BIOINFORMATICS ORIGINAL PAPER Vol. 28 no. 11 2012, pages 1508–1516
doi:10.1093/bioinformatics/bts166

Systems biology

Advance Access publication April 5, 2012

Approximate probabilistic analysis of biopathway dynamics

Bing Liu, Andrei Hagiescu, Sucheendra K. Palaniappan, Bipasa Chattopadhyay, Zheng Cui, Weng-Fai Wong and P. S. Thiagarajan*

Department of Computer Science, National University of Singapore, 117417 Singapore, Singapore

Associate Editor: Trey Ideker

ABSTRACT

Motivation: Biopathways are often modeled as systems of ordinary differential equations (ODEs). Such systems will usually have many unknown parameters and hence will be difficult to calibrate. Since the data available for calibration will have limited precision, an approximate representation of the ODEs dynamics should suffice. One must, however, be able to efficiently construct such approximations for large models and perform model calibration and subsequent analysis.

Results: We present a graphical processing unit (GPU) based scheme by which a system of ODEs is approximated as a dynamic Bayesian network (DBN). We then construct a model checking procedure for DBNs based on a simple probabilistic linear time temporal logic. The GPU implementation considerably extends the reach of our previous PC-cluster-based implementation (Liu *et al.*, 2011b). Further, the key components of our algorithm can serve as the GPU kernel for other Monte Carlo simulations-based analysis of biopathway dynamics. Similarly, our model checking framework is a generic one and can be applied in other systems biology settings.

We have tested our methods on three ODE models of biopathways: the epidermal growth factor–nerve growth factor pathway, the segmentation clock network and the MLC-phosphorylation pathway models. The GPU implementation shows significant gains in performance and scalability whereas the model checking framework turns out to be convenient and efficient for specifying and verifying interesting pathways properties.

Availability: The source code is freely available at http://www.comp.nus.edu.sg/~rpsysbio/pada-gpu/

Contact: thiagu@comp.nus.edu.sg

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on December 2, 2011; revised on March 13, 2012; accepted on April 2, 2012

1 INTRODUCTION

The modeling and analysis of biopathways dynamics is a core activity in systems biology. A standard approach is to view a biopathway as a network of biochemical reactions and to model the network as a system of ordinary differential equations (ODEs; Aldridge *et al.*, 2006). Since biopathways often involve a large number of reactions, the corresponding ODE systems will not admit closed form solutions and one will have to resort to numerical simulations. However, the ODE systems will often contain many unknown parameters (rate constants and initial concentration levels) which will first have to be estimated using meager data of limited precision. Consequently, for large pathways model construction and analysis are difficult problems. With this as motivation, a probabilistic approximation method was developed in Liu *et al.* (2009) by which an ODE system is reduced to a dynamic Bayesian network (DBN). Parameter estimation followed by sensitivity analysis is then carried out on this simpler model using standard Bayesian inference techniques. This method is promising in terms of efficiency, accuracy and applicability (Liu *et al.*, 2011a, b). Our goal here is to extend this scheme in two significant ways. The core features of these two extensions are of independent interest and can be deployed in other settings involving biopathways models.

The first extension is a parallel implementation of the DBN approximation scheme using graphical processing units (GPUs). It is computationally intensive to construct the DBN from a system of ODEs. In our experience, a single PC is hopelessly inadequate while even a PC-cluster quickly runs into scalability issues. On the other hand, a supercomputing facility may not be available or affordable. In comparison, GPUs provide an excellent combination of cost and performance. However, not all algorithms map well onto a GPU platform due to its memory hierarchy. Specifically, one must carefully balance parallelism with memory accesses to obtain good performance.

In our DBN construction—explained in more detail in the next section—a computationally intensive phase is the generation of a large number of trajectories using numerical simulations. This can be done in parallel and hence the GPU platform is a natural choice. However, each variable can appear in multiple equations. Hence to generate a trajectory one must, in principle, access all the equations in each integration step. Further, the threads generating the trajectories will have to record a good deal of intermediate information to construct the conditional probability tables (CPTs) of the DBN. For large ODE systems the intermediate data generated will be too large to be stored in the fast local memory. One will instead have to use the slow global memory for this purpose. This can lead to a severe degradation in performance. To get around this we create a heterogeneous pool of threads in which each trajectory is computed in a distributed fashion whereas a second group of threads manage the data movement. There are other settings in which a set of trajectories (Li *et al.*, 2010; Lüdtke *et al.*, 2008) is generated which is then subjected to statistical analysis to derive system properties. In such applications too, our compilation strategy will likely lead to high-performance GPU implementations.

The second major contribution of the article is a probabilistic model checking procedure for DBNs. We use a simple probabilistic variant of linear time temporal logic (LTL; Pnueli, 1977) in which

*To whom correspondence should be addressed.

the atomic propositions are of the form $(X,v)\leq c$ or $(X,v)\geq c$ where X is a finite-valued random variable corresponding to a node in the DBN and c is rational number in $[0,1]$. The assertion $(X,v)\leq c$ says that the probability of the random variable X currently having the value v is $\leq c$; similarly for the assertion $(X,v)\geq c$. The rest of the syntax is standard. Though probability enters the logic only via atomic propositions it turns out that one can still express many interesting dynamical properties. A key component of our model checking procedure is a Bayesian inferencing algorithm called the factored frontier algorithm (Murphy and Weiss, 2001) that is used to *approximately* determine the truthhood of the atomic propositions. We do this approximately since exact inference is computationally infeasible for large DBNs. The accuracy of this inferencing procedure can be improved—at additional computational cost—by using a parametrized variant of the FF algorithm that we have recently developed (Palaniappan *et al.*, 2011; more details can be found in the Supplementary Material.) Again, this simple and novel version of probabilistic verification can be applied to other related settings (Langmead *et al.*, 2006).

Figure 1 shows the overall framework in which we have carried our work. As indicated, our approximation procedure—implemented on a GPU platform—will often yield an uncalibrated DBN due to the presence of unknown parameters. We then perform parameter estimation using Bayesian inferencing to obtain a calibrated model. One can then carry out—on a PC platform—tasks such as sensitivity analysis and probabilistic verification. At present these tasks are performed on a PC platform. In Liu *et al.* (2011b) we show how parameter estimation followed by sensitivity analysis is performed using the Bayesian inferencing. Hence we do not deal with this here and instead focus on the new analysis method, namely, probabilistic verification.

We have tested the GPU implementation on ODE models of the (epidermal growth factor–nerve growth factor) EGF–NGF pathway (Brown *et al.*, 2004), the segmentation clock network (Goldbeter and Pourquie, 2008) and the thrombin-dependent MLC phosphorylation pathway (Maeda *et al.*, 2006). We obtained these models from the BioModels database (Le Novere *et al.*, 2006). Although the parameter values for these models are known, in each case we set a subset of the parameters as unknown to mimic realistic biopathways models. This considerably increases the computational demands placed on the DBN construction. For each model, we constructed a DBN by generating 3 million trajectories. We then compared the performance of the GPU implementation with that of a 10-CPU cluster. The EGF–NGF model consists of 32 differential equations and 48 kinetic parameters from which 20 were set to be unknown. The GPU implementation ran 26 times faster. The segmentation clock network model consists of 22 differential equations and 75 kinetic parameters out of which 40 were set to be unknown. In this case the GPU implementation ran 32 times faster. The third model was that of the thrombin-dependent MLC phosphorylation pathway consisting of 105 differential equations and 197 kinetic parameters out of which we set 164 to be unknown. In this case, the GPU implementation took 38 h whereas the cluster implementation turned out to be infeasible. Even for 30 000 trajectories it took 37 h and hence for a 3 million trajectories, it would have taken 5 months. Further, the 30 000-trajectories-based DBN that the PC-cluster produced was of poor quality.

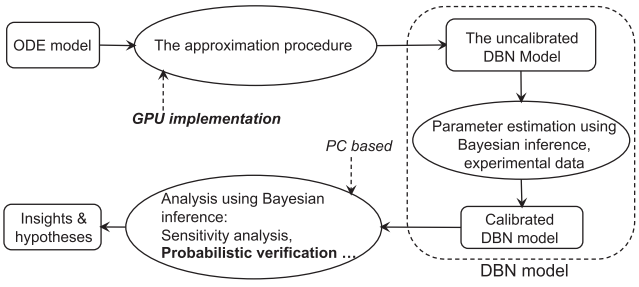

Fig. 1. The approximation, calibration and analysis framework.

We also tested our model checking procedure on the DBN approximations of these three pathway models. For each model we formulated a number of dynamical properties and verified them to be true or false. It only took <1 s on a PC to verify each property.

Turning to related work, an excellent overview of biopathways dynamics and formalisms for studying their behaviors can be found in Klipp *et al.* (2005). Modeling and analysis of biopathways using ODEs is well-established domain (Aldridge *et al.*, 2006; Sreenath *et al.*, 2008). As mentioned earlier, our approximation was presented in Liu *et al.* (2011b) and has been applied in a concrete biological setting (Liu *et al.*, 2011a). As for related GPU applications, a survey of hardware accelerators for biocomputing including GPUs is presented in Dematte and Prandi (2010). Of particular relevance is the Python language-based package called `cuda-sim` reported in Zhou *et al.* (2011). This package enables accelerated simulations of biochemical network models on GPUs. Apart from ODEs-based models, the `cuda-sim` package also supports models based on stochastic differential equations as well as Markov Jump processes. We consider this work to be orthogonal to ours in the sense its main focus is on the simulations of ODEs. In our setting, generating a large number of trajectories through numerical integration is just one component of the larger task of constructing the DBN approximation. At present we do not have a GPU-based implementation of the parameter estimation, sensitivity analysis and probabilistic verification procedures. In this connection, the work on the GPU implementation of the sum-of-product algorithm (Silberstein *et al.*, 2008) for DBNs is promises to be very relevant.

As for the second component of our work, model checking based on temporal logics is a well-established domain. It was founded in Pnueli (1977) through the seminal work of Amir Pnueli where he proposed LTL as the basis for reasoning about the behavior of programs. An ideal starting point for delving into this field is Clarke *et al.* (1999) which presents the basic family of temporal logics as well the automated verification procedure called model checking. This framework was extended to probabilistic systems—based on discrete time Markov chains—using the probabilistic temporal logic called *PCTL* in Hansson and Jonsson (1994). It is worth noting that our specification logic *PBL* is simple fragment of *PCTL*. Overviews of this active area of research can be found for instance in Kwiatkowska *et al.* (2010) and Legay *et al.* (2010).

A number of modeling formalisms (Ciocchetta and Hillston, 2009; Danos *et al.*, 2007; Henzinger *et al.*, 2010; Kwiatkowska *et al.*, 2008) for biochemical networks are based on continuous time Markov chains (CTMCs). Model checking the properties of these stochastic models using PCTLs such as *PCTL* and its variants is being actively pursued by a number of researchers (Clarke *et al.*, 2008; Gong *et al.*, 2010; Kwiatkowska *et al.*, 2008; Li *et al.*, 2010). Of particular interest in our context is Donaldson and Gilbert (2008)

which defines the probabilistic LTL with numerical constraints (PLTLc) and verifies properties of ODE models through Monte Carlo integration. Here instead we carry out Monte Carlo simulations *once* to construct a DBN approximation. One can then repeatedly perform probabilistic verification of properties expressed in *PBL* using Bayesian inferencing. In Clarke *et al.* (2008), a technique called statistical model checking is used to verify properties of a CTMC model expressed in a logic called bounded LTL (BTL) which is basically LTL interpreted over finite sequences. They stochastically simulate a CTMC model for a finite number of runs and use BTL to check each simulation run. They then perform statistical tests to verify that this property meets a required probability bound. Our logic *PBL* is also interpreted over finite sequences. However it is a probabilistic logic in that the atomic propositions make probabilistic assertions. Further, the logic is interpreted over the *single* sequence of marginal distributions generated by the DBN (as explained in Section 4). Finally, the work reported in Langmead *et al.* (2006) also carries out DBN-based model checking. However, the DBN is first converted to a data structure called a multi-terminal binary decision diagram (MTBDD) and then existing probabilistic model checking algorithms based on *PCTL* are applied. Additional background information on these topics can be found in the Supplementary Material.

In the next section, we explain how a DBN is constructed from a system of ODEs. In Section 3, we present the main features of our GPU implementation. In the subsequent section, we introduce our probabilistic variant of LTL and the associated model checking procedure. In Section 5, we present our experimental results. The final section summarizes and discusses possible extensions of the work presented here.

2 THE DBN APPROXIMATION

The dynamics of a biopathway is often modeled as a system of ODEs with one equation of the form $dx/dt = f(\mathbf{x}, \mathbf{p})$ for each molecular species x in the pathway. Here f describes the kinetics of the reactions that produce and consume x and $\mathbf{x}$ are the molecular species taking part in these reactions whereas $\mathbf{p}$ are the rate constants governing these reactions. For large pathways, this ODE system which will typically have many unknown parameters will be difficult to calibrate and analyze. To get around this an approximation scheme was developed in (Liu *et al.*, 2009) through which a system of ODEs can be reduced to a DBN. Our goal here is to briefly explain this method. The technical details can be found in Liu *et al.* (2011b).

First, we assume the states of the system are observed only at a finite number of time points, $\{0, 1, \ldots, T\}$. Next, the range of each variable x_i (rate constant r_j) is partitioned into a set of intervals $\mathbf{I}_i$ ($\mathbf{I}_j$). Both these discretizations are motivated by the fact that experimental data will be available only for a finite set of time points and this data will be of limited precision.

Next, the initial values of the variables as well as the rate constants are assumed to be distributions (usually uniform) over certain of these intervals. We then sample the initial states of the system according to this distribution sufficiently many times, and generate a trajectory by numerical integration for each sampled initial state. The resulting set of trajectories is then treated as an approximation of the dynamics of ODE system.

To handle unknown rate constants we assume that the minimum and maximum values of these constants are known. We then partition these ranges of values also into a finite numbers of intervals, and fix a uniform distribution over *all* the intervals. After building the DBN, we use a Bayesian inference-based technique to perform parameter estimation to complete the construction of the model. However, unlike the variables, once the initial value of an unknown rate constant has been sampled, this value will not change during the generation of a trajectory. Naturally different trajectories can have different initial values for an unknown rate constant.

A key idea is to compactly store the generated set of sequences as a DBN. This is achieved by means of a simple counting procedure that exploits the network structure. In order to keep the focus on the approximation procedure we give only an informal description of DBNs here and defer a precise presentation to Section 4.

A DBN consists of a directed acyclic graph where the nodes are grouped into layers with each layer representing a time point (Murphy, 2002). The nodes in layer $t-1$ will be connected to the nodes in the layer t in the same way as t ranges from 1 to T. Each node will have a random variable associated with it. In our setting, there will be one random variable x_i^t (r_j^t) corresponding to each variable x_i (unknown rate constant r_j) to capture in which interval the value of x_i (r_j) falls at time t. Further, for each unknown rate constant k, we add the equation $dk/dt = 0$ to capture the fact that once the value of k has been sampled, this value will not change during the numerical integration of a trajectory.

$\mathrm{Pa}(x_i^t)$, the set of parent nodes of x_i^t is determined as follows. The node x_k^{t-1} (r_j^{t-1}) will be in $\mathrm{Pa}(x_i^t)$ iff x_k (r_j) appears in the equation for x_i or $x_k = x_i$. On the other hand, r_j^{t-1} will be the only parent of the node r_j^t in case r_j is an unknown rate constant. In Figure 2, we show a simple enzymatic reaction network, its ODE model and the structure of its DBN approximation. In this example, we have assumed that k_3 is the only unknown parameter.

As indicated in Figure 2c, each node will also have a CPT associated with it to specify the local probabilistic dynamics. A typical entry in the CPT of x_i^t will be of the form $\Pr(x_i^t = I \mid z_1^{t-1} = I_1, z_2^{t-1} = I_2, \ldots, z_l^{t-1} = I_l) = p$ with $\mathrm{Pa}(x_i^t) = \{z_1^{t-1}, z_2^{t-1}, \ldots, z_l^{t-1}\}$. Such an entry means that p is the probability that the value of x_i falls in the interval I at time t, given that the value of z_u was in I_u at time $t-1$ for each z_u^{t-1} in $\mathrm{Pa}(x_i^t)$. The probability p is

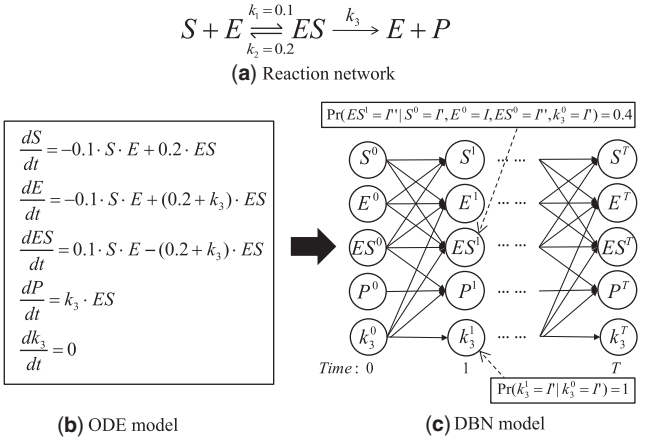

Fig. 2. (**a**) The enzyme catalytic reaction network. (**b**) The ODE model. (**c**) The DBN approximation.

calculated through simple counting. Suppose N is the number of generated trajectories. We record the number of the trajectories from this collection for which their value of z_u fell in the interval I_u for each z_u in $\{z_1, z_2, \ldots, z_l\}$ at time $t-1$. Suppose this number is J. We then determine for how many of these J trajectories, the value of x_i fell in the interval I at time t. If this number is J', then p is set to be J'/J.

If k is unknown, in the CPT of k^t we will have $\Pr(k^t = I \mid k^{t-1} = I') = 1$ if $I = I'$ and $\Pr(k^t = I \mid k^{t-1} = I') = 0$ otherwise. This is because the sampled initial value of k does not change during numerical integration. Suppose k appears on the right-hand side of the equation for x and $\mathrm{Pa}(x_i^t) = \{z_1^{t-1}, z_2^{t-1}, \ldots, z_\ell^{t-1}\}$ with $z_\ell^{t-1} = k^{t-1}$. Then for each choice of interval values for nodes other than k in $\mathrm{Pa}(x_i^t)$ and for each choice of interval value $\widehat{I}$ for k there will be an entry in the CPT of x^t of the form $\Pr(x_i^t = I \mid z_1^{t-1} = I_1, z_2^{t-1} = I_2, \ldots, k = \widehat{I}) = p$. This is so since we will sample for all possible initial interval values for k and $k^0 = k^{t-1}$. In this sense, the CPTs record the approximated dynamics for all possible combinations of interval values for the unknown rate constants. These features are illustrated in Figure 2c for the unknown rate constant k_3.

By performing parameter estimation on the resulting uncalibrated DBN one can obtain a calibrated DBN in which each parameter will have a specific interval value assigned to it.

3 MAPPING TO A GPU ARCHITECTURE

We now describe how our approximation algorithm is mapped onto the NVIDIA's Fermi platform. In this platform each GPU unit consists of 16 *streaming multiprocessors* (SMs for short). Each SM has 32 arithmetic cores divided into 2 groups of 16 each for scheduling purposes (Glaskowsky, 2009). Threads are grouped into *warps* of size 32 each. In the current generation of CUDA processors, half of a warp, i.e. 16 threads, will execute in one of the 2 sets of cores in a SM in a SIMD (single instruction multiple data) manner. Thus in any cycle—subject to some mild constraints—two half-warps can be executing in a SM. However, the threads belonging to a half warp must execute the same instruction. If not, they must be serialized. Pipelining offers an additional dimension of parallelism. Loosely speaking, it takes ~22 cycles to execute each instruction. Through pipelining, when the $i+1$-th processing cycle of the instructions belonging to a warp starts executing, the system can schedule the execution of the i-th processing cycle of the instructions belonging to a different wrap. As for the memory hierarchy, each SM has 32 K registers of 32 bits length and each thread in an SM can be allocated upto 64 of these registers. Each SM has 48 KB of shared fast memory which the threads belonging to the SM can use to store data and synchronize with each other. Additional data must be stored in the slow off-chip global memory of size 2 GB. This is also the only medium through which threads belonging to different SMs can synchronize. In addition, there is a small $L1$ cache of size 16 KB belonging to each SM and an $L2$ cache of size 768 KB that is shared by all the SMs. Each SM computes a set of trajectories and records the number of times these trajectories hit the intervals of values of the variables at different time points. This binning information is stored in a specific area of the global memory which will be summed up to produce the CPTs of the DBN. We now describe how the computation within a single SM is orchestrated according to the scheme shown in Figure 3.

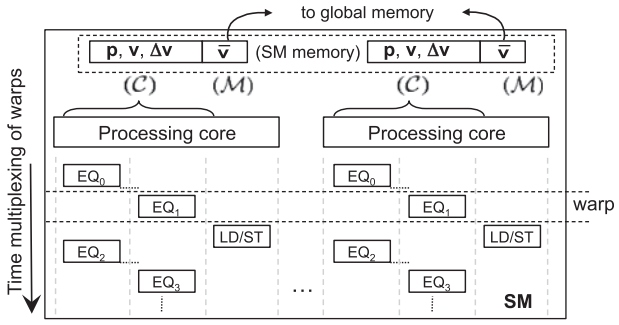

Fig. 3. Concurrent execution of trajectories inside an SM.

Starting from an initial state at $t=0$, for each time interval Δt, the new value of a variable x is determined by applying numerical integration using the current values of the variables and the values of the rate constants appearing in the ODE for x as well as the current value of x. Due to the coupling between the variables, the entire front of the new values of all the variables must be computed at each time step per trajectory. If we naively allocate as many threads as possible to each SM with each thread computing a trajectory then their memory requirements will exceed the size of the (fast) local memory of the SM.

To get around this we partition the set of equations into blocks and allocate each block to a thread. Thus a single trajectory will be computed by a *set* of threads $\mathcal{C}$. Each member of $\mathcal{C}$ will handle a different block of equations and compute the new values of the variables appearing on the left-hand sides of these equations. Each thread will read from the local memory of the SM the current values of the variables ($\mathbf{v}$) and the rate constants ($\mathbf{p}$) appearing in the equations allotted to it. The $\Delta\mathbf{v}$ changes during a time step are computed in parallel and are stored back to the local memory. The vector of variables $\mathbf{v}$ is then updated. This process is applied iteratively for each time step.

We must also perform the binning steps to record the number of times the threads hit the various intervals of values of the variables (and unknown rate constants). This is required for constructing the CPTs of the DBN. Accordingly, the vector $\mathbf{v}$ is replicated as $\bar{\mathbf{v}}$. The binning process executes in parallel, during the subsequent Δt iteration, using the memory access threads ($\mathcal{M}$), which will store the results in a large table located in global memory. This will ensure that the numerical integration continues during the binning process.

Many copies of the $\mathcal{C}$ and $\mathcal{M}$ groups of threads will be assigned to an SM. How they are scheduled is guided by the hardware organization of the SM explained above. In particular, we allot the $\mathcal{C}$ threads belonging to each trajectory to the warps so that threads executed together in the same warp process the *same* block of equations corresponding to *different* trajectories. This implies that threads computing different blocks corresponding to the same trajectory are assigned to different warps.

We next turn to the issue of partitioning the set of equations into blocks with each block handled by a thread in $\mathcal{C}$. The compiler converts each equation into a set of memory loads and stores and a set of arithmetic operations. To achieve good balance, we have created a simple timing model that assigns a weight to each operation that appears in each equation. We then distribute the equations so that the corresponding weight of the operators appearing among the $\mathcal{C}$ threads is balanced.

Our compilation strategy leads to a better utilization of the fast local memory in comparison to the naïve implementation in which each thread computes a trajectory and the CUDA compiler decides how many parallel threads should be launched. Our experiments using an initial implementation showed that for one of the case studies reported in Section 5 (the EGF–NGF model) it achieves approximately a 40% improvement over the naïve solution. We have not, however, carried out a systematic and detailed comparison so far.

It is also worth pointing out that there will be ODE systems on which our implementation will not do much better than the naïve implementation. This is so since structure of the DBN is determined by the couplings between the variables in the ODE system. Hence our solution will not do much better for networks in which the graph denoting the couplings between the variables is not sparse but instead has many nodes that have a large number of parent nodes. This situation will arise when the biochemical network under study has many species each of which takes part in many reactions as a reactant and/or product. However all the networks we have encountered so far have been relatively sparse. When there are a few 'fat' nodes that have many parent nodes, one can break them up into leaner nodes—at an additional computational cost—as explained in Liu *et al.* (2011b).

To generate the initial states, we used a Mersenne twister algorithm based on the *MT*19937 random number generator (Matsumoto and Nishimura, 1998), running in each of the $\mathcal{C}$ threads. We used a fourth-order Runge-Kutta algorithm for the numerical integration process (Hindmarsh, 1983). We have extended the implementation described above to multiple GPUs running in parallel. Each GPU computes independently its portion of the trajectories and records the results in a table in its own global memory. Once the computation has finished, these tables are transferred over the network to the master host that combines them to form the CPTs. More details regarding our GPU implementation can be found in the Supplementary Material.

4 PROBABILISTIC MODEL CHECKING

Once the DBN approximation has been constructed many analysis tasks can be carried out efficiently, such as parameter estimation and sensitivity analysis via Bayesian inferencing (Liu *et al.*, 2011b). Here we formulate a new analysis method based on probabilistic model checking. As mentioned earlier, the FF algorithm (FF for short) will play a key role in our model checking procedure. Hence we start with DBNs and then describe FF.

4.1 The FF algorithm

We fix an ordered set of n random variables $\{X_1,\ldots,X_n\}$ and let i, j range over $\{1,2,\ldots,n\}$. We denote by $\mathbf{X}$ the tuple $(X_1,\ldots,X_n)$. The random variables are assumed to take values from the finite set V of cardinality K. We let x_i, u_i and v_i to denote a value taken by X_i. In our application the random variables will correspond to the variables (and unknown rate constants) appearing in the ODE model. Our DBNs will be time-variant but with a regular structure. They will be unrolled over a finite number of time points. Further, there will be no hidden variables (Murphy and Weiss, 2001).

Formally, a DBN is a structure $\mathcal{D}=(\mathcal{X},T,\mathrm{Pa},\{C_i^t\})$ where,

- T is a positive integer with t ranging over the set of time points $\{0,1,\ldots,T\}$.
- $\mathcal{X}=\{X_i^t \mid 1\le i\le n, 0\le t\le T\}$ is the set of random variables. As usual, these variables will be identified with the nodes of the DBN. X_i^t is the instance of X_i at time slice t.
- Pa assigns a set of parents to each node and satisfies: (i) $\mathrm{Pa}(X_i^0)=\emptyset$). (ii) If $X_j^{t'}\in\mathrm{Pa}(X_i^t)$ then $t'=t-1$. (iii) If $X_j^{t-1}\in\mathrm{Pa}(X_i^t)$ for some t then $X_j^{t'-1}\in\mathrm{Pa}(X_i^{t'})$ for every $t'\in\{1,2,\ldots,T\}$. Thus the way nodes at the $(t-1)$-th time slice are connected to nodes at the t-th time slice remains invariant as t ranges over $\{1,2,\ldots,T\}$.
- C_i^t is the *CPT* associated with node X_i^t specifying the probabilities $\Pr(X_i^t\mid\mathrm{Pa}(X_i^t))$. Suppose $Pa(X_i^t)=\{X_{j_1}^{t-1},X_{j_2}^{t-1},\ldots,X_{j_m}^{t-1}\}$ and $(x_{j_1},x_{j_2},\ldots,x_{j_m})\in V^m$. Then as usual we require, $\sum_{x_i\in V}C_i^t(x_i\mid x_{j_1},x_{j_2},\ldots,x_{j_m})=1$. Since our DBNs are time-variant, in general C_i^t will be different from $C_i^{t'}$ if $t\neq t'$.

A (global) state of the DBN at t will be a member of V^n, say $\mathbf{x}=(x_1,x_2,\ldots,x_n)$ specifying that $X_i^t=x_i$ for $1\le i\le n$. This in turn stands for $X_i=x_i$ for $1\le i\le n$ at t. Suppose $Pa(X_i^t)=\{X_{j_1}^{t-1},X_{j_2}^{t-1},\ldots,X_{j_m}^{t-1}\}$. Then a CPT entry of the form $C_i^t(x_i\mid x_{j_1},x_{j_2},\ldots,x_{j_m})=p$ says that if the system is in a state at $t-1$ in which $X_{j_\ell}=x_{j_\ell}$ for $1\le\ell\le m$, then the probability of $X_i=x_i$ being the case at t is p. In this sense the CPTs specify the probabilistic dynamics locally.

The regular structure of our DBNs induces the function *PA* given by: $X_j\in PA(X_i)$ iff $X_j^{t-1}\in\mathrm{Pa}(X_i^t)$. We define $\hat{i}=\{j\mid X_j\in PA(X_i)\}$ to capture Pa in terms of the corresponding indices.

The probability distribution $\Pr(X_1^t,X_2^t,\ldots,X_n^t)$ describes the possible states of the system at time t. In other words, $\Pr(\mathbf{X}^t=\mathbf{x})$ is the probability that the system will reach the state $\mathbf{x}$ at t. Starting from $\Pr(\mathbf{X}^0)$ at time 0, given by $\Pr(\mathbf{X}^0=\mathbf{x})=\prod_i C_i^0(\mathbf{x}_i)$, one would like to compute $\Pr(X_1^t,\ldots,X_n^t)$ for a given t.

We can use the CPTs to inductively compute this:

$$\Pr(\mathbf{X}^t=\mathbf{x})=\sum_{\mathbf{u}}\Big(\prod_i C_i^t(\mathbf{x}_i\mid\mathbf{u}_{\hat{i}})\Big)\Pr(\mathbf{X}^{t-1}=\mathbf{u}) \qquad (1)$$

with $\mathbf{u}$ ranging over V^n.

Since $|V|=K$, the number of possible states at t is K^n and hence explicitly computing and maintaining the probability distributions is feasible only for special cases. One must instead use approximate methods. Here we shall focus on a simple and efficient approximate algorithm called the FF algorithm (Murphy and Weiss, 2001). The

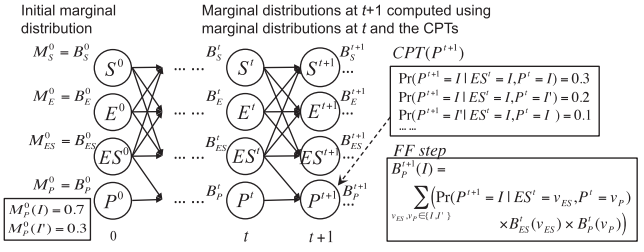

Fig. 4. The FF algorithm.

key idea of FF is to maintain and propagate joint probability distributions $\Pr(X_1^t, X_2^t, \ldots, X_n^t)$ in terms of *marginal* distributions $\{M_i^t\}$.

The marginal distribution M_i^t is map $M_i^t : V \to [0,1]$ satisfying $\sum_{v \in V} M_i^t(v) = 1$. Intuitively, $M_i^t(v)$ is the probability of X_i assuming the value v at time t. It is given by: $M_i^t(v) = \sum_{\mathbf{x}, \mathbf{x}(i)=v} \Pr(X_j^t = \mathbf{x}(j) \mid 1 \le j \le n)$.

To describe FF compactly we will assume the following notations. $\mathbf{x}_J$ will denote a vector of values over the index set $J \subseteq \{1, 2, \ldots, n\}$. It will be viewed as a map $\mathbf{x}_J : J \to V$. We will often denote $\mathbf{x}_J(i)$ as $\mathbf{x}_i$ if J is clear from the context. Further, we denote by $\mathbf{X}^t$ the vector of random variables $(X_1^t, \ldots, X_n^t)$.

We assume that we are given M_i^0 for each i. In our application this will correspond to the initial distribution over the intervals of values of the variables in our discretization procedure.

Consequently, if X_j corresponds to an unknown rate constant then $M_j^0(I) = 1/m$ in case the values this rate constant can assume has been partitioned into m uniform sized intervals. To highlight the approximate nature of FF we use B_i^t to denote the marginal probabilities computed by FF and reserve M_i^t for the actual marginal. Using the family $\{M_i^0\}$, FF inductively and approximately computes the marginal probabilities B_i^t using the CPTs as follows.

- $B_i^0 = M_i^0$
- $B_i^t(u) = \sum_{\mathbf{v} \in V_i} \left(C_i^t(u \mid \mathbf{v}) \prod_{j \in \hat{i}} B_j^{t-1}(\mathbf{v}_j) \right)$.

Thus FF generates in one sweep the sequence of (approximate) marginal distribution vectors $(B_1^0, B_2^0, \ldots, B_n^0)$ $(B_1^1, B_2^1, \ldots, B_n^1)$ … $(B_1^T, B_2^T, \ldots, B_n^T)$ as illustrated in Figure 4 (for convenience we have assumed that all the rate constants are known). It is an approximation of the sequence of (exact) marginal distribution vectors $(M_1^0, M_2^0, \ldots, M_n^0)$ $(M_1^1, M_2^1, \ldots, M_n^1)$ … $(M_1^T, M_2^T, \ldots, M_n^T)$ which in turn will be used to interpret our temporal logic formulas later.

The time complexity of FF is $O(T \cdot n \cdot K^{d+1})$ where $|V| = K$ and d is the maximum over the number of parents that a node can have. Usually d will be much smaller than n and in this sense FF is efficient since its time complexity is linear in n.

4.2 Probabilistic BTL

In our temporal logic, the atomic formulas (i.e. propositions) will be of the form $(i, v) \# r$ with $\# \in \{\le, \ge\}$ and $r \in [0,1]$. The proposition $(i, v) \ge r$, if asserted at time point t, says that $M_i^t(v) \ge r$; similarly for $(i, v) \le r$.

The formulas of our logic termed *PBL* (probabilistic BTL) is then given by: (i) Every proposition is a formula. (ii) If φ and φ' are formulas then so are $\sim\varphi$ and $\varphi \vee \varphi'$. (iii) If φ and φ' are formulas then so are $\boldsymbol{O}(\varphi)$ and $\varphi \boldsymbol{U} \varphi'$.

The derived propositional connectives such as $\wedge, \supset, \equiv$ etc. are defined in the standard fashion. The temporal connectives $\boldsymbol{F}$ ("sometime from now") and $\mathbf{G}$ ("always from now") are defined in the usual way via: $\boldsymbol{F}(\varphi) = true \boldsymbol{U} \varphi$ and $\boldsymbol{G}(\varphi) = \sim \boldsymbol{F}(\sim\varphi)$.

The formulas are interpreted over the sequence of marginal probability distribution vectors $\sigma = \mathbf{s}_0 \mathbf{s}_1 \ldots \mathbf{s}_T$ generated by the DBN $\mathcal{D}$. In other words, for $0 \le t \le T$, $\mathbf{s}_t = (M_1^t, M_2^t, \ldots, M_n^t)$. Consequently $\mathbf{s}_t(i) = M_i^t$ for $1 \le i \le n$. We also let $\sigma(t) = \mathbf{s}_t$ for $0 \le t \le T$. We now define the notion of $\sigma(t) \models \varphi$ (φ holds at t in $\mathcal{D}$) inductively:

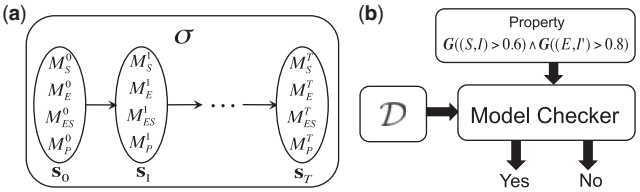

Fig. 5. (**a**) The model (sequence of states) defined by the DBN. (**b**) The model checking procedure.

- $\sigma(t) \models (i, v) \ge r$ iff $M_i^t(v) \ge r$. Similarly $\sigma(t) \models (i, v) \le r$ iff $M_i^t(v) \le r$.
- The propositional connectives $\sim$ and $\vee$ are interpreted in the usual way.
- $\sigma(t) \models \boldsymbol{O}(\varphi)$ iff $\sigma(t+1) \models \varphi$.
- $\sigma(t) \models \varphi \boldsymbol{U} \varphi'$ iff there exists $t \le t' \le T$ such that $\sigma(t') \models \varphi'$ and for every t'' with $t \le t'' < t'$, $\sigma(t'') \models \varphi$.

We say that the DBN $\mathcal{D}$ meets the specification φ and this is denoted as $\mathcal{D} \models \varphi$ iff $\sigma(0) \models \varphi$. The model checking problem is, given $\mathcal{D}$ and φ, to determine whether or not $\mathcal{D} \models \varphi$.

We begin by letting $\mathrm{SF}(\varphi)$ denote the set of sub-formulas of φ and define it as follows. Since φ will remain fixed we will write below SF instead of $\mathrm{SF}(\varphi)$.

SF is the least set of formulas containing φ such that (i) $\sim\varphi' \in$ SF implies $\varphi' \in$ SF (ii) $\varphi' \vee \varphi'' \in$ SF implies $\varphi', \varphi'' \in$ SF (iii) $\boldsymbol{O}\varphi' \in$ SF implies $\varphi' \in$ SF (iv) $\varphi' \boldsymbol{U} \varphi'' \in$ SF implies $\varphi', \varphi'' \in$ SF.

The main step is to construct a labeling function st which assigns to each formula $\varphi' \in$ SF a subset of $\{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_T\}$ denoted $st(\varphi')$. After the labeling process is complete, we declare $\mathcal{D} \models \varphi$ just in case $\mathbf{s}_0 \in st(\varphi)$. Starting with the atomic propositions, the labeling algorithm goes through members of SF in ascending order in terms of their structural complexity. Thus φ' will be treated before $\sim\varphi'$ is treated and both φ' and φ'' will be treated before $\varphi' \boldsymbol{U} \varphi''$ is treated and so on.

Let $\varphi' \in \mathrm{SF}(\varphi)$. Then:

- If $\varphi' = A$ then $\mathbf{s}_t \in st(A)$ iff $\sigma(t) \models A$. We run FF to determine this. In other words, $\mathbf{s}_t \in st(A)$ iff $B_i^t(v) \ge r$ where $A = (i, v) \ge r$ and B_i^t is the marginal distribution of X_i^t computed by FF. Similarly $\mathbf{s}_t \in st(A)$ iff $B_i^t(v) \le r$ in case $A = (i, v) \le r$.
- If $\varphi' = \sim \varphi''$ then $\mathbf{s}_t \in st(\varphi')$ iff $\mathbf{s}_t \notin st(\varphi'')$.
- If $\varphi' = \varphi_1 \vee \varphi_2$ then $\mathbf{s}_t \in st(\varphi')$ iff $\mathbf{s}_t \in st(\varphi_1)$ or $\mathbf{s}_t \in st(\varphi_2)$.
- Suppose $\varphi' = \boldsymbol{O}(\varphi'')$. Then $\mathbf{s}_T \notin st(\varphi')$. Further, for $0 \le t < T$, $\mathbf{s}_t \in st(\varphi')$ iff $\mathbf{s}_{t+1} \in st(\varphi'')$.
- Suppose $\varphi' = \varphi_1 \boldsymbol{U} \varphi_2$. Then we decide whether or not $\mathbf{s}_t \in st(\varphi')$ by starting with $t = T$ and then treating decreasing values of t. Firstly $\mathbf{s}_T \in st(\varphi')$ iff $\mathbf{s}_T \in st(\varphi_2)$. Next suppose $t < T$ and we have already decided whether or not $\mathbf{s}_{t'} \in st(\varphi')$ for $t < t' \le T$. Then $\mathbf{s}_t \in st(\varphi')$ iff $\mathbf{s}_t \in st(\varphi_2)$ or $\mathbf{s}_t \in st(\varphi_1)$ and $\mathbf{s}_{t+1} \in st(\varphi')$.

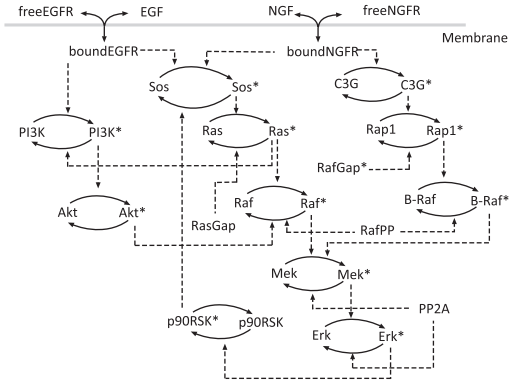

Fig. 6. The reaction network diagram of the EGF-NGF pathway (Brown *et al.*, 2004).

$\varphi' = \boldsymbol{F}(\varphi'')$ and $\varphi' = \boldsymbol{G}(\varphi'')$ can be handled directly. As in the case of $\boldsymbol{U}$, we start with $t = T$ and consider decreasing values of t:

- Suppose $\varphi' = \boldsymbol{F}(\varphi'')$. Then $\mathbf{s}_T \in st(\varphi')$ iff $\mathbf{s}_T \in st(\varphi'')$. For $t < T$, $\mathbf{s}_t \in st(\varphi')$ iff $\mathbf{s}_t \in st(\varphi'')$ or $\mathbf{s}_{t+1} \in st(\varphi')$.
- Suppose $\varphi' = \boldsymbol{G}(\varphi'')$. Then $\mathbf{s}_T \in st(\varphi')$ iff $\mathbf{s}_T \in st(\varphi'')$. For $t < T$, $\mathbf{s}_t \in st(\varphi')$ iff $\mathbf{s}_t \in st(\varphi'')$ and $\mathbf{s}_{t+1} \in st(\varphi')$.

Due to the fact the model checking procedure just needs to treat one finite sequence as a model, it is particularly simple. Its time complexity is linear in the size of the formula φ whereas in traditional settings it will be exponential in the size of φ.

Figure 5 summarizes our model checking procedure. Properties of pathway dynamics are formulated as PBL formulas. They are then verified using the above labeling algorithm which will call the FF algorithm when dealing the atomic propositions.

5 RESULTS

Here, we present our results concerning the GPU implementation and the probabilistic model checking method.

5.1 Performance of the GPU implementation

We have implemented the DBN approximation algorithm on the NVIDIA Tesla 2.0 (Fermi) platform consisting of 4 GPUs of 2 GB global memory each. The 4 units are attached in pairs to 2 Xeon E5405 @ 2 GHz hosts with 16 GB of memory each and the hosts can communicate with each other. We compared the performance of our GPU algorithm with that of a MPI-based C implementation on a cluster of 10 Xeon E5430 @ 2.66 GHz CPUs each with 40 GB of memory. We did so by constructing the DBN approximations for three biopathways models described below. Although the parameter values for these models are known, in each case we set a subset of the parameters as 'unknown' to mimic realistic biopathways models. This considerably increases the computational demands placed on the approximation algorithm. In what follows we briefly describe each of the pathways. The reaction schemes and the corresponding ODE systems can be obtained via the links to the Biomodels database (Le Novere *et al.*, 2006) provided in the Supplementary Material.

5.1.1 The EGF–NGF signaling pathway PC12 cells proliferate in response to EGF stimulation but differentiate into sympathetic neurons in response to NGF. Brown *et al.* (2004) developed an ODE model of this pathway. The corresponding reaction network is shown in Figure 6. The model consists of 32 differential equations and 48 kinetic parameters. In total, 20 of the 48 parameters were singled out to be unknown. The ranges of each variable and unknown parameter were discretized into five intervals of equal size. The time step Δt was fixed to be 6 s and 3×10^6 trajectories were generated up to 600 s to fill up the CPTs associated with the DBN approximation.

5.1.2 The segmentation clock network During the development of vertebrate embryos, the somites are rhythmically produced to establish the segmental pattern of the spines. The periodic formation of somites is driven by the oscillatory expression of a large number of genes. The expression of these genes is controlled by an underlying signaling network called the segmentation clock network (Goldbeter and Pourquie, 2008). The corresponding ODE model consists of 22 differential equations and 75 kinetic parameters. A total of 40 of the 75 parameters were singled out to be unknown. The ranges of each variable and unknown parameter were discretized into five equal-size intervals. The time step Δt was fixed to be 5 min while 3×10^6 trajectories were generated up to 500 mins to fill up the CPTs.

5.1.3 The thrombin-dependent MLC phosphorylation pathway The endothelial cells form a dynamic barrier between blood and tissues, which plays an important role in various physiological and pathological processes. The barrier function is determined by the contraction of endothelial cells, which is triggered by the MLC phosphorylation and thrombin is an agonist that can induce the MLC phosphorylation through two different signaling cascades (Maeda *et al.*, 2006). This rather large model consists of 105 differential equations, 110 reactions and 197 kinetic parameters. In constructing the DBN approximation, we singled out 164 of the 197 parameters to be unknown. We discretized the ranges of each variable and unknown parameter into five equal-size intervals and fixed the time step Δt to be 2 s. To fill up the CPTs, we generated 3×10^6 trajectories up to 200 s.

More details concerning their DBN approximations can be found in the Supplementary Material.

5.2 Performance

The overall runtime are summarized in Table 1. It shows our implementation achieves significant speedup compared with a 10-CPU cluster implementation for the first two case studies. The third case study illustrates the scalability of the GPU implementation. It took 38 h to compute a high-quality DBN approximation using 3×10^6 trajectories whereas the cluster implementation took 37 h for just 30 000 trajectories. Further, this DBN was of poor quality in that for some biologically significant species it differed significantly from the ODE model (the details can be found in the Supplementary Material). For a 3 million trajectories-based DBN approximation, the PC-cluster it would have taken ~5 months! In this sense the GPU implementation can handle much larger models than the PC-cluster. Further, the compilation strategy we have developed can be applied to other Monte Carlo simulations-based analysis methods for biopathways models (Donaldson and Gilbert, 2008; Li *et al.*, 2010).

Table 1. Comparative performance of CPUs cluster and GPUs cluster

Pathway model	Runtime (s)		Speedup
	10-CPU cluster	4-GPU cluster	
EGF–NGF	4985	191.4	26×
Segmentation clock	17 881	543.6	32.9×
Thrombin-MLC	–	135 660.9	–

5.3 Verification results

For the three case studies, we formulated some properties and verified whether they were true or not. For convenience we fixed the values of rate constants and the initial concentrations according to the models taken from the BioModels database. This in turn fixed the truth values of the propositions at time 0.

5.3.1 The EGF–NGF signaling pathway

- It is known that the concentration of EGF and NGF remains constantly high. We formulated this property as the formula:

$$\boldsymbol{G}((\mathrm{EGF}, I_4) > 0.9) \wedge \boldsymbol{G}((\mathrm{NGF}, I_4) > 0.9)$$

  The property was verified to be *true*.

- The profile of activated extracellular-signal-regulated kinase (ERK) is expected to reach a peak after which the concentration begins to fall. The corresponding formula was:

$$(((\mathrm{ERK}^*, I_0) > 0.6) \wedge \boldsymbol{F}(((\mathrm{ERK}^*, I_3) > 0.6) \wedge \boldsymbol{F}(\boldsymbol{G}((\mathrm{ERK}^*, I_2) > 0.6)))$$

  The above query was verified to be *true*.

- We next checked whether the concentration of activated C3G reaches a steady state as experimentally observed. The corresponding formula is:

$$((C3G^*, I_0) > 0.8) \wedge \boldsymbol{F}(\boldsymbol{G}((C3G^*, I_4) > 0.8))$$

  It was verified to be *true*.

5.3.2 The segmentation clock network We checked the oscillatory behavior of various species. Following Donaldson and Gilbert (2008), we formulated the property for the oscillatory behavior of Axin as:

$$\boldsymbol{F}(((\mathrm{Axin}, I_0) > 0.6) \wedge \boldsymbol{F}(((\mathrm{Axin}, I_2) > 0.6) \wedge \boldsymbol{F}(((\mathrm{Axin}, I_0) > 0.6) \wedge \boldsymbol{F}(((\mathrm{Axin}, I_2) > 0.6) \wedge \boldsymbol{F}((\mathrm{Axin}, I_0) > 0.6)))))$$

The property specifies the number of peaks and troughs to be expected in an oscillation cycle within the given time bound of the system. Specifically, it says that initially (with a high probability) the system is at the discretized interval 0 followed by a state some time in future where (with a high probability) the system moves to a higher discretized interval and then falls back to initial levels and so on. This query was verified to be *true*.

5.3.3 The thrombin-dependent MLC phosphorylation pathway The following are some of the formulas considered for this model:

- The profile of activated Rho starts at a very low level, reaches a high value after which the concentration drops back to the initial level. The corresponding formula was:

$$((\mathrm{Rho}^*, I_0) > 0.8) \wedge \boldsymbol{F}(((\mathrm{Rho}^*, I_4) > 0.8) \wedge \boldsymbol{F}((\mathrm{Rho}^*, I_0) > 0.8)))$$

  It was verified to be *true*.

- Rho gets activated and reaches its peak earlier than MLC:

$$((\mathrm{MLC}^*, I_4) < 0.1) \boldsymbol{U}(((\mathrm{Rho}^*, I_4) > 0.8) \wedge \boldsymbol{O}(\boldsymbol{F}((\mathrm{MLC}^*, I_4) > 0.7)))$$

  This was also verified to be *true*.

- Experimental observations suggest that the concentration of phosphorylated MLC starts at a low level, reaches a high-steady-state value. The PBL formula used to capture this property was:

$$((\mathrm{MLC}^*, I_0) > 0.7) \wedge \boldsymbol{F}(\boldsymbol{G}((\mathrm{MLC}^*, I_4) > 0.7))$$

  It was verified to be *false*.

- We then formulated a PBL formula to describe the behavior where the concentration starts with a low value, reaches a high value (peak) after which it drops back to the initial level.

$$((\mathrm{MLC}^*, I_0) > 0.7) \wedge \boldsymbol{F}(((\mathrm{MLC}^*, I_4) > 0.7) \wedge \boldsymbol{F}((\mathrm{MLC}^*, I_0) > 0.7))$$

  This formula evaluated to be *true*. This means the current ODE model is unable to explain the experimental data available for this pathway. Further investigation to identify the missing links of the pathway may be required.

6 CONCLUSION

Approximating the ODE-based biopathway dynamics as a DBN allows model analysis tasks such as parameter estimation, sensitivity analysis and probabilistic verification to be efficiently carried out. In this article, we have presented a GPU-based implementation for constructing the DBN approximations.

The significant read-sharing in the algorithm will prevent a naïve implementation from scaling upto large biopathways models. To overcome this, we have proposed a compilation strategy in which heterogeneous threads consisting of trajectory-computing threads and global memory access threads are suitably folded into warps. Further, load balancing was achieved using a simple timing model.

In our experiments, we were able to achieve significant speedup compared with a 10-CPU cluster implementation. Furthermore, our method scales well whereas the cluster-based implementation begins to consume infeasible amounts of resources for large models.

We have also formulated a simple PCTL and constructed an approximate but efficient model checking procedure. Though probability enters the picture solely via atomic propositions, one can still formulate many interesting dynamic properties of pathway models. Further, due the fact that there is a *single finite* run, the model checking procedure is particularly simple. Admittedly it is an approximate procedure. However, one can begin with our method to get a preliminary feel for the dynamics and in case a biologically

crucial and testable property shows up, one can compute its truth value more precisely by using the hybrid factored frontier (HFF) algorithm (Palaniappan *et al.*, 2011; the Supplementary Material contains more details on this).

Both contributions of this article have wider applicability. In our GPU implementation, we show how data sharing and many accesses to global memory can be tackled. In Monte Carlo-based analysis, one must stochastically generate trajectories and check whether they pass a statistical test. This will, however, entail storing a good deal of information generated by the individual trajectories and multiple passes through this information in cases where the statistical test is based on a temporal property (Donaldson and Gilbert, 2008; Fages and Rizk, 2007). In these settings our mapping techniques will lead to powerful GPU implementations. We plan to demonstrate this in our future work.

The model checking procedure we propose can be applied in other setting where DBNs arise as dynamic models (Sun and Hong, 2007). Further, the generic idea of restricting probabilities to just propositions is promising. This will enable classical model checking procedures to be combined with algorithms such as FF and HFF.

An interesting challenge will be to develop GPU-based implementation for tasks such as parameter estimation and sensitivity analysis that are computationally demanding but appear to offer opportunities for a parallel implementation. In this context, the sum-of-product algorithm implementation presented in Silberstein *et al.* (2008) promises to offer helpful pointers.

Funding: This work has been partially supported by the Singapore Ministry of Education (MoE) grant T1 251RES0917.

Conflict of Interest: none declared.

REFERENCES

Aldridge,B.B. *et al*. (2006) Physicochemical modelling of cell signalling pathways. *Nat. Cell Biol.*, **8**, 1195–1203.

Brown,K.S. *et al*. (2004) The statistical mechanics of complex signaling networks: nerve growth factor signaling. *Phys. Biol.*, **1**, 184–195.

Ciocchetta,F. and Hillston,J. (2009) PBio-PEPA: a framework for the modelling and analysis of biological systems. *Theor. Comput. Sci.*, **410**, 3065–3084.

Clarke,E.M. *et al*. (1999) *Model Checking*. MIT Press: Cambridge, USA.

Clarke,E.M. *et al*. (2008) Statistical model checking in BioLab: applications to the automated analysis of T-Cell receptor signaling pathway. In *CMSB'08*, Rostock, Germany. pp. 231–250.

Danos,V. *et al*. (2007) Rule-based modelling of cellular signalling. In *CONCUR'07*, Lisbon, Portugal. pp. 17–41.

Dematte,L. and Prandi,D. (2010)GPU computing for systems biology. *Brief. Bioinform.*, **11**, 323–333.

Donaldson,R. and Gilbert,D. (2008) A Monte Carlo model checker for probabilistic LTL with numerical constraints. *Technical Report TR-2008-282*. University of Glasgow.

Fages,F. and Rizk,A. (2007) On the analysis of numerical data time series in temporal logic. In *CMSB'07*, Edinburgh, UK. pp. 48–63.

Glaskowsky,P. (2009) *NVIDIA's Fermi: The First Complete GPU Computing Architecture*. NVIDIA. Santa Clara, USA

Goldbeter,A. and Pourquie,O. (2008) Modeling the segmentation clock as a network of coupled oscillations in the Notch, Wnt and FGF signaling pathways. *J. Theor. Biol.*, **252**, 574–585.

Gong,H. *et al*. (2010) Analysis and verification of the HMGB1 signaling pathway. *BMC Bioinform.*, **11** (Suppl. 7), 1–13.

Hansson,H. and Jonsson,B. (1994) A logic for reasoning about time and reliability. *Formal Asp. Comput.*, **6**, 512–535.

Henzinger,T.A. *et al*. (2010) Hybrid numerical solution of the chemical master equation. In *CMSB'10*, Trento, Italy. pp. 55–65.

Hindmarsh,A.C. (1983) ODEPACK, A Systematized Collection of ODE Solvers. *IMACS Trans. Sci. Comput.*, **1**, 55–64.

Klipp,E. *et al*. (2005) *Systems Biology in Practice: Concepts, Implementation and Application*. Wiley-VCH: Weinheim, Germany.

Kwiatkowska,M. *et al*. (2008) Using probabilistic model checking in systems biology. *SIGMETRICS Perform. Eval. Rev.*, **35**, 14–21.

Kwiatkowska,M. *et al*. (2010) Advances and challenges of probabilistic model checking. In *ALLER'10*, Monticello, USA. pp. 1691–1698.

Langmead,C. *et al*. (2006) Temporal logics as query languages for dynamic Bayesian networks: application to D. Melanogaster embryo development. *Technical Report CMU-CS-06-159*, Carnegie Mellon University.

Legay,A. *et al*. (2010) Statistical model checking: an overview. In *RV'10*, Malta. pp. 122–135.

Le Novere,N. *et al*. (2006) BioModels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res.*, **34**, D689–D691.

Li,C. *et al*. (2010) Online model checking approach based parameter estimation to a neuronal fate decision simulation model in Caenorhabditis elegans with hybrid functional Petri net with extension. *Mol. Biosyst.*, **11** (Suppl. 7), 1–13.

Liu,B. *et al*. (2009) Probabilistic approximations of signaling pathway dynamics. In *CMSB'09*, Bologna, Italy. pp. 251–265.

Liu,B. *et al*. (2011a) A computational and experimental study of the regulatory mechanisms of the complement system. *PLoS Comput. Biol.*, **7**, e1001059.

Liu,B. *et al*. (2011b) Probabilistic approximations of ODEs based bio-pathway dynamics. *Theor. Comput. Sci.*, **412**, 2188–2206.

Lüdtke,N. *et al*. (2008) Information-theoretic sensitivity analysis: a general method for credit assignment in complex networks. *J. R. Soc. Interf.*, **5**, 223–235.

Maeda,A. *et al*. (2006) Ca^{2+}-independent phospholipase A2-dependent sustained Rho-kinase activation exhibits all-or-none response. *Genes Cells*, **11**, 1071–1083.

Matsumoto,M. and Nishimura,T. (1998) Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, **8**, 3–30.

Murphy,K.P. (2002) Dynamic Bayesian networks: representation, inference and learning. PhD Thesis, University of California, Berkeley.

Murphy,K.P. and Weiss,Y. (2001) The factored frontier algorithm for approximate inference in DBNs. In *UAI'01*, Seattle, USA. pp. 378–385.

Palaniappan,S.K. *et al*. (2011) A hybrid factored frontier algorithm for dynamic Bayesian network models of biopathways. In *CMSB'11*, Paris, France. pp. 35–44.

Pnueli,A. (1977) The temporal logic of programs. In *FOCS'77*, Providence, Rhode Island, USA. pp. 46–57.

Silberstein,M. *et al*. (2008) Efficient computation of sum-products on GPUs through software-managed cache. In *ICS'08*, Cairo, Egypt. pp. 309–318.

Sreenath,S.N. *et al*. (2008) Modelling the dynamics of signalling pathways. *Essays Biochem.*, **45**, 1–28.

Sun,X. and Hong,P. (2007) Computational modeling of Caenorhabditis elegans vulval induction. *Bioinformatics*, **13**, i499–i507.

Zhou,Y. *et al*. (2011) GPU accelerated biochemical network simulation. *Bioinformatics*, **27**, 874–876.