, No.7, pp.1-10, July, 2007.

[4] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Yuichiro Shibata, Naoki Iwanaga, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Hideharu Amano, "FPGA-based Stochastic Biochemical Simulator", *IPSJ transactions on Advanced Computers System*, Vol.48, No.SIG3 ACS17, pp.45-58, February, 2007. (In Japanese).

[5] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "ReCSiP: A FPGA-Based General Purpose Biochemical Simulator", *IEICE Transactions on Information and Systems*, Vol.J89-D, No.6, pp. 1163-1172, January, 2006. (In Japanese).

[6] Yasunori Osana, Tomonori Fukushima, Masato Yoshimi and Hideharu Amano, "An FPGA-Based Acceleration Method for Metabolic Simulation", *IEICE Transactions on Information and Systems*, Vol.E87-D, No.8, pp.2029-2037, August, 2004.

International Conference Papers

[7] Masato Yoshimi, Yuri Nishikawa, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, "Practical Implementation of a Network-Based Stochastic Biochemical Simulation System on an FPGA", The 18th International Conference on Field Programmable Logic and Applications (FPL'08), pp. 663-666, August, 2008.

[8] Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, "A Framework

for Implementing a Network-Based Stochastic Biochemical Simulator on an FPGA, International Conference on Field-Programmable Technology(ICFPT'07)", pp.193-200, December, 2007.

[9] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Kenichi Miura and Hideharu Amano, "Performance Improvement Methodology for ClearSpeed's CSX600", The 2007 International Conference on Parallel Processing(ICPP-07), September, 2007.

[10] Hideki Yamada, Naoki Iwanaga, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Kiyoshi Oguri, "A Combining Technique of Rate Law Functions for a Cost-effective Reconfigurable Biological Simulator", The 17th International Conference on Field Programmable Logic and Applications (FPL'07), pp. 808-811, August, 2007.

[11] Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, "FPGA Implementation of a data-driven Stochastic Biochemical Simulator with the Next Reaction Method", The 17th International Conference on Field Programmable Logic and Applications(FPL'07), IEEE, pp.254-259, August, 2007.

[12] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "A Hardware Accelerator for Biochemical Simulations", Winter Simulation Conference 2006 (WSC06), CD-ROM, December, 2006,

[13] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Hardware Design of a Stochastic Biochemical Simulator", Winter Simulation Conference 2006 (WSC06), CD-ROM, December, 2006.

[14] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "An Acceleration of a Biochemical Simulator on Programmable Hardware", The Seventh International Conference on Systems Biology(ICSB2006), September, 2006.

[15] Miyashiro, T., Kitamura, A., Masato Yoshimi, Hideharu Amano, Nakajyo, H. and Tanabe, N., "DIMMNET-2: A Reconfigurable Board Connected into a Memory Slot", The 16th International Conference on Field Programmable Logic and Applications (FPL'06), pp. 825-828, August, 2006.

[16] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "An FPGA Implementation of High Throughput Stochastic Simulator for Large-Scale Biochemical Systems", The 16th International Conference on Field Programmable Logic and Applications(FPL'06), pp.227-232, August, 2006.

[17] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Performance Evaluation of an FPGA-Based Biochemical Simulator ReCSiP", The 16th International Conference on Field Programmable Logic and Applications(FPL'06), IEEE, pp.845-850, August, 2006.

[18] Daihan Wang, Hiroki Matsutani, Masato Yoshimi, Hideharu Amano and Michihiro Koibuchi, "A Parametric Study of Scalable Interconnects on FPGAs", The 2006 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'06), CD-ROM, June, 2006.

[19] Yuri Nishikawa, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "An Evaluation and Investigation of Dual-Thread Numerical Integration Mechanism for FPGA-based Biochemical Simulator ReCSiP", Coolchips IX, April, 2006.

[20] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "The Design of Scalable Stochastic Biochemical Simulator on FPGA", 2005 IEEE International Conference on Field Programmable Technology (FPT'05), IEEE, pp.339-340, December, 2005.

[21] Naoki Iwanaga, Yuichiro Shibata, Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Tomonori Fukushima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Kiyoshi Oguri, "Efficient Scheduling of Rate Law Functions for ODE-based Multimodel Biochemical Simulation on an FPGA", Proceedings of the 15th Field Programmable Logic and its applications (FPL'05), IEEE, pp. 666-669, August, 2005.

[22] Yasunori Osana, Yow Iwaoka, Tomonori Fukushima, Masato Yoshimi, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "A Framework for ODE-Based Multimodel Biochemical Simulations on an FPGA", Proceedings of the 15th Field Programmable Logic and its applications (FPL'05), IEEE, pp.574-577, August, 2005.

[23] Yasunori Osana, Tomonori Fukushima, Masato Yoshimi, Yow Iwaoka, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hiroaki Kitano and Hideharu Amano, "An FPGA-Based, Multi-model Simulation for Biochemical Systems", Proceedings of the 19th International Parallel and Distributed Processing Symposium / Reconfigurable Architecture Workshop(RAW'05), IEEE, CD-ROM, April, 2005.

[24] Masato Yoshimi, Yasunori Osana, Tomonori Fukushima and Hideharu Amano, "Stochastic Simulation for Biochemical Reactions on FPGA", The 14th International Conference on Field Programmable Logic and Applications (FPL'04), Springer, Vol.3203, pp.105-114, August, 2004.

[25] Masato Yoshimi, Yasunori Osana, Tomonori Fukushima and Hideharu Amano, "Implementation and Evaluation of Stochastic Simulation of Chemical Reaction Model on FPGA", COOL Chips VII, pp.83, April2004.

Domestic Conference Papers and Technical Reports

Original Papers

[26] Masato Yoshimi, Yuri Nishikawa, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, "Proposal of FPGA-based calculation system exploiting thread level parallelism for scientific applications", IPSJ SIG Notes, IPSJ, Vol. 2008, No.101, pp.63-66, October, 2008. (In Japanese).

[27] Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, "Evaluation of a Data-Driven Architecture for a Stochastic Biochemical Simulator on an FPGA", IEICE Technical Report, Vol.107, No.342, pp.43-48, November, 2007. (In Japanese).

[28] Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Yuichiro Shibata, Naoki Iwanaga, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Hideharu Amano, "Empirical Analysis on Implementation of High-Speed Stochastic Biochemical Simulator on an FPGA", 29th Parthenon workshop, Vol.1, No.5, pp.29-36, December, 2006. (In Japanese).

[29] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Yuichiro Shibata, Naoki Iwanaga, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Hideharu Amano, "FPGA-based Stochastic Biochemical Simulator", Proccedings of the Symposium on Advanced Computing Systems and Infrastructures 2006, No.5, pp. 151-158, May, 2006. (In Japanese).

[30] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Tomonori Fukushima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Design of a Heap-tree Based Scalable Stochastic Biochemical Simulator on an FPGA", IEICE Technical Report, Vol.105, No.42, pp.37-42, May, 2005. (In Japanese).

[31] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuichiro Shibata, Naoki Iwanaga and Hideharu Amano, "Design of Floating-Point Arithmetic and Logic Units for Computational Science and Technology on FPGA", 26th Parthenon workshop, Vol.1, No.7, pp.49-55, May, 2005. (In Japanese).

[32] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "A Stochastic Biochemical Simulator with a Data-transfer Network on an FPGA", IEICE Technical Report, Vol.105, No.518, pp.47-52, 2005. (In Japanese).

[33] Masato Yoshimi, Yasunori Osana, Tomonori Fukushima and Hideharu Amano, "Acceleration Hardware of Stochastic Simulation for Biochemical Reactions on FPGA", 4th Workshop on Reconfigurable Systems, No.33, pp. 220-226, September, 2004. (In Japanese).

Joint Papers

[34] Tomoaki Tsumura, Masato Yoshimi, Takashi Nakada, Takahiro Katagiri and Kenji Kise, "The Report on "Cell Speed Challenge 2008" ", IPSJ SIG Notes, IPSJ, Vol.2008, No.75, pp.103-108, August, 2008. (In Japanese).

[35] Tomoya Ishimori, Hideki Yamada, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi and Kiyoshi Oguri, "Pipeline Scheduling with Input Port Constraints for an FPGA-based Biochemical Simulator", IEICE Technical Report, Vol.108, No.48, pp.113-118, May, 2008. (In Japanese).

[36] Hideki Yamada, Tomoya Ishimori, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Hideharu Amano, Akira Funahashi, Noriko Hiroi and Kiyoshi Oguri, "An automatic combine algorithm of arithmetic pipelines for an FPGA-based biochemical simulator focused on similarities of rate law functions", IEICE Technical Report, Vol.108, No.220, pp.21-26, September, 2008. (In Japanese).

[37] Hideki Yamada, Naoki Iwanaga, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Kiyoshi Oguri, "Automatic combining of rate law functions for an FPGA-based biochemical simulator ReCSiP", IEICE Technical Report, Vol.107, No.41, pp.13-18, May, 2007. (In Japanese).

[38] Kenji Kise, Masato Yoshimi, Takahiro Katagiri and Hiroshi Nakamura, "The Report on Multicore Programming Contest Cell Speed Challenge 2007", IPSJ SIG Notes, IPSJ, Vol.2007, No.79, pp.193-198, August, 2007. (In Japanese).

[39] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi and Hideharu Amano, "A Proposal of Performance Improvement Based on A Parallel Benchmark Evaluation on a ClearSpeed Coprocessor", IPSJ SIG Notes, IPSJ, Vol.2007, No.17, pp.257-262, March, 2007. (In Japanese).

[40] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Kenichi Miura and Hideharu Amano, "A Study of Time Prediction Method for Running Parallel Applications on ClearSpeed's SIMD-Based Multi-Core Processor", IPSJ SIG Notes, IPSJ, Vol.2007, No.79, pp.43-48, August, 2007. (In Japanese).

[41] Toshinori Kojima, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Funahashi, H., Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Implementation and Evaluation of General-Purpose Host Interface on ReCSiP Board", IEICE Technical Report, Vol.106, No.49, pp.55-60, May, 2006. (In Japanese).

[42] Yuri Nishikawa, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "A Performance Improvement Strategy for Numerical Integration on an FPGA-Based Biochemical Simulator ReCSiP", IEICE Technical Report, Vol.105, No.518, pp.53-58, January, 2006. (In Japanese).

[43] Yasunori Osana, Naoki Iwanaga, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hiroaki Kitano and Hideharu Amano, "Hardware-resource Utilization Analysis on an FPGA-Based Biochemical Simulator ReCSiP", IEICE Technical Report, Vol.105, No.518, pp.59-64, January, 2006. (In Japanese).

[44] Daihan Wang, Hiroki Matsutani, Masato Yoshimi, Michihiro Koibuchi and Hideharu Amano, "A Parametric Study of Packet-Switched FPGA Overlay Networks", IEICE Technical Report, Vol.106, No.247, pp.31-36, September, 2006. (In Japanese).

[45] Naoki Iwanaga, Yuichiro Shibata, Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Tomonori Fukushima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Kiyoshi Oguri, "Scheduling of Rate Law Functions for an FPGA-based Biochemical Simulator", IEICE Technical Report, Vol.105, No.42, pp.43-48, May, 2005. (In Japanese).

[46] Yow Iwaoka, Yasunori Osana, Tomonori Fukushima, Masato Yoshimi, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Design of cellular simulation platform for SBML model", IEICE Technical Report, Vol.104, No.591, pp.63-68, January, 2005. (In Japanese).

[47] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "A System Design of Accelerating Biochemical Simulations on Programmable Hardware", The 10th Forum on Software Platforms for Systems Biology, October, 2005. (In Japanese).

[48] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Kojima, Y., Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Building of the SBML System for an FPGA-based Biochemical Simulator", IEICE Technical Report, Vol.105, No.287, pp.61-66, September, 2005, (In Japanese).

[49] Yasunori Osana, Tomonori Fukushima, Masato Yoshimi, Yow Iwaoka, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Design of the SBML Processor for an FPGA-based Biochemical Simulator", IPSJ SIG Notes, IPSJ, Vol.2005, No.7, pp.13-18, January, 2005. (In Japanese).

[50] Yasunori Osana, Masato Yoshimi and Hideharu Amano, "A Feasibility Study on Reconfigurable Multi-cellular System Simulator", IPSJ SIG Notes, IPSJ, Vol.2005, No.120, pp.87-92, November, 2005. (In Japanese).

[51] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Implementation and Evaluation

of Numerical Integrators on ReCSiP", IEICE Technical Report, Vol.105, No.42, pp.49-54, May, 2005. (In Japanese).

[52] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, "Control Mechanism of the FPGA-Based Biochemical Simulator ReCSiP", IEICE Technical Report, Vol.105, No.287, pp.55-60, September, 2005. (In Japanese).

[53] Yasunori Osana, Tomonori Fukushima, Masato Yoshimi and Hideharu Amano, "Multi-Model Metabolic Simulation of Cellular System on FPGAs", 4th Workshop on Reconfigurable Systems, No.8, pp. 49-54, September, 2004. (In Japanese).

[54] Yasunori Osana, Tomonori Fukushima, Masato Yoshimi and Hideharu Amano, "Metabolic simulation of cellular system by FPGA-based system", 1st Workshop on Reconfigurable Systems, No.7, pp. 43-48, September, 2003. (In Japanese).

Appendix A

Implementation of the floating-point logarithmic function module

A.1 Floating point format

IEEE754 is a standard for defining real numbers in 32-bit or 64-bit format [53]. The former is called a single precision, and the latter is double precision. Both consists of three parts: sign, exponent, and mantissa. For a single precision, the sign s is expressed in 1-bit, exponent e is 8 bits and mantissa m is 23 bits. In case of double precision, s is 1-bit, e is 11 bits and m is 52 bits.

Fig. A.1 shows the format of single-precision floating-point format. When a number x is expressed in a single precision format, values of s, e and m must satisfy the Eq. A.1.

$$x = (-1)^s \times 2^{e-127} \times m \tag{A.1}$$

Exponent part e is biased by 127, and it is used to adjust the decimal point of the mantissa. Mantissa m is a fixed-point value in the range of $[1, 2)$, and the value is expressed in 23-bits (the MSB which is constantly fixed to 1 is abbreviated). In case of $x = 1.0$ as an example, the sign is plus, exponent is 0 and mantissa is 1.0. Thus, the parameters will be $s = 0$, $e = 127$,and $m = 0$.

There are following special data types for defining particular values:

Not-a-Number(NaN): e is a definable maximum value and m is other than zero

Infinity ($\pm\infty$) : e is a definable maximum value and m is zero (s decides the positive or the negative of infinity)

Zero(± 0) : e and m are zero (There are positive and negative zeros according to s)

In case when the MSB of m is 1, the value is called SNaN. It is called QNaN in case of 0.

IEEE754 standard defines five possible exceptions:

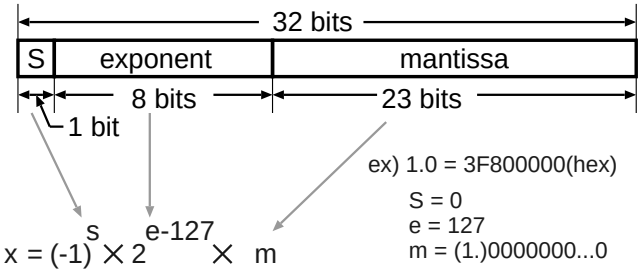


Fig. A.1: Format of a single-precision floating point

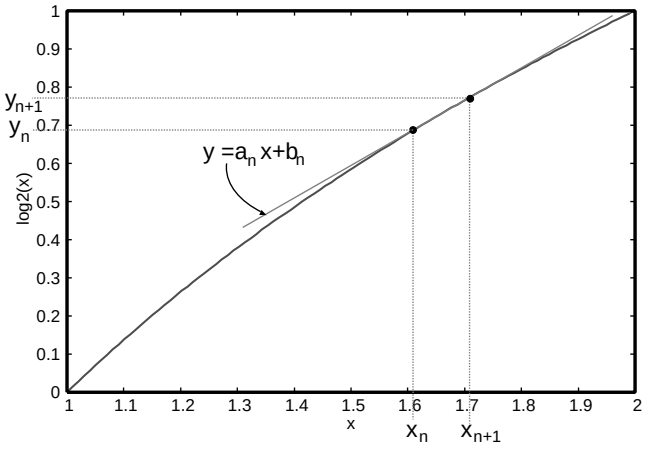


Fig. A.2: Linear interpolation

Invalid operation: Operations using infinity or NaN values, or when it fails to convert a value to integers

Division by zero: When non-zero value was divided by zero

Overflow: A result is too large to be represented

Underflow: A result is so small that it is out of definable range

Inexact: When one of the three rounding bits are 1

A.2 Logarithmic function module

A.2.1 Realization of logarithmic function

This section introduces a scheme to computes logarithmic value in this thesis. Logarithmic function module is implemented as a unit which computes logarithmic value of the input using second-order interpolation. The base is an arbitrary number.

Assume that the base is α and that the input is given as a single precision floating point format : $x = (-1)^{s_x} \times 2^{(e_x-127)} \times m_x$ ($s_x = 0$ or $1, 0 \leq e_x \leq 255, 1 \leq m_x < 2$). Then , $\log_\alpha(x)$ is yielded as:

$$\begin{aligned} \log_\alpha(x) &= \log_\alpha\left((-1)^{s_x} \times 2^{(e_x-127)} \times m_x\right) \\ &= \log_\alpha\left(2^{(e_x-127)}\right) + \log_\alpha(m_x) \\ &= \frac{\log_2\left(2^{(e_x-127)}\right) + \log_2(m_x)}{\log_2 \alpha} \\ &= \frac{(e_x - 127) + \log_2(m_x)}{\log_2 \alpha} \end{aligned} \tag{A.2}$$

Using this conversion, logarithmic function module with arbitrary base can be configured by computing $\log_2(m_x)$ whose m_x is within the range of [1,2). $\log_2(m_x)$ is obtained by second-order interpolation (it is done by correcting the value obtained in linear interpolation).

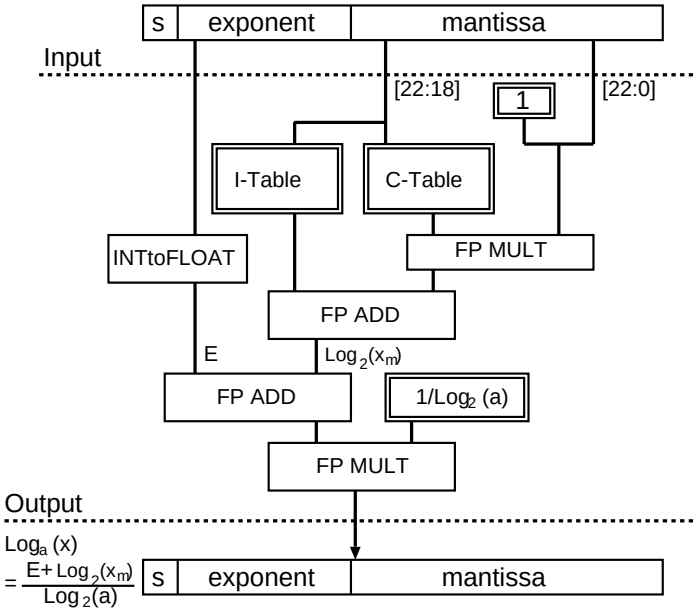


Fig. A.3: Operation flow of logarithmic function with linear interpolation

A.2.2 Linear(First order) interpolation

The $\log_2(x)$ curve is shown in Fig. A.2 whose x within the range of [1,2). x in Fig. A.2 corresponds to mantissa m_x of floating-point explained above.

In linear interpolation, $\log_2(x)$ curve is approximated with 1024 line segments. Each line is obtained by its coefficient a_i and intercept b_i $(i = 0, \cdots, 1023)$. The parameters a_i and b_i of linear line in Fig. A.2 are selected when input x within the range of $[x_i, x_{i+1})$.

To implement linear interpolation, two tables is used to store coefficients $\boldsymbol{a}$ (Coefficient Table : C-Table) and intercepts $\boldsymbol{b}$ (Intercept Table :I-Table). The operational flow is shown in Fig. A.3 $\log_2(x)$ in linear interpolation is obtained by following scheme :

1. Input x is decomposed to each part : s, e_x and m_x
2. Read coefficient table a_n and intercept table b_n according to its address n which is forehand 10 bits of mantissa part of m_x
3. Calculate $\log_2(m_x) = a_n m_x + b_n$
4. Calculate Eq. A.2 using the value of $\log_2(m_x)$

The reason to partition segments 1024 lines is to adjust a unit of BlockRAM in Xilinx's Virtex series. Because partition is divided 1024 line segments, forehand 10 bits of mantissa part of x is used to select an appropriate line segment.

A.2.3 Validation of linear interpolation

To investigate the error of linear interpolation, Fig. A.4 and Fig. A.5 show the error value of the $\log_2(m_x)$. The error value in their figures is derived by (primary value − linear interpolation value).

When the partition is divided in $32(= 2^5)$ line segments which is shown in Fig. A.4, the accuracy acquires about 12-bits. It is about four digits in decimal. Meanwhile, in the partition divided in $64(= 2^6)$ line segments, the accuracy increases about 14-bits. Although error value is adequately small as

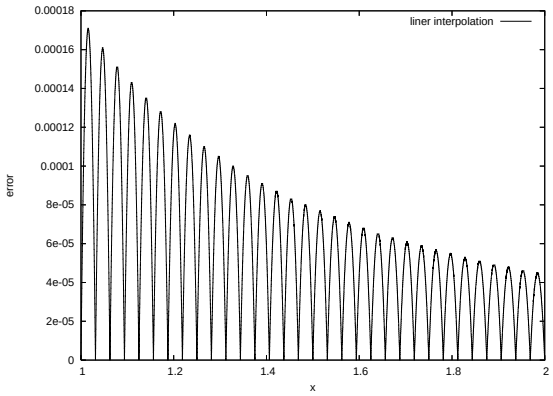


Fig. A.4: Error in linear interpolation(32 line segments)

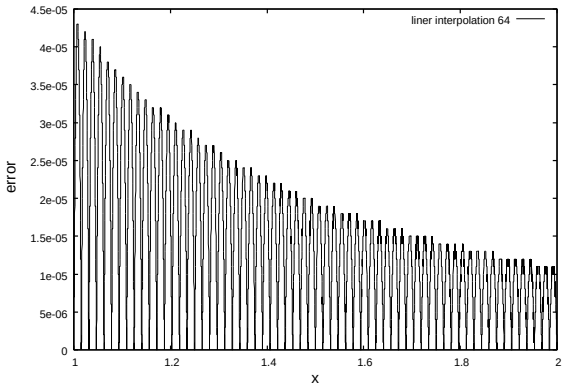


Fig. A.5: Error in linear interpolation(64 line segments)

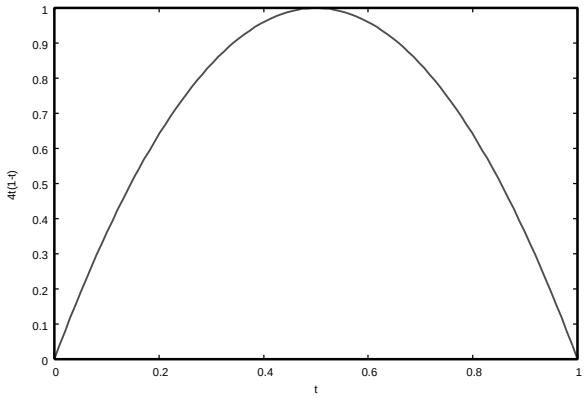


Fig. A.6: Approximated curve in a line segment of linear interpolation

against single-precision as implemented line segments are $1024(= 2^{10})$, almost linear interpolation values are smaller than ideal values. To solve this problem, second-order interpolation is introduced.

A.2.4 Second-order interpolation

To adjust the difference between linear interpolation value and actual value, second-order interpolation is introduced. Error function between in each line segment used for linear interpolation and actual value of $\log_2(x)$ draws a quadratic curve as Fig. A.6. The curve is descripted as Eq. A.3 where t is the inside segment in the line segment. u indicates a ratio compared to the maximum difference between interpolation value and actual value.

$$u = 4t(t-1) = 4t^2 - 4t \tag{A.3}$$

Second-order interpolation is done by adding linear interpolation value and differential value. This method can be utilized by introducing a table which is stored difference between for each line segments in the midpoint of the segment. The table is called differential table(D-Table). Offset in second-order interpolation is calculated using the value from D-Table and a ratio u.

To implement of second-order interpolation, an additional table (D-Table) is exploited in addition of linear interpolation. It stores difference d_n between linear approximation $y = ax + b$ and $\log_2(x)$, which is shown in Eq. A.4. D-Table is read by the address as same as other two tables in linear

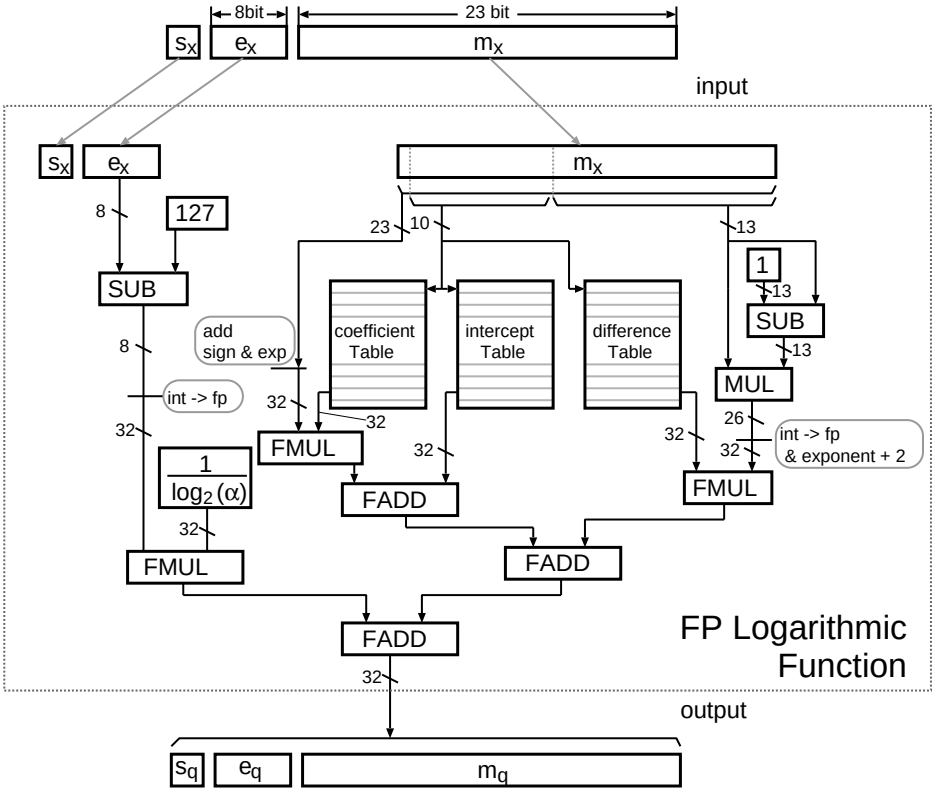


Fig. A.7: Block diagram of logarithmic function with second-order interpolation

interpolation. Eleventh to 20th bit of mantissa part of x is used as t whose format is the fixed-point number. The digit point is located in the left of MSB.

$$d_n = \log_2\left(\frac{x_n + x_{n+1}}{2}\right) - a_n\left(\frac{x_n + x_{n+1}}{2}\right) + b_n \tag{A.4}$$

The process for obtaining $\log_\alpha(x)$ are as follows:

1. Upper 10 bits n of mantissa part m_x (besides MSB) is used to read coefficient a_n, intercept b_n, differential d_n from corresponding tables.
2. Linear-interpolated value y_l is computed by $a_n \times m_x + b_n$ (linear interpolation).
3. Eleventh to 20th bit of fractional part of mantissa t is chosen. Then, $y_s = d_n \times t \times (1.0 - t) \times 4$ is computed and added to y_l (second-order interpolation). y_s is equal to $\log_2(m_x)$.
4. $\log_\alpha(x)$ is introduced by adding y_s and $(e - 127)$, and multiplying its result and inverse number of $\log_2(\alpha)$

In our implementation, value of $1/\log_2(\alpha)$ was set in arithmetic units in advance so as to omit its computing process. Fig. A.7 shows a hardware connection diagram of logarithmic function with second-order interpolation. Operation for quadruple of $t(t-1)$ in Eq. A.3 is utilized by 2-bit left shifting instead of multiplication.

A.2.5 Validation of second-order interpolation

Fig. A.8 shows the error value of the $\log_2(m_x)$ when the fractional part inside a line segment is is divided in $32(= 2^5)$. From Fig. A.8, the accuracy acquires about 23-bits. Moreover, as differences of

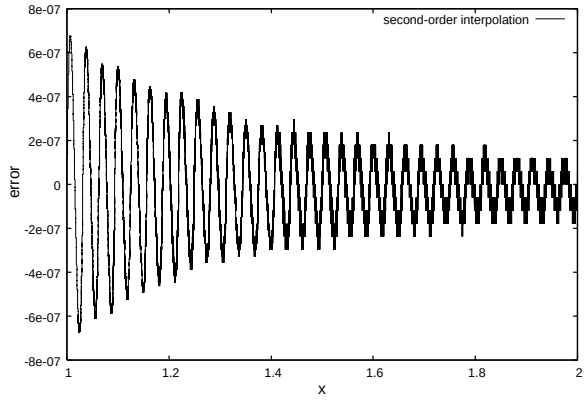


Fig. A.8: Error in second-order interpolation(32 line segments)

actual number are almost equally distributed positive and negative regions, second-order interpolation is superior than linear interpolation.

Appendix B

Derivation of Stochastic Biochemical Simulation Algorithm

B.1 Methods for biochemical simulation

There are two methods to mathematically describe a behavior of a biochemical system in arbitrary space, (1) deterministic method and (2) stochastic method.

The former is called a deterministic method, and time variation of concentration of molecules is determined by a set of ordinary differential equation (which is abbreviated to ODE).

On the other hand, stochastic approach assumes time variation of populations of chemical species as a random walk process determined by a single differential-difference equation. Calculation methodology of stochastic chemical reacting model originally derives from a stochastic and statistical approach called master equation. However, it is difficult to mathematically solve a master equation in general, so it is often solved by introducing equivalent numerical calculation method.

Gillespie's "Stochastic Simulation Algorithm" [16] (abbreviated as SSA for the rest of this appendix) is an algorithm to simulate behavior of biochemical models by applying a computation method of general stochastic models.

SSA was developed to find solutions to following problems:

> How is the variability of populations over time when initial values are given for N types of molecules which uniformly distributes in a space whose volume is constantly V, are given, and the number of their cross-reactions are M?

B.2 Solution for an analytical model

To solve the problem above, it was conventionally general to take an analytical approach by describing the problem with ordinary differential equations.

Here, we define a biochemical system, whose index populations is defined by a nonlinear function $X_i(t)$ $(i = 1, \ldots, N)$. Here, t is time and i is the index number that is unique for each type of molecules. Assuming that the number of M types of molecules change continuously, it yields following first-degree ordinary differential equation:

$$\frac{dX_i}{dt} = f_i(X_1, \cdots, X_N) \qquad (i = 1, \cdots, N) \tag{B.1}$$

Here, function f_i is determined by the degree (monomolecular, bimolecular, or higher) of reaction M and its reaction rate constant. These equations are called reaction-rate equation. Trajectory of $X_1(t), \ldots, X_N(t)$ represents time variation of each molecules.

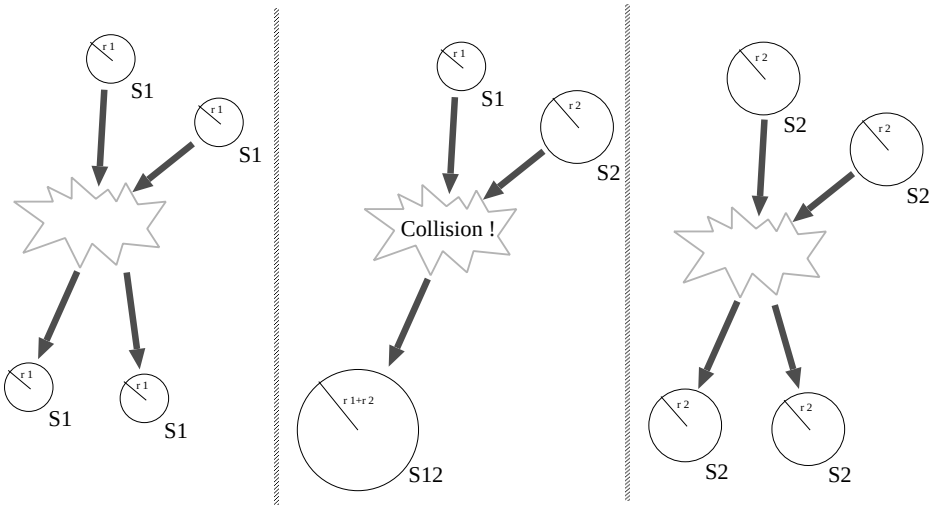


Fig. B.1: Collision of molecules

Analytical method of such reaction-rate equations are often applied to simple computer systems. Because it an approximate solution for ordinary differential equations, it implicitly assumes that a time change of a target biochemical system is continuous and deterministic. However, population is obviously a discrete integer value, and time variation of actual biochemical system is naturally discontinuous. Even when behavior of molecules are governed by certain equation, it is impossible to simulate a behavior of whole system unless positions and velocities of all molecules are precisely computed.

Another drawback of deterministic approach is error rate. Marginal error of initial rate may grow at an exponential manner as simulation proceeds, because the small error is integrated by analytical method of ordinary differential equations. It is also unsuitable for simulating phenomena that goes through a rapid change, such as regulation of gene expressions. In such cases, small populations react in a short period of time, and their reaction involves stochastic factors which makes it difficult to solve with deterministic approach.

B.3 Formulation of chemical reactions with stochastic models

Chemical reactions occur when two or more molecules collide with each other. SSA computes the collision of molecules under a condition of thermal equilibrium.

B.3.1 Collision of molecules

Assume that there is a system in a equilibrium and consists of two types of gas molecules S_1 and S_2. For simplification, radii of S_1 and S_2 are r_1 and r_2, respectively, and their shape are perfect spheres. Reaction always occur when the center of two spheres overlap, and they become a molecule whose radius is $r_{12} = r_1 + r_2$, as shown in Fig. B.1.

Based on the conditions above, frequency of the collision (or reaction rate) in a space V is computed as follows. Conventionally, two molecules that react are randomly selected from arbitrary pairs. Next, v_{12}, a relative velocity of molecular movement for S_2 against S_1, is computed. Also, collision volume, a volume that migrates from S_1 to S_2 within time period Δt, is also computed

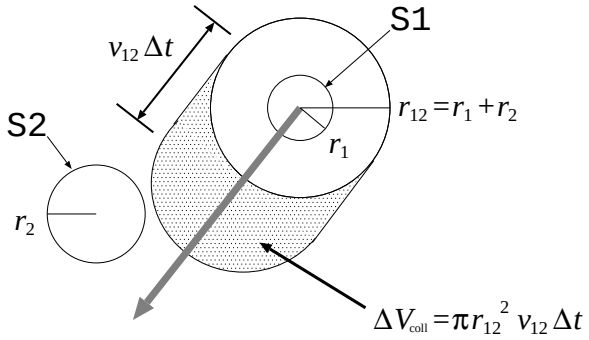


Fig. B.2: Collision volume ΔV_{coll}

(Fig. B.2). The collision volume ΔV_{coll} is obtained by the following equation.

$$\Delta V_{\text{coll}} = \pi r_{12}^2 \cdot v_{12}\Delta t \tag{B.2}$$

If the center of S_2 were inherent in space ΔV_{coll} at time t, two molecules collide within time period $(t, t + \Delta t)$. Accuracy of this method improves by reducing the value of timestep Δt. However, with $\Delta V_{\text{coll}} \to 0$, population inside collision volume become 0 or 1. As this indicates, it is not always rigorous to simulate by introducing minimal timestep, and more accurate results can be obtained by stochastic approach.

Because the system is thermally equilibrium, molecules uniformly distributes inside space V. Thus, the probability for the center of arbitrary S_2 molecule is inside ΔV_{coll} at time t is simply obtained by $\Delta V_{\text{coll}}/V$. Hence, the average of reaction rate distribution of S_1 and S_2 is obtained as Eq. B.3. This value is equal to the ratio of collision volume against the volume of the whole system.

$$\overline{\left(\frac{V_{\text{coll}}}{V}\right)} = \frac{\pi r_{12}^2 \overline{v_{12}}\Delta t}{V} = \text{Average of probability that } S_1 \text{ and } S_2 \text{ collides within minimal time } \Delta t \tag{B.3}$$

According to Maxwell's velocity distribution, average of relative rate $\overline{v_{12}}$ is $(8kT/\pi m_{12})^{1/2}$, where k is Boltzmann coefficient, T is the absolute temperature, and m_{12} is reduced mass obtained by $m_1 m_2/(m_1 + m_2)$. Eq. B.3 assumes that there are only one molecule in V for each S_1 and S_2. If each population is X_1 and X_2, Eq. B.3 is rewritten as Eq. B.4.

$$\overline{\left(\frac{V_{\text{coll}}}{V}\right)} = \frac{X_1 X_2 \pi r_{12}^2 \overline{v_{12}} dt}{V} = \text{Probability that } S_1 \text{ and } S_2 \text{ collide within time } (t, t + dt) \tag{B.4}$$

It is difficult to deterministically obtain the number of reaction that occurred, but it is possible to compute the probability of collision in V within limited time. Eq. B.4 exhibits stochastic Markov process instead of deterministic reaction rate equation. Thus, it is not affected by past behavior, and only current population of molecules and "probability of collision within time unit" are the only factors that characterizes the system in thermal equilibrium.

B.3.2 Stochastic reaction rate constant

As described in the previous section, reaction rate of chemical reaction can be defined as "Probability of collision within time unit = probability for occurrence of reaction within time unit".

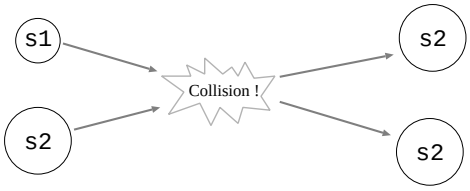


Fig. B.3: Assumed reaction

Assume that molecules S_1 and S_2 react according to Eq. B.5.

$$R_1 : \quad S_1 + S_2 \rightarrow 2S_2 \tag{B.5}$$

Eq. B.3 determines c_1, which is a constant that only depends on their physical characteristics and system temperature.

$$\begin{aligned} c_1 dt \quad &= \quad \text{Probability that } S_1 \text{ and } S_2 \text{ react} \\ &\quad\ \ \text{with Reaction } R_1 \text{ within minimal timestep } dt \end{aligned} \tag{B.6}$$

Following equation as approved when the populations of molecules in volume V are X_1 and X_2 for S_1 and S_2 at time t:

$$\begin{aligned} X_1 X_2 c_1 dt \quad &= \quad \text{Probability that Reaction } R_1 \text{ occur} \\ &\quad\ \ \text{inside } V \text{ within minimal timestep } (t, t + dt) \end{aligned} \tag{B.7}$$

Generalization of Eq. B.7 yields Eq. B.8. Assume that there are N types of molecules in volume V, and population of molecule S_i is X_i. If M types of molecules cross-react with equation R_μ inside the volume, we can determine M types of constant c_μ with Eq. B.8. The constant only depends on physical properties of molecules and system temperature:

$$\begin{aligned} c_\mu dt \quad &= \quad \text{Average of probability that molecules involving to} \\ &\quad\ \ \text{reaction } R_\mu \text{ collides within minimal timestep } dt \end{aligned} \tag{B.8}$$

The term "average" in Eq. B.8 means that a multiple of $c - \mu$ and a combination number for whole pair of molecules involved in the reaction R_μ is equal to the probability that reaction R_μ occurs in volume V during $(t, t + dt)$.

B.3.3 Relationship between reaction rate constant and stochastic reaction rate constant

Stochastic reaction rate constant c_μ has a close connection with reaction rate constant k_μ. Following equation is formed for a reaction R_1 in Eq. B.5.

$$k_1 = \frac{V c_1 \langle X_1 X_2 \rangle}{\langle X_1 \rangle \langle X_2 \rangle} \qquad (\langle x \rangle \text{ is an average of } x) \tag{B.9}$$

Right-hand side of Eq. B.9 can be interpreted as $\langle X_1 X_2 \rangle = \langle X_1 \rangle \langle X_2 \rangle$. Hence,

$$k_1 = V c_1 \tag{B.10}$$

If reaction R_1 have three reactants, parameter of volume changes from V to V^2. Also, if reaction R_1 only has one reactant (such as cases of isomerization), V is equal to 1.

Next, we are going to consider a counterreaction of R_1 based on Eq. B.11.

$$R_2 : \quad 2S_1 \rightarrow S_1 + S_2 \tag{B.11}$$

From Eq. B.5, constant c_2 is determined (constant c_2 is an average of probability that reaction R_2 occurs when a certain pair of S_1 collides within minimal timestep). Here, combination number of S_1 molecules in V is $X_1(X_1 - 1)/2!$.

$$k_2 = \frac{Vc_2\langle\frac{X_1(X_1-1)}{2!}\rangle}{\langle X_1\rangle\langle X_1\rangle} = \frac{Vc_2}{2} \tag{B.12}$$

B.3.4 Calculating variability of biochemical system over time with stochastic method

Now we are going to track the status of biochemical system with N types of molecules cross-reacting with M types of reactions proceeding over time. This can be obtained from Eq. B.6 which gives basic hypothesis.

(1) Master equation

Time course is obtained by setting up and solving a master equation for the biochemical system. As previously mentioned, it is difficult to mathematically solve master equations, so we execute an equivalent process as stated below.

Most important factor in master equation is a grand probability function. The function is defined as Eq. B.13 for biochemical systems:

$$\begin{aligned} P(X_1, X_2, \cdots, X_N; t) \equiv & \text{Population of } S_1 \text{ is } X_1 \\ & \text{and population of } S_2 \text{ is } X_2 \cdots, \\ & \text{and population of } S_N \text{ is } X_N \text{ in volume } V \text{ at time } t \end{aligned} \tag{B.13}$$

Eq. B.13 describes a stochastic state of a biochemical system at time t. For example, moment (or k-power mean) of P is given as Eq. B.14.

$$\begin{aligned} X_i^{(k)}(t) \equiv \sum_{X_1=0}^{\infty} \cdots \sum_{X_N=0}^{\infty} X_i^k P(X_1, \cdots, X_N; t) \\ (i = 1, \cdots, N; k = 0, 1, 2, \cdots) \end{aligned} \tag{B.14}$$

Mean of power of k for P for X_i corresponds to "Average of k-power for X_i in volume V at time t". The term "average" above indicates an average of X_i obtained by repeating vast amount of simulation between time 0 to t with same initial value X_ν and average reaction occurrence probability defined in Eq. B.6.

Population X_i differs per simulation, but the average k-power mean converges to $X_i^{(k)}$ as simulation is repeated. First and second moment ($k = 1, 2$) is especially important. $X_i^{(k)}$ when $k = 1$ represents the average of population of S_i in V at time t, and the value when $k = 2$ obtained by Eq. B.15 represents the variance of population mean.

$$\begin{aligned} X_i^{(k)}(t) \equiv \sum_{X_1=0}^{\infty} \cdots \sum_{X_N=0}^{\infty} X_i^k P(X_1, \cdots, X_N; t) \\ (i = 1, \cdots, N; k = 0, 1, 2, \cdots) \end{aligned} \tag{B.15}$$

$X_i(t)$ in Eq. B.1 that computes reaction rate with analytical model approximates the first moment, but equality is not necessarily true.

Master equation is a function which obtained time variance with parameters $P(X_1, \cdots, X_N; t)$. There are $1 + M$ ways to realize status $(X_1, \cdots, X_N)$ at time $t + dt$, and $P(X_1, \cdots, X_N; t)$ is a sum those probabilities. P is obtained by sum and multiple of probabilities obtained by Eq. B.6.

$$P(X_1, \cdots, X_N; t + dt) = P(X_1, \cdots, X_N; t)\left[1 - \sum_{\mu=1}^{M} a_\mu dt\right] + \sum_{\mu=1}^{M} B_\mu dt \tag{B.16}$$

Now, let's define a parameter a_μ with

$$\begin{aligned} a_\mu dt \quad &\equiv \quad c_\mu dt \times \{\text{A combination populations that reacts in } R_\mu \\ &\qquad \text{when the status of the model is } (X_1, \cdots, X_X)\} \\ &= \quad \text{Probability that } R_\mu \text{ occurs at time } (t, t + dt) \\ &\qquad \text{when the status of the model is } (X_1, \cdots, X_N) \text{ at time } t \end{aligned} \tag{B.17}$$

The first item of Eq. B.16 is a probability that the status of the model is $(X_1, \cdots, X_N)$ at time t and remains with the same state at time $t + dt$. $B_\mu dt$ in the second item is a probability that the status is $(X_1, \cdots, X_N)$ at time t and R_μ occurs at time $t + dt$.

The master equation derives from Eq. B.16.

$$\frac{\partial}{\partial t} P(X_1, \cdots, X_N; t) = \sum_{\mu=1}^{M} \Big[B_\mu - a_\mu(X_1, \cdots, X_N; t)\Big] \tag{B.18}$$

Except for special cases, description of master equation is simple but is difficult to solve. In addition, unlike reaction rate equations, it is difficult to analytically solve with computers due to the quantity and characteristics of independent variables. Unless all R_μs are simple monomolecular equations, equations that derives moments include moments with higher degree, and it takes infinitely long time to solve time change of moment for certain X_i or $\Delta_i(t)$. Thus, master equation can rigorously and simply describe a model, but it is unsuitable to solve with analytical method. Consequently, we use a method that yields equivalent solution with master equation for stochastic simulation.

B.3.5 Reaction probability density function

This section describes a method to obtain variability of biochemical models over time using stochastic method.

Assuming a system whose status is $(X_1, \cdots, X_N)$ at time t, we need to clarify what reaction occurs at certain time. Thus, we introduce a function $P(\tau, \mu)$ defined by Eq. B.19.

$$\begin{aligned} P(\tau, \mu) \quad &\equiv \quad \text{Probability that } R_\mu \text{ occurs within minimal timestep } (t + \tau, t + \tau + d\tau) \\ &\qquad \text{in volume } V \text{ when status of the model is } (X_1, \cdots, X_N) \end{aligned} \tag{B.19}$$

Function $P(\tau, \mu)$ is called a reaction probability density function. τ represents a time when next reaction occurs, and μ is a type of the reaction.

Now we are going to obtain a function h in reaction R_μ, which is defined in Eq. B.20.

$$\begin{aligned} h_\mu \quad &\equiv \quad \text{Combination populations involved to reaction} \\ &\qquad R_\mu \text{ when status of the model is} (X_1, \cdots, X_N) \qquad (\mu = 1, \cdots, M) \end{aligned} \tag{B.20}$$

If the shape of reaction R_μ is $S_1 + S_2 \rightarrow S$, $h_\mu = X_1 X_2$ is approved. In case when $2S_1 \rightarrow S$, $h_\mu = \frac{1}{2} X_1(X_1 - 1)$ is approved. In general, h_μ is a complex function of variables $X_1, \cdots, X_N$.

Let's assume $P_0(t)$, which is a probability that reactions do not occur within time $(t, t+dt)$ when status of the model is $(X_1, \cdots, X_N)$. If a_μ is a probability that R_μ occurs within time $(t+\tau, t+\tau+d\tau)$, a_μ is described with Eq. B.21.

$$a_\mu d\tau \equiv h_\mu c_\mu d\tau \tag{B.21}$$

Hence, $P(\tau, \mu)$ is yielded as:

$$P(\tau, \mu) d\tau = P_0(\tau) \cdot a_\mu d\tau \tag{B.22a}$$

Next, $P_0(\tau)$ is yielded when $[1 - \sum_\nu a_\nu d\tau']$ is a probability that reactions do not occur at time $d\tau'$ when the status is $(X_1, \cdots, X_N)$:

$$P_0(\tau' + d\tau') = P_0(\tau') \cdot \left[1 - \sum_{\nu=1}^{M} a_\nu d\tau'\right] \tag{B.22b}$$

$$P_0(\tau) = \exp\left[-\sum_{\nu=1}^{M} a_\nu d\tau'\right] \tag{B.22c}$$

Reaction probability density function $P(\tau, \mu)$ is obtained by assigning Eq. B.22c to Eq. B.22a.

$$P(\tau, \mu) = \begin{cases} a_\mu \exp(-a_0 \tau) & \text{if } 0 \leqslant \tau < \infty \text{ and } \mu = 1, \cdots, M \\ 0 & \text{otherwise} \end{cases} \tag{B.23}$$

B.4 Direct Method

This section describes the time and type of reactions that occurs in a biochemical system. The method described here is called "Direct Method", which is statistically-equivalent function with First Reaction Method and Next Reaction Method.

The algorithm computes τ, μ by generating a random number that distributes based on reaction probability density function $P(\tau, \mu)$ based on uniform random numbers.

$$\tau = \frac{1}{a_0} \ln\left(\frac{1}{r_1}\right) \tag{B.24}$$

$$\sum_{\nu=1}^{\mu-1} a_\nu < r_2 a_0 \leqslant \sum_{\nu=1}^{\mu} a_\nu \qquad (\mu : \text{Integer}) \tag{B.25}$$

With the two equations above, we obtain integer random numbers μ that holds $P_2(\mu) = a_\mu / a_0$ and a random number τ that follows a probability density function $P_1(\tau) = a_0 \exp(-a_0 \tau)$ that is used in $P_1(\tau) \cdot P_2(\mu) = P(\tau, \mu)$.

Simulation proceeds by following three steps (Fig. B.4).

Step 0. Initialization

Memories for M types of reaction constants $c_1, \cdots, c_M$ and initial population $X_1, \cdots X_N$ for N types of molecules are allocated. Next, t and reaction counter n are set to 0, and random number generator are initialized.

Step 1.

Value $a_1 = h_1 c_1, \cdots, a_M = h_M c_M$ is computed for all Ms based on the population of molecules, and the sum of a_ν is assigned to a_0.

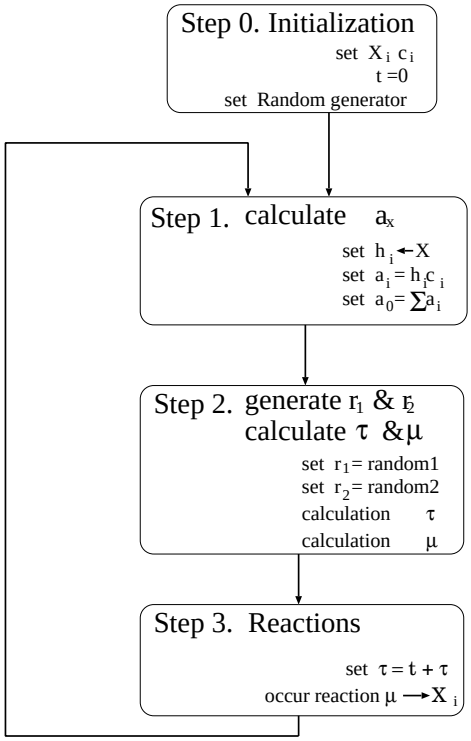


Fig. B.4: Computation flow of Direct Method

Step 2.

r_1 and r_2, two uniform random numbers between $(0, 1)$, are generated, and τ and μ are computed.

Step 3.

t is incremented according to the value of τ. Next, population of molecules affected by occurrence of R_μ is adjusted. For example, if R_μ is a reaction in Eq. B.5, X_2 is incremented by 1, and X_1 is decremented by 1. Then reaction counter n is incremented before going back to Step 1.

Value of $(X_1, \cdots, X_N, t)$ is read out within Step 1-3 or between arbitrary interval of t or n. Computation ends when t or n reaches certain value, or when a_0 becomes 0.

B.5 Advantages and limits of stochastic method

This stochastic simulation algorithm allows a rigorous simulation for biochemical system that is defined based on a basic hypothesis (Eq. B.6). Also, this SSA does not simply repeat the computation per minimal timestep Δt like numerical methods for conventional ordinary differential equations, but calculates a time until the next reaction occurs. Thus, it is effective for models whose population increases or decreases in a very short time interval. Another advantage is small memory space requirement when this this simulation algorithm is executed on a computer system. For example, in case of simulating a reaction with N types of molecules and M types of molecules, the required memory space is N word for storing population of N types of molecules, M words for storing c_ν, and $M + 1$ words for a_ν. This is a large benefit for implementing SSA on an FPGA.

Another benefit is that simulation process and results can be easily read out. Thus, it is facile to

obtain average, variance and correlation of molecules because their population is obtained numerically per simulation cycle.

Computation time of SSA is linear to the number of reactions. Thus, simulation time should be controlled by specifying the number of reactions, types of molecules and their population. The simulation can be conducted when the system is well-stirred, and simulation system size should be enlarged if there is a bias of populations in simulation space. In order to obtain sure results, we must use a "reliable" random number generator, and it is very difficult to testify its dependability. Moreover, multiple simulation trials are required to obtain statistically certain simulation results. Thus, there is a trade-off between reliability of the result (or number of simulation) and execution time.

B.6 Lotka system

Gillespie picked up four types of biochemical systems to apply stochastic simulation algorithm: noninvertible isomerization, Lotka system, Blusselator system and Oregonator system. Noninvertible isomerization is a reaction that exhibits noninvertible mutation, and Gillespie testified the model by stochastic simulation. Other three biochemical system is a more complex model. Noninvertible isomerization is a basic model that is a part of Lotka system. (Eq. B.26c corresponds to the noninvertible isomerization).

Lotka system has been studied since 1920 when Lotka found a following autocatalyzed reaction:

$$\bar{X_1} + X_2 \xrightarrow{c_1} 2X_2 \tag{B.26a}$$

$$X_2 + X_3 \xrightarrow{c_2} 2X_3 \tag{B.26b}$$

$$X_3 \xrightarrow{c_3} Z \tag{B.26c}$$

This model is based on predator-prey model studied by Volterra. He described reactions above with following ordinary differential equations, and studied a method to apply reaction rate equations to the model.

$$\frac{dX_1}{dt} = c_1 X_1 X_2 - c_2 X_2 X_3 \tag{B.27a}$$

$$\frac{dX_3}{dt} = c_2 X_2 X_3 - c_3 X_3 \tag{B.27b}$$

Reaction Eq. B.26b describes a behavior that predators X_3 consumes prey X_2 to reproduce themselves. Reaction Eq. B.26a is a phenomenon that X_2 consumes X_1 for their reproduction. Here, X_1 is unilaterally consumed and do not increase. Isomerization reaction is Eq. B.26c, which describes the death of X_3.

It is obvious that steady state is kept when a condition $dX_2/dt = dX_3/dt = 0$ are met.

Lotka system is one of a simple example of chemical reactions, but it can be easily understood by applying it to actual phenomenon in our ecosystem. In this work, we added another condition that the rate of the death of X_2 and population of X_1 do not change during the simulation in addition to reaction Eq. B.26.