# A multi-algorithm, multi-timescale method for cell simulation

**Kouichi Takahashi [*] (shafi@e-cell.org)**
**Kazunari Kaizu (t00220kk@sfc.keio.ac.jp)**
**Bin Hu (hubin@sfc.keio.ac.jp)**
**Masaru Tomita (mt@sfc.keio.ac.jp)**
**Institute for Advanced Biosciences, Keio University, Fujisawa, Kanagawa, 252-8520, Japan.**

[*] To whom correspondence should be addressed.

# Abstract

**Motivation** Many important problems in cell biology require the dense nonlinear interactions between functional modules to be considered. The importance of computer simulation in understanding cellular processes is now widely accepted, and a variety of simulation algorithms useful for studying certain subsystems have been designed. Many of these are already widely used, and a large number of models constructed on these existing formalisms are available. A significant computational challenge is how we can integrate such sub-cellular models running on different types of algorithms to construct higher order models.

**Results** A modular, object-oriented simulation meta-algorithm based on a discrete-event scheduler and Hermite polynomial interpolation has been developed and implemented. It is shown that this new method can efficiently handle many components driven by different algorithms and different timescales. The utility of this simulation framework is further demonstrated with a "composite" heat-shock response model that combines the Gillespie-Gibson stochastic algorithm and deterministic differential equations. Dramatic improvements in performance were obtained without significant accuracy drawbacks. A multi-timescale demonstration of coupled harmonic oscillators is also shown.

**Availability** An implementation of the method is available as part of E-Cell Simulation Environment Version 3 downloadable from http://www.e-cell.org/software. Benchmark models are included in the package, and also available upon request.

**Supplement** Complete lists of reactions and parameters of the heat-shock model, and more results are available at http://www.e-cell.org/bioinfo/takahashi03-1-supp.pdf.

**Contact** shafi@e-cell.org

# 1 Introduction

Computational cell biology is a rapidly growing simulation-oriented research field that has been greatly stimulated by the array of high throughput methods developed in recent years. In a previous publication, we have argued that cell simulation poses many significant computational challenges that are distinct from those encountered in problems of other disciplines such as molecular dynamics, or computational physics, and even conventional biochemical simulations (Takahashi *et al.*, 2002). A vast array of molecular processes occur simultaneously in the cell. The involved physical and chemical processes include molecular diffusion, molecular binding, enzymatic catalysis and higher order phenomena such as cytoplasmic streaming, complex macromolecular interactions (such as the dynamics of RNA polymerase on the DNA molecule), and global structural changes caused by cell division and cell differentiation.

Many different kinds of simulation algorithms have been proposed and are currently being used for simulation of cellular processes. By far the most widely accepted scheme of describing continuous dynamical systems is representation by differential equations. Apart from standard generic solver algorithms (Gear, 1971), specialized methods are often used for S-System and Generalized Mass-Action (GMA) canonical forms of power-law differential systems in simulations of metabolic and gene regulations (Irvine and Savageau, 1990). To complement a lack of quantitative kinetic data, an attempt to marry dynamic rate-equation based continuous models and static flux balance from stoichiometry is ongoing (Yugi *et al.*, 2002). In addition to those continuous representations, discrete formulation is often used. Stochastic discrete event simulation of coupled chemical reactions was pioneered by Gillespie and others (Gillespie, 1976), and improved by Gibson (Gibson and Bruck, 2000) to work more efficiently. StochSim is a discrete-time stochastic simulation algorithm originally devised as a means of investigating the bacterial chemotaxis signaling pathway (Morton-Firth and Bray, 1998). Simulation methods based on cellular automaton have many applications from reactive gas physics to sociology, and have been found to be very useful for biological simulations (for example, see Weimar, 2002). Brownian dynamics has proven to be an useful framework of interactions between ligands and binding sites on receptors, enzymes, and transporters (Bartol *et al.*, 1996).

Different algorithms have different strengths, and are often suited to different spatial and temporal scales. When constructing models that span multiple scales, one might choose between two principal approaches: the first is the combined algorithm approach which aims to bind strengths of existing simulation algorithms to produce a unified simulation algorithm of wide utility. The other option is an embedded algorithm approach, in which existing algorithms are implemented as pluggable modules wrapped to meet a common interface specification of a generic framework of time advance and inter-module communications. In this work, we have explored the latter approach, which requires the development of a meta-algorithm that serves as a framework for integrating heterogeneous units of a composite model. A virtue of this style of scheme integration is that there is orthogonality between subsystems, that is to say a degree of modularity can be assumed for each subsystem simulated using a particular algorithm. If this requirement is satisfied, one can expect established and well-studied algorithms, and the numerous existing models that exploit them, to be useful when combined in a "plug-in" fashion to let them take part in composite simulations that utilize the multiple algorithms.

# 2 Algorithm

As noted above, the novel approach we propose depends on the design of a meta-algorithm, which we describe here in three parts: (1) data structure, which outlines the organization of information required for model definition and execution, (2) driver algorithm, which describes how interactions between sub-modules are handled, and (3) integration algorithm, which explains the procedure by which state variables are updated.

In this article, a vector is denoted with a small bold letter (*e.g.* $\mathbf{x}$). Capital letters (*e.g.* $X$) are used for sets. Class names are capitalized (*e.g.* Stepper). Small and capital letters are used for scalars and objects.

## 2.1 Data Structure

In this meta-algorithm, a Model consists of a vector of state variables and a set of Steppers. Each Stepper consists of a set of Processes, each of which changes values of the state variables according to its transition function through *variable references*. The meta-algorithm knows *Stepper dependencies* by tracking the variable references.

*Model* is a class:

$$M = \langle \mathbf{x}, S \rangle \tag{1}$$

where $\mathbf{x} = \{x_0, x_1, \ldots, x_n\}$ (typically $\in \mathbb{R}^n$) is a vector of state variables, and $S$ is a set of *Steppers*.

A Stepper is a class which represents a computational sub-unit of the model. The Stepper class is defined as:

$$S_i = \langle P_i, \tau_i, \Delta\tau_i, \Lambda_i, I_i \rangle \tag{2}$$

where $P_i = \{P_{i,0}, P_{i,1}, \ldots P_{i,m}\}$ is a set of *Processes*, $\tau_i$ is the current local time, $\Delta\tau_i$ is the current time step interval, $\Lambda_i$ is the step method, and $I_i$ is the interruption method, of the $i$th Stepper. The interruption method $I_i$ is called by other (interrupting) Steppers to let this (interrupted) Stepper know that this Stepper may need recalculation because of the changes of the variables caused by the interrupting Stepper. See Stepper dependency below and the next Driver Algorithm section. Ranges of $\tau$ and $\Delta\tau$ are the same: $(\mathbb{R}^{+,0} \cup \infty)$.

Definition of the Process class is given as follows:

$$P_{i,j} = \langle F_{i,j}, R_{i,j} \rangle \tag{3}$$

where $F_{i,j}$ is a transition function, and $R_{i,j}$ is a set of variable references such that $\{x_\rho\}_{\rho \in R} \subseteq \mathbf{x}$, of the $j$th Process instance which belongs to Stepper $S_i$. $R_{i,j}$ consists, in turn, of two subsets. $\check{R}_{i,j} \subseteq R_{i,j}$ is the accessor variable reference subset, and $\hat{R}_{i,j} \subseteq R_{i,j}$ is the subset of mutator variable references, such that $\check{R} \cup \hat{R} = R$. Read accesses to variables by a Process is restricted by its accessor variable references, and it can only change values of variables in its mutator variable reference subset. This information is used to calculate the Stepper dependency below. For brevity we denote $R_i$ to mean $\bigcup_j R_{i,j}$ in the following sections.

$F$ has a general form of:

$$F : (\check{R}^*, \Delta\tau, \tau) \rightarrow (\hat{R}^*, \delta\hat{R}) \tag{4}$$

where $\delta\hat{R}$ is a set of contributions to derivatives of the mutator variables of $F$. $(^*)$ means that each reference element is de-referenced to get variable instances; thus $R^* \subseteq \mathbf{x}$.

We refer to a specialized form of $F$, $F : (\check{R}^*, \Delta\tau, \tau) \rightarrow (\delta\hat{R})$ as a *continuous transition function* and $F : (\check{R}^*, \Delta\tau, \tau) \rightarrow (\hat{R}^*)$ as a *discrete transition function*. For instance, differential equations generally have an even simpler form of the continuous transition function, $F : (\check{R}^*, \tau) \rightarrow (\delta\hat{R})$. Difference equations can be viewed as a type of discrete transition function. Discrete event processes, which are constrained to discrete-valued changes in state variables at scheduled time points, are formulated as: $F : (\check{R}^*, \tau) \rightarrow (\hat{R}^*)$.

We call a Stepper made up of Processes with continuous transition functions a *continuous Stepper*, and a *discrete Stepper* if it has Processes with discrete transition functions.

In addition to the Model data structure presented above, the meta-algorithm requires some additional data structures to execute a simulation.

The current global time, $T$, during a simulation is defined as: $T \equiv \min_i(\tau_i + \Delta\tau_i)$.

An *updated-time vector* $\mathbf{t} = \{t_0, t_1, \ldots, t_n\} \in (\mathbb{R}^{+,0} \cup \infty)^n$ is used to store the last updated times of all the variables.

A binary relation, which we call the Stepper Dependency, $D$, is defined on an ordered pair of Steppers $S_a$ and $S_b$ as:

$$D = \{(S_a, S_b) | (S_a, S_b) \in S \times S \wedge S_a \neq S_b \wedge \check{R}_a \cap \hat{R}_b \neq \emptyset\} \tag{5}$$

The interpretation is very simple; if a Stepper $S_b$ can change a value of at least one variable that the other Stepper $S_a$ can read, $S_a$ is said to be dependent on $S_b$, that is, $(S_a, S_b) \in D$ (or equivalently, $S_a D S_b$). This relation is not transitive (*i.e.* $S_a D S_b \wedge S_b D S_c$ does not imply $S_a D S_c$), and not symmetric (*i.e.* $S_a D S_b$ does not imply $S_b D S_a$).

## 2.2 Driver Algorithm

Time is advanced during the simulation in an iterative fashion. Each iteration of the simulation consists of the following nine procedures.

1. Initialize $\mathbf{t}$ and all the $\tau$s to zero.
2. Compute the Stepper Dependency $D$ defined by Equation 5 for all combinations of Steppers.
3. From $S$, pick a Stepper $S_i$ which has the minimum scheduled time $\tau_i + \Delta\tau_i$ over the set.
4. Update the local time of $S_i$: $\tau_i \leftarrow \tau_i + \Delta\tau_i$.
5. *integrate*: Update values of variables $R_i^*$ according to the integration algorithm given below.

6. *step*: Allow the Stepper $S_i$ to execute a "step" by calling its implemented "step method" $\Lambda_i$. $\Lambda_i$ may call transition functions $F_{i,j}$ of Processes $P_{i,j}$ in the set $P_i$ according to its implemented algorithm. $\Lambda_i$ may also update the step size $\Delta\tau_i$, the timescale parameter $\theta_i$, and any other implementation-specific parameters of the Stepper $S_i$.
7. *dispatchInterruptions*: In this procedure the Stepper $S_i$ notifies the change of the variables to other Steppers. $S_i$ calls interruption methods $I_j$ of all Steppers $S_j$ that depend on $S_i$, *i.e.*, $\{S_j | S_j \in S \wedge (S_j, S_i) \in D\}$. The interruption method $I_j$ may change $\Delta\tau_j$, $\theta_j$, and any other implementation-specific parameters of the Stepper $S_j$.
8. *log*: For post-simulation data processing, record values of variables related to this Stepper, $R_i^*$, if necessary.
9. End if the termination condition given by the user (such as the value of $T$ and the number of iterations) is met. Otherwise go to 3.

The time is advanced and the values of the state variables are updated in three actions (procedures 4, 5, and 6). Procedure 4 updates the local time $\tau_i$ of the Stepper $S_i$ and the global time $T$. In procedure 5, the variables are updated by continuous Steppers (See Integration Algorithm section). In procedure 6, $\Lambda_i$ of a discrete Stepper may call its transition functions to change the values of variables $R_i^*$. A continuous Stepper usually does not change the values of variables in this procedure, but uses $\Lambda_i$ to recalculate input parameters to its interpolants from $\delta\hat{R}_i$ (See Integration Algorithm and Implementation sections). $\Delta\tau$ is also recalculated in this procedure, and is used in procedure 4 of the next step of the Stepper to advance the time. In procedures 6 and 7, Steppers set an ancillary parameter $\theta$ ( $\in \mathbb{R}^{+,0} \cup \infty$ ) which indicates the time scale of change of $\hat{R}_i^*$ and $\delta\hat{R}_i$. Normally $\theta > 0$ for continuous Steppers. If $S_i$ is a discrete Stepper, usually $\theta = 0$. In procedure 7, Steppers which depend on $S_i$ may use this value as a clue to optimize computation by ignoring some of the interruptions from this Stepper. See the Implementation section for example usage and calculation of $\theta$. If more than one Stepper has the same scheduled time in procedure 3, the one with the highest 'priority' value (defined as part of the model) is chosen.

### 2.3 Integration Algorithm

In the *integrate* phase of the meta-algorithm (procedure 5 above), interpolation is used to recompute the set of variables $R_i^*$.

In the simplest case, where all the continuous Steppers in the model implement first-order numerical integration algorithms, the procedure has a form of Euler's method with a second order error term. For each $j \in R_i$,

$$x_j(\tau_i) = x_j(t_j) + \Delta t_j \sum_{k \in \tilde{S}} x'_{j,k} + O(\Delta t_j{}^2), \quad \Delta t_j = \tau_i - t_j \tag{6}$$

$$x'_{j,k} = \sum_{l \in P_k} \delta\rho_{k,l}, \quad \rho_{k,l} \in R_{k,l} \tag{7}$$

where $\tilde{S} \subseteq S$ is the set of continuous Steppers, and $t_j$ is the last updated time of $x_j$. Here $\delta\rho_{k,l}$ is a contribution to the velocity of change of $x_j$ given by the $l$th Process of the $k$th *Stepper*, and $t_j$ is the last updated time of $x_j$.

Although mathematically concise, this scheme spoils the benefits of higher order algorithms. No matter how high the order of the internal computational procedure being used, the outcome will be integrated as a linear sum of the most recently computed first order derivatives, resulting in a first order global precision. First-order algorithms, however, only occasionally satisfy practical requirements, in terms of accuracy, efficiency and stability, for numerical simulations of continuous systems. Therefore this procedure should be extended to have a higher order error term.

Replacing the first order derivative term of Equation 6 by a difference of higher order interpolants we get:

$$x_j(\tau_i) = x_j(t_j) + \sum_{k \in \tilde{S}} \Phi_{k,j}(\tau_i, \Delta t_j) + O(\Delta t_j{}^s) \tag{8}$$

where $\Phi_{k,j}(\tau_i, \Delta t_j)$ is an interpolant difference given by the $k$th Stepper for $x_j$. The order of the global error term is $s = \min_{k \in \tilde{S}} d_k$. Here $d_k$ is the order of the $k$th Stepper.

The interpolant difference $\Phi_{k,j}(\tau_i, \Delta t_j)$ is defined as a difference between values of the interpolant $\iota_{k,j}$ at time points $\tau_i$ and $\tau_i - \Delta t_j$.

$$\Phi_{k,j}(\tau_i, \Delta t_j) = \iota_{k,j}(\tau_i) - \iota_{k,j}(\tau_i - \Delta t_j) \tag{9}$$

The relationship between $\tau_i$ and $t_j$ is depicted in Figure 1.

To minimize the total expected error over the whole model, the input parameters to each interpolant should be recomputed upon receiving an interruption (procedure 7 above). This functionality is not guaranteed by the meta-algorithm, and is a requirement in the implementation of specific algorithm modules, some examples of which are described below.

## 3 Implementation

The meta-algorithm has been implemented in the ISO C++ programming language with extensive use of template-based generic programming techniques (Alexandrescu, 2001) and object-orientation. To optimize the operation of finding the Stepper with the minimum scheduled time in procedure 3 of the driver algorithm, a discrete event scheduler was implemented as a heap-tree-based dynamic priority queue for which the rescheduling cost is $O(\log |S|)$. The implementation has an XML parser for model description files, C++ and Python language APIs for user scripting and frontend module development. Simulation algorithms can be developed in the C++ language as dynamically loadable plug-in modules by inheriting base classes provided by the system.

Three sub-classes of Stepper, *DiscreteEventStepper*, *DiscreteTimeStepper* and *DifferentialStepper*, are provided according to Zeigler's classification of simulation algorithms (Zeigler *et al.*, 2000). Each concrete implementation of an algorithm is a subclass of one of these base classes.

At least the step method $\Lambda$, the interruption method $I$, and a function to determine the timescale parameter $\theta$ must be defined in each implementation of a Stepper.

## 3.1 Differential equation modules

As subclasses of the DifferentialStepper, two general-purpose ordinary differential equation (ODE) solvers have been implemented. Many variants of the Runge-Kutta algorithm exist, and the general form of the algorithm is given as follows:

$$\begin{aligned} k_j &= f\left(x_n + c_j h, y_n + h\textstyle\sum_{i=1}^{s} a_{j,i} k_i\right), \; j = 1 \dots s \\ y_{n+1} &= y_n + h\textstyle\sum_{j=1}^{s} b_j k_j \end{aligned} \tag{10}$$

where $h$ is the step size, $s$ is the number of right hand side (RHS) evaluations required for a single step. Specification of the matrix of coefficients $a$, $b$, and $c$ determine each specific variation of the algorithm.

Two instances of embedded explicit Runge-Kutta algorithms with dense output capabilities have been implemented: second-order Fehlberg with a third-order error estimation (Fehlberg 2(3)) (Fehlberg, 1969), and fourth-order Dormand-Prince with a fifth-order error term (Dormand-Prince 5(4)7M) (Dormand and Prince, 1980).

For some integration algorithms including these two, it is possible to derive interpolants by using Hermite polynomial interpolation, the general form of which is:

$$g(x) = \textstyle\sum_i \sum_j \alpha_{i,j}(x) f_i^{(j)}, \; \left[\frac{d^j}{dx^j}\alpha_{i,j}(x)\right]_{x=x_k} = \delta_{i,k} \tag{11}$$

where $x_k (k = 1, 2, \cdots, n)$ is the time point where input is given, $f_i^{(j)}$ is the input function defined at $x_i$ as the $j$th order derivative, and $\delta_{i,k}$ is the Kronecker delta. If $f_i^{(j)}$ defines data points at $m$ different combinations of $i$ and $j$, then it can make an interpolant of order $m-1$.

A $C^0$ interpolant of the second-order Fehlberg algorithm can easily be derived because we have three data points, the initial value, $(x_j(\tau_k))$, the value at $\tau_i + \Delta\tau_i$, $(x_j(\tau_k) + \sum_{i=1}^{3} b_i k_i)$, and the first-order derivative value at the initial point, $(k_1)$.

$$\begin{aligned} \iota_{k,j}(t) &= (1-\sigma^2)\, x_j(\tau_k) \; + \; (-\sigma^2 + \sigma)\Delta\tau_k k_1 \\ &\quad + \sigma^2 \left(x_j(\tau_k) + \Delta\tau_k \textstyle\sum_{i=1}^{3} b_i k_i\right) + O(\Delta t_j{}^3) \end{aligned} \tag{12}$$

$$\sigma = (t - \tau_k)/h, \;\; 0 < \sigma \leq 1 \tag{13}$$

where $\tau_k$ and $\Delta\tau_k$ are the current time and the current step size of this Stepper, respectively. Here $\alpha_{0,0} = (1-\sigma^2)$, $\alpha_{0,1} = (-\sigma^2 + \sigma)$, and $\alpha_{1,1} = \sigma^2$. $\Delta\tau_k$ appears in the second and the third terms of the RHS because of the change of variable from $t$ to $\sigma$.

Shampine (1986) gives a $C^1$ interpolant of the Dormand-Prince 5(4)7M with additional coefficients $b_i^*$:

$$\begin{aligned} \iota_{k,j}(t) &= x_j(\tau_k) + \sigma h k_1 \; + \; 4\sigma^2 h(\sigma - 1)(k_1 - v_{\frac{1}{2}}) \\ &\quad + 2\sigma^2 h(\sigma - \tfrac{1}{2})(v_1 - k_1) \\ &\quad + 2\sigma^2 h(\sigma - \tfrac{1}{2})(\sigma - 1)(k_7 - k_1 + 4v_{\frac{1}{2}} - 4v_1) \\ &\quad + O(\Delta t_j{}^5), \\ v_{\frac{1}{2}} &= \textstyle\sum_{i=1}^{7} b_i^* k_i, \; v_1 = \sum_{i=1}^{7} b_i k_i \end{aligned} \tag{14}$$

The number of RHS evaluations required in each step of the simulation is three for Fehlberg 2(3).

Interestingly, $k_7^{[n]} = k_1^{[n+1]}$ for the Dormand-Prince 5(4)7M algorithm if there is no interruption in a time interval $(\tau_n, \tau_{n+1}]$. Thus the number of RHS evaluations required by the algorithm is seven for starting up, and six or seven during the simulation.

The step method $\Lambda$ for these ODE modules are implemented with a standard adaptive step sizing mechanism (Press *et al.*, 2002). In addition, time step sizes are constrained by the following condition to ensure that the Stepper does not change variable values too much at once in a single step:

$$\forall j \in \hat{R}_i : x_j \epsilon_{rel} + \epsilon_{abs} \geq \sum_{l \in P_i} \delta\rho_{i,l} \cdot \Delta\tau_i \tag{15}$$

where $\epsilon_{abs}$ and $\epsilon_{rel}$ (usually $\lesssim 0.1$) are absolute and relative constraint parameters of changes of the variables in a step, and $\rho_{i,l}$ is a variable reference of the $l$th Process in the $i$th Stepper pointing to $x_j$. The constraint parameters $\epsilon_{abs}$ and $\epsilon_{rel}$ should be provided according to external conditions given to this Stepper, such as the total number of continuous Steppers and their implemented algorithms.

In this implementation $\theta_i$ has the same value as $\Delta\tau_i$.

The interruption method $I_i$ for the $i$th ODE solver is defined as follows:

1. If the timescale of the interrupter is smaller than the step size of this Stepper, $\theta_k < \Delta\tau_i$, then use $\theta_k$ as the initial guess of the next step size. Otherwise, just return ignoring this interruption.
2. If the next step of the interrupter is before this Stepper, $(\tau_i + \Delta\tau_i > \tau_k + \Delta\tau_k)$, set $\Delta\tau_i$ to $T - \tau_i$.

Procedure 2 makes sure that the interrupted Stepper $S_i$ steps at least once between the current and the next steps of the interrupting Stepper $S_k$. This guarantees that the constraint in Equation 15 takes effect on the interrupted Steppers. Procedure 1 is an optimization. If the interrupted steps more frequently than the interrupter's timescale parameter $\theta_k$, the constraint is assumed to be satisfied without the interruption. By using $\theta_k$ as the next step size of the interrupted Stepper, this optimization more likely happens again in the next step.

### 3.2 Gillespie-Gibson module

A standard Gillespie-Gibson algorithm, or the Next Reaction Method, has been implemented as a subclass of the DiscreteEventStepper in a fully object-oriented fashion. The probability density function for the next reaction to occur at time $\tau$, $P(\tau)$, and the probability that this reaction is of type $\mu$, $Pr(\mu)$, are given by the following two equations.

$$P(\tau) = \left(\sum_j a_j\right) \exp\left(-\tau \sum_j a_j\right) \tag{16}$$

$$Pr(\mu) = a_\mu / \sum_j a_j \tag{17}$$

where $a_j$ is the propensity function of the $j$th reaction. The propensity function $a_j$ is defined as $k \prod X$ (if the reaction is in either form of $X_1 \rightarrow X_2$ or $X_1 + X_2 \rightarrow X_3$), or as $kX_1(X_1 - 1)/2$ (if the reaction is $2X_1 \rightarrow X_2$), where $k$ is the rate constant and $X_i$ is the number of copies the molecular species involved in the reaction. In this implementation, we only consider uni- and bi-molecular reactions, and if $X_i$ is not an integer, a floor is taken to calculate the propensity.

The interruption method $I$ has been implemented to enable synchronization with other modules. The procedure is: (1) Recalculate the propensities of the reactions, (2) Redetermine the time of the next reaction $\tau_i + \Delta\tau_i$ and the next reaction $\mu$ according to the main algorithm, and (3) Reschedule the Stepper to time $\tau_i + \Delta\tau_i$ on the scheduler.

The timescale $\theta_i$ is determined as:

$$\theta_i = \xi \cdot \Delta\tau_i \min_{\rho \in R_{i,\mu}} \frac{\rho^*}{|\nu_\rho|} \tag{18}$$

where $\xi$ is a tolerance parameter, $i$ is an index of the Stepper, and $\nu_\rho$ is the stoichiometric coefficient for the variable reference $\rho$ defined as a part of the model. $\xi$ takes a small number such as 0.1, or 0. $\xi \neq 0$ directs it to pretend to be a continuous component. This may make the simulation more efficient if this Stepper takes smaller step sizes than other parts of the model. For instance, $\xi = 0.1$ lets $\theta$ be the expected time for the reactant of the current Process with the smallest population to change its value by 10 %. If precision precedes efficiency, set $\xi = 0$.

The Mersenne-Twister algorithm (Matsumoto and Nishimura, 1998) has been employed as a pseudo-random number generator.

# 4 Results

Two relatively simple models have been constructed to examine the performance of the meta-algorithm. The first is a model of the heat-shock response, and has three variations, deterministic (ODE), stochastic (Gillespie), and composite (ODE/Gillespie). This example is to show multi-algorithm and multi-timescale capabilities of the algorithm. The second example is a simple cascade of coupled harmonic oscillators with gradually changing timescales that span six orders of magnitude.

All the timings and results given in this work have been taken on a PC running RedHat Linux 9 operating system with a dual Hyper-Threading-enabled Intel Xeon 2.8GHz CPU and 1GB of RAM. This implementation is a single-thread application.

### 4.1 A multi-algorithm heat-shock model

When cells are exposed to high temperature, the synthesis of a small number of proteins known as heat-shock proteins becomes selective and rapid (Gross, 1999). This process is called the heat-shock response. $\sigma^{32}$, a variation of the $\sigma$ subunit of RNA polymerase, has been implicated as the global regulator for this system (Grossman, 1987). DnaJ is a molecular chaperone that plays an important role in regulating the activity and stability of $\sigma^{32}$ (Gamer *et. al.*, 1996). Many molecular species involved in this process are present in small numbers, most of which relate to gene expression processes, and require stochastic simulation. In contrast, protein folding involves large quantities of molecules, making stochastic simulation computationally challenging.

To evaluate the performance of a composite simulation scheme, we set up a pure stochastic model, a pure ODE model and a stochastic/ODE composite model, based on Srivastava's stochastic petri net model of the *E.coli* heat-shock (Srivastava *et al.*, 2001). Protein folding/unfolding processes have been added to the model, and modeled as differential equations in the ODE and composite models, and using the Gillespie algorithm in the stochastic model (Figure 2). In the composite model, the stochastic and ODE parts are coupled bidirectionally. Complete lists of reactions, parameters and initial values are available in **Supplement**. All parameter settings and initial values of the three variations of the model are identical.

We ran each model ten times for 100 seconds and traced the quantities of $\sigma^{32}$ and DnaJ to benchmark the performance. These two species were selected because $\sigma^{32}$ is biologically important, as it controls the expression level of heat shock proteins, and DnaJ is on the boundary of the different algorithms in the composite model. The composite model ran about 2.6 times as fast as the ODE model, and 351 times faster than the stochastic model (Table 1). The difference in mean steady-state levels in these three variations were within 2.7% for $\sigma^{32}$, and 0.0008% for DnaJ.

The standard deviation of $\sigma^{32}$ in the composite run was indistinguishable from that of the stochastic run. In contrast, the trajectory of DnaJ was devoid of stochastic fluctuations in the composite model (Table 1, Figure 4). The total number of protein molecules in this example model is of order $10^6$ – $10^7$. Histograms to compare $\sigma^{32}$ results in composite and stochastic runs are available in **Supplement**.

## 4.2 A simple model of multi-timescale oscillators

A cascade of twenty harmonic oscillator components $X_0, X_1, \cdots, X_{19}$ have been constructed. Each component $X_n$ is coupled with the next one $X_{n+1}$ with a positive feedforward. The rate constant of $X_n$ is doubled for $X_{n+1}$ (Figure 3), resulting in a total timescale difference in the model of about $\log_{10}(2^{20}) \simeq 6.0$ orders of magnitude. Five sets of the cascade are included in a model.

Benchmarking was conducted on the model with one, two, four, ten, and twenty Steppers (Table 2). As an example, in the two Steppers variant, the components were divided into two groups, $X_0, \cdots, X_9$ and $X_{10}, \cdots, X_{19}$. Relative and absolute differences between results from these variations were smaller than the error tolerance parameter given to the Steppers ($10^{-6}$). The simulation speed improved as the number of Steppers was increased. The twenty Steppers variant achieved almost an order of magnitude better performance than the original.

# 5 Discussion

In the limit of large numbers of reactant molecules, stochastic and deterministic simulations are equivalent (Gillespie, 1977). In contrast, if the system has low copy numbers of species, the deterministic law of mass action breaks down because the steady-state fluctuations in the number of molecules (which is proportional to the square root of the number of molecules) becomes a significant factor in the behavior of the system (Singer, 1953). ODE models isolate the biochemical system into a group of deterministic and continuous reactions, and tacitly ignore fluctuations in the pathway (Rao *et al.*, 2002). With identical parameter settings, the stochastic and deterministic models can produce different results, and stochastic models are generally believed to be more accurate (Marion *et. al.*, 1998; Srivastava *et al.*, 2002).

Despite its advantages, however, Gillespie's scheme of exact stochastic simulation has limited utility due to its computational cost which is proportional to the number of molecules. The composite simulation of the heat-shock demonstration model shows that, if carefully chosen, ODEs can replace some parts of the stochastic model, resulting in a dramatic improvement in performance without jeopardizing the benefits of the original purely stochastic model. A logical extension to this "static" combination of the deterministic and stochastic schemes would be "dynamic" switching between these different simulation methods. This will be further explored in future work.

The composite heat-shock model runs faster than both the ODE and the Gillespie models. The following crude approximation of computational costs of these models withstand some discussion.

$$O_{Gillespie} \propto \left[\sum a\right] \log N, \quad O_{ODE} \propto \gamma \left[\frac{\partial f}{\partial x}\right] N,$$
$$O_{Composite} \propto \left[\sum a\right]_m \log N_m \qquad + \gamma \left[\frac{\partial f}{\partial x}\right]_f N_f$$

where $[\frac{\partial f}{\partial x}]$ is a measure of the degree of stiffness, $N$ is the number of reactions, $[\sum a]$ denote the total propensity, and $\gamma$ is a constant parameter to relate $[\frac{\partial f}{\partial x}]$ and $[\sum a]$. Subscripts $m$ and $f$ indicate the main part and the folding part of the model, respectively. Obviously $N = N_m + N_f$. Computational cost of explicit ODE solvers such as Dormand-Prince 5(4)7M is largely determined by the degree of stiffness, which is dominated by $\frac{\partial f}{\partial x}$ (where $x$ is a dependent variable, and $f$ is the derivative function for $x$). The large (137-fold) performance difference between the ODE and Gillespie implies $\gamma[\frac{\partial f}{\partial x}]N \ll [\sum a] \log N$. In fact, $\Delta\tau$ of the ODE Stepper was about $10^{-3}$, while the step size of the Gillespie-Gibson Stepper was typically in the range of $10^{-6}$ – $10^{-9}$. The composite model is faster than the ODE model, if $[\frac{\partial f}{\partial x}]_f N_f < [\frac{\partial f}{\partial x}]N$ and $[\sum a]_m \log N_m \ll \gamma[\frac{\partial f}{\partial x}]N$. Because the degree of stiffness is dominated by the fastest component of the system, it can be assumed that $[\frac{\partial f}{\partial x}]_f \simeq [\frac{\partial f}{\partial x}]$. Thus the first proposition can be verified by $N_f < N$. The latter proposition is consistent with the $\Delta\tau$ observed in Gillespie simulation of the main model in isolation ($10^{-1}$ – $10^{-3}$). This speed difference, however, is less important than that of the Gillespie and the composite models because the dependency of the ODE solvers to stiffness can partially be overcome by use of an implicit or a semi-implicit solver (Gear, 1971). Now the difference between the composite model and Gillespie model is explained because $[\sum a]_m \ll [\sum a]$ and $\gamma[\frac{\partial f}{\partial x}]_f N_f < \gamma[\frac{\partial f}{\partial x}]N \ll [\sum a] \log N$.

In the composite model, $\sigma^{32}$ behaves identically with the pure stochastic model, and the trajectory of DnaJ almost matches that of the deterministic model. $\sigma^{32}$ is one step away, and DnaJ is on, the boundary between the deterministic and the stochastic parts of the model. In the composite simulation, stochastic fluctuation of DnaJ is absorbed by the fast rates of the deterministic reactions involved in controlling its stady-state value. The primary path from the boundary (DnaJ) to $\sigma^{32}$ is a slow reaction (dissociation of $\sigma^{32}$ / DnaJ complex, $k = 4.4 \times 10^{-4}$) which filters higher frequency fluctuations. Thus $\sigma^{32}$ is insensitive to such rapid fluctuations even in the pure stochastic model, and no difference is observed between the stochastic and composite runs in the first and second moments of the time course. (An analysis of higher moments showed a slight difference in skewness, and no difference in kurtosis. See

**Supplement**.) However, as the behavior of DnaJ exemplifies, the composite simulation must be used with caution if the stochastic effect on species near the boundary is of interest, or is expected to influence the overall behavior of the model.

The Gillespie-Gibson Stepper recalculates putative times of all the reactions upon interruptions by other Steppers. The correctness of these recalculated times in a composite simulation is guaranteed because the interruption procedure itself does not affect the probability density $P(\tau)$. To verify this, consider an interruption at $\tau'$, ($\tau' < \tau$), the probability density of the next reaction time, $\tilde{P}(\tau)$, is then calculated by the following equation.

$$\begin{aligned}\tilde{P}(\tau) &= \left(1 - \int_0^{\tau'} P(t)dt\right) \cdot P(\tau - \tau') \qquad (19)\\ &= \exp\left(-\tau' \textstyle\sum_j a_j\right) \cdot P(\tau - \tau')\\ &= \left(\textstyle\sum_j a_j\right) \exp\left(-\tau \textstyle\sum_j a_j\right)\end{aligned}$$

This is equal to the original function in Equation 16.

The coupled oscillator model demonstrates the efficiency with which multi-timescale phenomenon can be modeled in our novel framework. In this toy model, slower components were unidirectionally connected to faster components. This type of coupling is commonly observed in the cell. For instance, expression of enzymes from genes, which is relatively slow, controls metabolic reactions. Conversely, if there is a feedback from the faster components to the slower ones, which is also common in the cell, frequent synchronizations are required. In this case the performance of this implementation is expected to be the same as, or worse than, conventional synchronous single-algorithm simulators because of the interruption overhead. However, if the slow component is assumed to be insensitive to perturbations on the faster component's timescale, synchronization costs could be reduced. This improvement will also be considered in future work to improve the meta-algorithm.

Time-driven simulation algorithms can be classified into three categories: differential equation solvers, discrete time systems, and discrete event systems. Those formalisms can be combined by or embedded in a "discrete event world view" (Zeigler *et al.*, 2000). Although this work is not a direct derivation of this rigid discrete event theory of modeling and simulation, some fundamental concepts such as the classification of simulation algorithms and the use of an event scheduler have been borrowed from that framework.

# 6 Conclusion

Due to the non-linear nature of the subsystems and the intimate coupling between them, simulation is crucial for cell biology research. In the past, however, it has been the norm to adopt different simulation algorithms for different subsystems of the cell. This had made it difficult to combine the sub-cellular models and in many cases limited the applications of simulation to single-scale, sub-cellular problems.

In this work, we have provided a modular meta-algorithm with a discrete event scheduler that can incorporate any type of time-driven simulation algorithm. It was shown that this meta-algorithm can efficiently drive simulation models with different simulation algorithms with little intrusive modification to the algorithms themselves. Only a few additional methods to handle communications between computational modules are required.

The best algorithm for a specific model is determined by the nature of the target system. Our meta-algorithm provides a solution for situations in which it is necessary to simulate processes concurrently across multiple scales of time, space or concentration (Rao *et al.*, 2002).

# 7 Acknowledgments

The authors would like to thank Tomoya Kitayama and Gabor Bereczki for help in implementation, and Dr. Toshikazu Ebisuzaki for useful advice. Dr. Thomas Shimizu and Chris Pickett kindly helped us by critical readings and editing. This work was supported in part by the Japan Science and Technology Corporation, the Ministry of Agriculture, Forestry and Fisheries of Japan (Rice Genome Project), and New Energy and Industrial Technology Development Organization (NEDO).

# References

Alexandrescu,A. (2001) *Modern C++ Design: Generic Programming and Design Patterns Applied.* Addison Wesley Professional, Boston.

Bartol,T.M.Jr., Stiles,J.R., Salpeter,M.M., Salpeter,E.E. and Sejnowski,T.J. (1996) MCELL: generalized Monte Carlo computer simulation of synaptic transmission and chemical signaling. *Soc. Neurosci. Abs.*, **22**, 1742.

Dormand,J.R. and Prince,P.J. (1980) A family of embedded Runge-Kutta formulae. *J. Comp. Appl. Math.*, **6**, 19-26.

Fehlberg,E. (1969) Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems. NASA Technical Report, NASA-TR-R-315.

Gear,C.W. (1971) *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, New Jersey.

Gibson,M.A. and Bruck,J. (2000) Efficient exact stochastic simulation of chemical systems with many species and many channnels. *J. Phys. Chem.*, **104**, 1876-1889.

Gillespie,D.T. (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comput. Phys.*, **22**, 403-434.

Gillespie,D.T. (1977) Concernning the validity of the stochastic approach of chemical kinetics. *J. Stat. Phys.* , **16**, 311-319

Gamer,J., Multhaup,G., Tomoyasu,T., McCarty,J.S., Rudiger,S., Schonfeld,H.J., Schirra,C., Bujard,H., Bukau,B. (1996) A cycle of binding and release of the DnaK, DnaJ and GrpE chaperones regulates activity of the Escherichia coli heat shock transcription factor $\sigma^{32}$. *EMBO J.* **15**, 607-617

Gross,C.A. Function and regulation of the heat shock proteins. In Neidhardt,F.C. and Ingraham,J.L. (eds), *Escherichia coli and Salmonella: Cellular and Molecular Biology*. American Society for Microbiology, Washington D.C., pp.1382-1399.

Grossman,A.D., *et al.* (1987) $\sigma^{32}$ synthesis can regulate the synthesis of heat shock proteins in *Escherichia coli*. *Genes& Dev.*, **1**, 179-184.

Irvine,D.H. and Savageau,M.A. (1990) Efficient solution of nonlinear ordinary differential equations expressed in S-System canonical form. *Siam. J. Numer. Anal.*, **27**, 704-735.

Marion, G., Renshaw, E., Gibson, G. (1998) Stochastic effects in a model of nematode infection in ruminants. *IMA Journal of Mathematics Applied in Medicine and Biology*, **15, 2**, 97-116.

Matsumoto,M. and Nishimura,T. (1998) Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation*, **8**, 3-30.

Morton-Firth,C.J. and Bray, D. (1998) Predicting temporal fluctuations in an intracellular signalling pathway. *J. Theor. Biol.*, **192**, 117-128.

Press,W.H., Teukolsky,S.A., Vetterling,W.T. and Flannery,B.P. (eds) (2002) *Numerical Recipes in C++: The Art of Scientific Computing.* Cambridge University Press, New York.

Rao,C.V., Wolf,D.M. and Arkin,A.P. (2002) Control, exploitation and tolerance of intracellular noise. *Nature*, **420**, 231-237.

Shampine,L.F. (1986) Some Practical Runge-Kutta Formulas. *Math. Comput.*, **46**, 135-150.

Singer, K. (1953) Application of the theory of stochastic processes to the study of irreproducible chemical reactions and nucleation process. *J. R. Stat. Soc. B*, **15**, 92-106.

Srivastava,R., Peterson,M.S. and Bentley,W.E. (2001) Stochastic kinetic analysis of the *Escherichia coli* stress circuit using sigma(32)-targeted antisense. *Biotechnol. Bioeng.*, **75**, 120-129.

Srivastava, R., You, L., and Yin, J. (2002) Stochastic vs. Deterministic Modeling of Intracellular Viral Kinetics. *J. Theor. Biol.* **218**, 309-321.

Takahashi,K., Yugi,K., Hashimoto,K., Yamada,Y., Pickett,C. and Tomita,M. (2002) Computational challenges in cell simulation. *IEEE Intell. Syst.*, **17**, 64-71.

Weimar,J.R. (2002) Cellular automata approaches to enzymatic reaction networks. In Bandini,S., Chopard,B. and Tomassini,M.(eds), *Cellular Automata (Fifth International Conference on Cellular Automata for Research and Industry ACRI).* Springer-Verlag, Berlin, pp.294-303.

Yugi,K., Nakayama,Y. and Tomita,M. (2002) A hybrid static/dynamic simulation algorithm: Towards large-scale pathway simulation. *Proceedings of the Third International Conference on Systems Biology.* 235.

Zeigler,B.P., Praehofer,H. and Kim,T.G. (2000) *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, San Diego.

|  | Timing (sec.) | Speed | $\sigma^{32}$ Std. Dev. | $\sigma^{32}$ Level | DnaJ Std. Dev. | DnaJ Level |
|---|---|---|---|---|---|---|
| Stochastic | 507.7±0.76 | 1 | 3.74±0.072 | 15.18±0.391 | 5.273±0.206 | 464.4±0.498 |
| Composite | 1.445±3.5e-4 | 351 | 3.64±0.112 | 14.83±0.348 | 3.156e-2±0.282e-2 | 464.4±0.007 |
| Deterministic | 3.703 | 137 | 0 | 15.01 | 0 | 464.4 |

Table 1: Benchmark results of the heat-shock model
The three variations, stochastic, composite, and deterministic, of the heat-shock model were run for 100 seconds. Each timing is an average of ten runs. Only the random number seed was changed from run to run in stochastic and composite simulations. $\xi = 0$. The mean levels and the standard deviations of $\sigma^{32}$ were calculated over the simulation results, and trapezoid method was used for the mean calculation. Dormand-Prince 5(4)7M algorithm has been used for ODE parts in the composite and deterministic runs. The results do not include times for program invocation and parsing of the XML model file. See also Figure 4.

| $\lvert S\rvert$ | Predicted Cost | Timing | Predicted Speed | Speed |
|---|---:|---:|---:|---:|
| 1 | $2^{19}\cdot 20$ $[100\%]$ | 68m43s | 1.00 | 1.00 |
| 2 | $2^{9}\cdot 10+2^{19}\cdot 10$ $[\sim 50\%]$ | 35m13s | 2.00 | 1.95 |
| 4 | $5(2^{4}+2^{9}+2^{14}+2^{19})$ $[\sim 26\%]$ | 18m16s | 3.88 | 3.76 |
| 10 | $4\cdot\frac{4^{10}-1}{4-1}$ $[\sim 13\%]$ | 9m33s | 7.50 | 7.20 |
| 20 | $\frac{2^{20}-1}{2-1}$ $[\sim 10\%]$ | 7m10s | 10.00 | 9.5 |

Table 2: Benchmark results of the cascade oscillators model

The cascade oscillators model was run for 0.5 seconds. The components were equally divided into one, two, four, ten, and twenty Steppers. An average of three runs is shown for each configuration. Dormand-Prince 5(4)7M algorithm has been used for all the components. The timings include a program startup and an XML parsing. The predicted simulation costs are calculated by assuming that it is directly proportional to values of time constants.

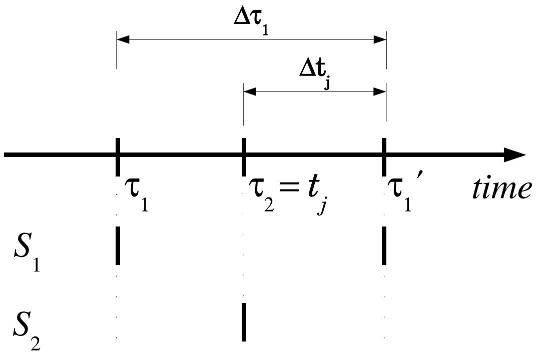

Figure 1: Relation between $\tau$ and $t$
This example has only two continuous Steppers, $S_1$ and $S_2$. Both share a variable $x_j$. Time points $\tau_1$ and $\tau_2$ are where the last steps of these Steppers occured, and $\tau_1'$ $(= \tau_1 + \Delta\tau_1 = T)$ is the current time. $S_1$ steps at $\tau_1$ and $\tau_1'$. $S_2$ steps at $\tau_2$. $x_j$ is updated at $\tau_1$, $\tau_1'$, and $\tau_2$. If the last Stepper stepped before $\tau_1'$ is $S_2$, then $\tau_2 = t_j = \tau_1' - \Delta t_j$.

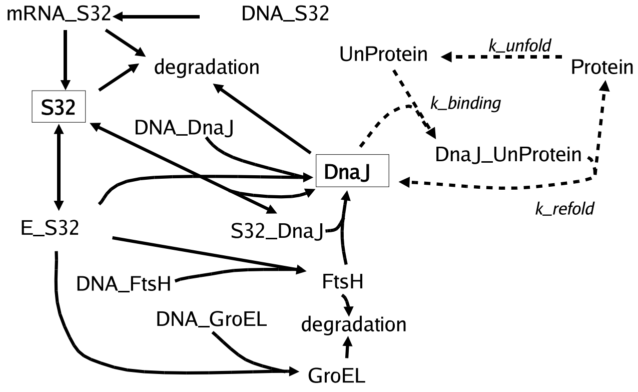

Figure 2: The the heat-shock demonstration model
Model scheme for the heat-shock model. S32 means $\sigma^{32}$; E_S32 means the RNAP core enzyme with $\sigma^{32}$; Protein means folded protein and UnProtein means unfolded protein. Dashed lines in the upper-right corner represent reaction modeled using ODE in the composite model, and Gillespie in the stochastic model. Rate constants for $\sigma^{32}$ transcription and translation are $1.4 \times 10^{-3}$ and $7 \times 10^{-2}$, respectively, which are the only different parameters from the Srivastava's model. *k_unfold* = *k_binding* = 0.2, and *k_refold* = $9.73 \times 10^{6}$.

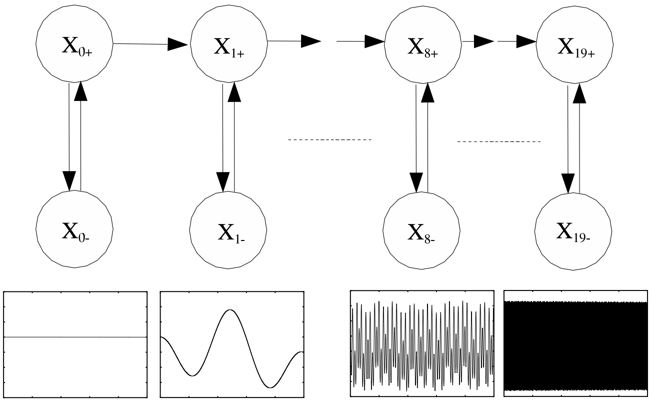

Figure 3: The cascade oscillators demonstration model The model has a basic unit of two variables coupled by the following equations: $\frac{d[X_{i+}]}{dt} = -k2^i[X_{i-}] + k2^{i-1}[X_{(i-1)-}]$, $\frac{d[X_{i-}]}{dt} = k2^i[X_{i+}]$ where $k$ is a time scaling factor. A cascade has twenty instances of the unit. The model has five sets of the cascade. Four sample solutions of the cascade oscillators model are shown below the first (slowest), second, nineth, and twentieth (fastest) components.

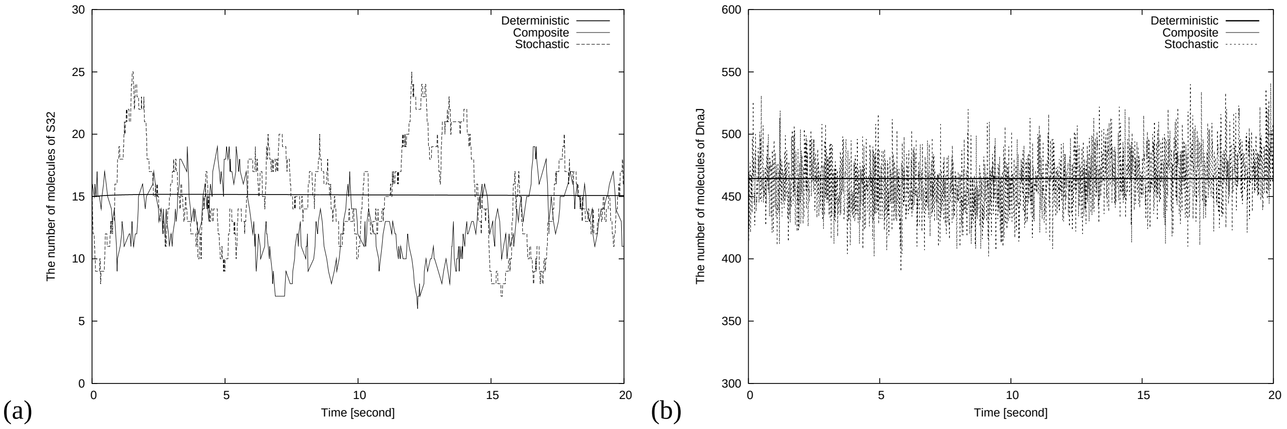

Figure 4: Simulation results of the the heat-shock model

Comparisons of (a) $\sigma^{32}$ and (b) DnaJ time courses of the first 20 seconds of the simulation for the stochastic model, the composite model, and the deterministic model. The composite and the ODE trajectories of DnaJ overlap. It can be seen that means of the ODE (fine curve), the stochastic (dashed curve), and the composite (thick curve) models agree well both in $\sigma^{32}$ and DnaJ cases. The standard deviations of $\sigma^{32}$ in the stochastic and composite models are equivalent. In contrast, the stochastic fluctuations cannot be seen for the DnaJ trajectory of the composite run, making it equivalent to the ODE model (see Table 1 for numerical values).