Taming the complexity of biological pathways through parallel computing

Paolo Ballarini, Rosita Guido, Tommaso Mazza and Davide Prandi

Submitted: 8th November 2008; Received (in revised form): 2nd March 2009

Abstract

Biological systems are characterised by a large number of interacting entities whose dynamics is described by a number of reaction equations. Mathematical methods for modelling biological systems are mostly based on a centralised solution approach: the modelled system is described as a whole and the solution technique, normally the integration of a system of ordinary differential equations (ODEs) or the simulation of a stochastic model, is commonly computed in a centralised fashion. In recent times, research efforts moved towards the definition of parallel/distributed algorithms as a means to tackle the complexity of biological models analysis. In this article, we present a survey on the progresses of such parallelisation efforts describing the most promising results so far obtained.

Keywords: ODE numerical solutions; stochastic simulation; model checking; parallel computing; biological pathways

INTRODUCTION

The technological progresses in biology are producing a large amount of experimental data allowing scientists to adopt a systematic study of the complex interactions that characterise biological systems [1]. Regulatory pathways for gene expression, intracellular metabolic pathways and intra/inter-cellular communication pathways are examples of typical interactions among the basic elements of biological systems. Even though the large availability of high-throughput tools results in an accurate specification of the fundamental components of living systems, yet a comprehensive knowledge on how these individual components are related to each other and how interactions give rise to functions and behaviours is still missing. Predicting the effects of interactions relying on intuition only is becoming unfeasible as the complexity of the considered systems grows. In such context, mathematical modelling becomes a necessary means to unambiguously represent the information about the behaviour of the considered system and to enable sophisticated predictive analysis. Although advances in formal modelling are certainly to be seen as a positive fact, they also give rise to new challenges: the computational power required to simulate and analyse complex models is huge.

Parallel computing [2] is arguably the most popular approach for tackling the computational requirements of complex applications. Simply speaking, parallel computing is a computational paradigm by means of which many instructions are executed concurrently (or in parallel). In this article, we consider how the idea of parallel computation is exploited to manage the complexity of model analysis of living systems. The available techniques and the performance

Corresponding author. Paolo Ballarini, The Microsoft Research - University of Trento Centre for Computational and Systems Biology, Piazza Manci 17 38100 Povo, Trento, Italy. Tel: +39-0461-882841; Fax: +39-0461-882814; E-mail: ballarini@cosbi.eu

Paolo Ballarini is a researcher of the CoSbi centre. His research interests are in formal techniques for systems’ verification/specification, probabilistic model-checking, stochastic Petri nets and temporal logic reasoning. Earlier, he held a position of temporary lecturer at the University of Liverpool and a researcher at University of Glasgow.

Rosita Guido is a researcher at the University of Calabria. Her research interests are in optimisation modelling and methods (particularly applied to medical decision-making problems), support vector machines and machine learning and model selection.

Tommaso Mazza is a researcher at the CoSbi centre. His research interests are in parallel and distributed computing, topology analysis of bio-networks, bio-data handling and storing and standard languages for biology. Earlier, he held positions at the University of Catanzaro.

Davide Prandi is a researcher at the CoSbi centre. His research interests are in complexity reduction of biological systems, process algebras, mathematics and logics. Earlier, he held positions at the University of Catanzaro.
improvement that can be achieved are strictly related to the kind of model used, namely the mathematical representation of the considered system. Biological models may be distinguished according to their nature into deterministic continuous-state models, in the form of systems of differential equations, and discrete-state models, in the form of a Markov chain (i.e. stochastic model), or of a transition system (i.e. qualitative model). We organise our discussion according to three distinct subjects: the design of numerical methods for differential equations, the development of stochastic simulation algorithms and finally the application of model-checking verification to systems biology.

The organisation of the article reflects this classification: in the first section, we give an overview on the parallel computational paradigm. The following three sections cover three different approaches to model and analyse the biological systems: first, we introduce continuous deterministic mathematical models based on differential equations surveying both sequential and parallel methods for their analysis. Second, we consider discrete-state stochastic models and we describe the up-to-date advances in stochastic simulation algorithms, again first referring to sequential ones and then to parallelisation attempts. Finally, we describe the model-checking verification approach, starting from its origins in the computer science community and analysing its most recent applications to life sciences. We illustrate the essence of model-checking algorithms for different types of models providing an overview about up-to-date parallelisation efforts. Concluding remarks are presented in the final section.

AN OVERVIEW ON PARALLEL COMPUTATION

The main concern of this article is to present in a simple and concise manner different attempts to parallelise the analysis of biological pathways. To set a common language, we briefly provide some background about parallel computing.

Two main aspects characterise parallel systems: communication speed and scalability. On one hand, the speed at which communication among processing units takes place is a key factor in determining the proportion between the data exchanged and the workload of processing units; on the other hand, scalability, namely the possibility to increase the number of processing units, influences the order of magnitude of the problems that can be handled. Therefore, choosing the right architecture requires a careful evaluation of the trade-off between communication speed and scalability. As a consequence, (parallelised) applications are often classified according to how often their subtasks need to communicate with each other. An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; it exhibits coarse-grained parallelism if subtasks do not communicate many times per second and it is embarrassingly parallel if subtasks rarely or never have to communicate.

The actual trend in processors development ranges from single to multi-core processors. A multi-core processor merges many processing units into a single integrated circuit. Since messages among processing units travel on-chip, multi-core processors are particularly suited for applications that exhibit fine-grained parallelism. However, multi-core architecture's ability to scale up is inherently bounded by the technology. The wide diffusion and the relative low cost of multi-core architectures are promising, but most of the current software does not take any advantage of multi-core architectures potential as special programming techniques are required. Classic parallel programming techniques and library, such as Message Passing Interface (MPI), can be used on multi-core platforms, but more tailored approaches are needed. For instance, Intel Threading Building Blocks [3] is a C++ template library that avoids complications coming from the synchronisation of processes, which are typical in MPI-like environments. Another interesting solution is Cilk++ [4], a platform that allows legacy C++ code to be ‘multi-core enabled’ by embedding reserved keywords into the program source.

Computer clusters (or simply clusters) are a type of architecture for parallel computing where many independent processing units are connected to each other through fast and dedicated network hardware. Communication speed among processing units is slower than in multi-core architectures, making clusters more suitable for coarse-grained parallelism applications. However, clusters scale quite well and it is possible to go from small computing systems with hundreds of processing units to supercomputers with several thousands of processors (in the most recent Top500 rank of supercomputers, 410 were computer clusters). MPI is the de facto standard for
programming cluster computers. Essentially, MPI is a communication protocol for processes running on a distributed system, and it is implemented through dedicated routines in most diffused programming language.

Due to their prohibitive cost, clusters are often confined to large academic institutions and industry research laboratories. Favoured by the wide diffusion of the internet and the increasing power of home computing, a new form of distributed computing, known as GRID computing, has been developed in recent times. The term GRID refers to a network of loosely coupled, heterogeneous and world-wide distributed computers. Essentially, a GRID is nothing but a cluster where the processing units are connected through internet. As such a GRID may, potentially, scale to the entirety of computers connected to the internet. Alas, the heterogeneity of GRID nodes and the unreliability of their connections make GRID systems difficult to program and to maintain. Applications that show an embarrassing parallelism can have impressive benefits from GRID computing. An example is given by the project Folding@home [5] that allows accurate simulation of protein folding using hundreds of thousands of personal computers. Developing GRID application may be fostered by software tools such as the Berkeley Open Infrastructure for Network Computing [6], a platform for the development of distributed applications in many different areas, such as mathematics and biology.

The classes of parallel architectures presented so far are, in a certain sense, theoretical. For instance, a GRID system may be composed of connected clusters of multi-core processors. In this respect, it is worth stressing that the choice of an adequate parallel architecture strongly depends on the type of application it is referred to. Hence, for example, it is not productive to invest large budgets on a cluster, if the target application shows a fine-grained parallelism.

An aspect of parallel computing that is closely linked to the choice of an adequate parallel architecture concerns the parallelisation of sequential computations. When a computational solution to a given problem, being it, for example, a simulation algorithm or a numerical method for the solution of a system of differential equations is introduced, it is usually devised in a sequential fashion, which is: its formulation is normally readily implementable on centralised computer architecture. However, in order to exploit the computational efficiency provided by parallel architectures, standard centralised algorithms have to be turned into parallel ones. Parallelisation of sequential algorithms, essentially, involves determining a ‘suitable’ partition of the computation so that the calculation of the output result can be distributed over a parallel architecture. As a consequence, the parallelisation step helps in determining the kind of parallelism that characterises the problem, thus in choosing the best architecture. Algorithms for determining a partition of a given computational problem are usually referred to as clustering algorithms or graph partitioning algorithms.

They operate on the so-called workload graph (WG), a graph that represents the dependencies between computational units and/or input data. In a WG, nodes and arcs denote computational units and communications, respectively. Partitioning of the WG is based on the following simple criteria: computation should be distributed uniformly (i.e. clusters should roughly contain an equal amount of nodes) and, furthermore, communication between clusters (i.e. the number of cutting edges) should be minimised. A whole plethora of solutions exists in the literature about clustering methods. Here, we limit ourselves to sketching the basic principles behind two most important families of such algorithms, namely geometric algorithms and structural algorithms. Solutions in both families are based on bisection according to which a partition is determined by recursive application of a division procedure that splits the original WG into two disjoint sub-graphs. Geometric algorithms require geometric coordinates, which are used to calculate the bisection (i.e. coordinate bisection and variations). Furthermore, geometric algorithms rely on the assumption that (nodes) connectivity is equivalent to (geometric) proximity, an assumption which is not always reasonable. Structural algorithms, on the other hand, determine a bisection of the WG based exclusively on the graph’s connectivity information. With level-structure bisection, the simplest form of structural bisection, two nodes of near-maximal distance is found, and a bisection is obtained through a breadth-first traversing that starting from one such node reaches as many as half the vertices of the graph: these will be the elements of one part of the bisection, the remainder of the other. Several variants of this simple structural
CONTINUOUS AND DETERMINISTIC MODELS

Ordinary differential equations (ODEs) are likely the most used formalism in science to model dynamic systems. In pathway modelling, molecules are modelled as time-dependent variables representing concentrations. Interactions are therefore interpreted as differential relations between variables. In particular, a reaction-rate equation expresses the rate of production or consumption of a component of the system as a function of the time and of the concentrations of the other components. The general form of a system of rate equations is:

$$\frac{dx_i}{dt} = f_i(t, x(t)), \quad 1 \leq i \leq n,$$

where $x = [x_1, \ldots, x_n] \geq 0$ is the vector of the concentrations of the components of the system, $t \in \mathbb{R}^+$ represents time and $f_i : \mathbb{R}^n \to \mathbb{R}$ is usually a non-linear function. The fundamental hypothesis underlying differential equation modelling is that concentrations of substances change continuously and deterministically. Due to the non-linearity of $f_i$, finding an analytical solution of a system of reaction rate equations is not in general possible. The common way to work around analytical intractability is to resort to numerical techniques.

Sequential methods for ODE solution

Numerical analysis is a field of mathematics that includes studying algorithms for the approximation of the solution of ODE [8]. Finding an approximate solution of an ODE can be formulated as the initial value problem (IVP) or as the boundary value problem (BVP).

An IVP consists of a system of ODEs of the form

$$x'(t) = \frac{dx_i}{dt} = f_i(t, x(t)) \tag{1}$$

And an initial point $x_0$ is called initial condition. A solution of an IVP is a system of function $x(t)$ that is a solution of the system of differential equations and satisfies the initial condition, i.e. $x(t_0) = x_0$. A BVP generalises an IVP by introducing a set of constraints on the solution of the given ODE. There are more than two centuries of researches on numerical analysis of ODE and the space of this article is not enough to cite all the papers written on this topic. Therefore here, we give only a taste of the available methods in order to help a non-expert to understand parallelisation techniques described below. We refer the reader to [9] for a deeper treatment.

In 1768, Leonhard Euler described an explicit method for solving first-order ODEs, named finite differences method. A differential equation can be approximated as a finite difference

$$x'(t) = f(t, x) \approx \frac{x(t + h) - x(t)}{h}, \tag{2}$$

where $h$ is called step size. Equation (2) is then rearranged as $x(t + h) \approx x(t) + hf(t, x)$, and, starting from the initial point $x_0$ and the initial time $t_0$, a polygonal curve $(x_0, t_0), (x_1, t_1), \ldots, (x_n, t_n)$ is computed as

$$t_{n+1} = t_n + h \quad x_{n+1} = x_n + hf(t_n, x_n).$$

Taking from Euler method, many techniques have been introduced that improve the precision of the solution. There are two main families: multistep methods compute the value of $x_{n+1}$ as a function of several (past) values rather than a single one; Runge-Kutta methods, instead, use more points in the interval $(t_n, t_{n+1})$ to improve the quality of the solution.

Parallel methods for ODE solution

Since the middle of the 1980s, there have been significant efforts in designing efficient numerical techniques for ODE solution exploiting parallel and distributed architectures. In the following, we sketch the possible techniques together with pointers to some popular software tools that implement them.

The numerical solution of ODEs involves the use of linear algebra tools, as this boils down to performing matrix–matrix/matrix–vector calculations. Performance improvements can be achieved by applying parallel linear algebra techniques [10] and using available software libraries as in [11–14]. A more tailored approach to improve the performance of numerical methods for ODEs is to redesign or modify a sequential algorithm in order to exploit a specific target parallel architecture. The type of approach adopted deeply influences the performance improvement that can be achieved.

Parallelism across the method regards the use of parallel architectures to increment the strength and the efficiency of existing sequential algorithm. These kinds of methods are particularly used within the
class of Runge–Kutta methods, and have a simple implementation on a parallel machine. A limit is given by the large data exchanged with respect to the workload per processor, and therefore this type of parallelism can capitalise the recent spread of cheap multi-core systems. Clearly, this approach allows only small-scale parallelism, and in general, it is used to obtain the same performance of sequential methods but at stringent tolerances. Massive parallelism, in which a large number of processing units are available, requires different approaches. Parallelism across the system involves the decomposition of the domain of a problem into simpler sub-domains that can be solved independently. This approach requires sophisticated techniques and it is not always applicable. The general idea is to decompose an IVP into sub-problems that can be solved with different methods and different step-size strategies. Waveform relaxation is a well-known class of decomposition techniques, where a continuous problem is split and the corresponding interactions a la Picard is defined. These methodologies require a stringent synchronisation of the computations in order to assure the consistency of the results. A different method exploits parallelism by performing concurrently several integration steps with a given interaction method, leading to the class of techniques of parallelism across the steps. These techniques may be theoretically employed with a large number of processors, but intrinsic poor convergence behaviour could lead to robustness problems. However, this parallelisation method is receiving great attention because of its potential in scaling up the size of the problem that can be managed (e.g., [15]).

These are the main approaches to the parallelisation of numerical methods for ODE. For a deeper introduction, we refer to the good monograph [8] and to the special issue [16]. As in the case of parallel linear algebra, many libraries that can be included in general purpose softwares have been developed. Without any foolish ambition of completeness, we could cite the SUNDIALS [17], ParSODES [18] and ParalleloGAM [19]. These libraries are then used within complex tools such as, for example, the Systems Biology Workbench [20], a tool which offers design, simulation and analysis instruments. Behind the use of libraries within specific simulation and analysis tools, new research lines specifically tailored on biological pathways deserve a separate discussion.

ReCSiP [21] is a field-programmable gate array (FPGA)-based ODE solver explicitly designed for simulating biochemical pathways. An ODE model is translated by a software front-end into a circuit on the FPGA. The on-chip solver computes concentration of substances for each time step by integrating rate law functions. Often biologists use cluster computers to launch many simulations of the same model with different parameters at the same time. ReCSiP is particularly suited for this kind of job offering a speed about 10- to 200-fold compared with modern microprocessors, and it is cheaper than the cluster solution. General purpose scientific computing on graphics processing units (GPUs) [22] is receiving great attention since the performance of a small cluster can be achieved at a cost that does not reach $1000. Both linear algebra applications and ODE solvers are actively studied, but actually specific works on pathway-related problems are not available. A new and fruitful research line can be opened. Another interesting proposal is to parallelise algorithms that are specific to the analysis of biological pathways, as opposed to general ODE methods. For instance, extreme pathways [23] are an algorithm for the analysis of metabolic networks. A solution of the IVP describes a particular metabolic phenotype, while extreme pathways analysis aims at finding the cone of solutions corresponding to the theoretical capabilities of a metabolic genotype. Extreme pathways algorithm shows combinatorial complexity, but its parallel version [24] exhibits super-linear scalability, which means that the execution time decreases faster than the rate of increase in the number of processors.

**DISCRETE AND STOCHASTIC MODELS**

Some authors, e.g., [25], argue that the evolution of a biological system is not continuous since the number of components available can change only by an integer value. Furthermore, the evolution is a stochastic process rather than a deterministic one. Stochastic fluctuations in the evolution of biological systems become particularly relevant when a small number of molecules are available, as in the domain of genetic regulatory systems [26]. In the stochastic realm, the amount of a molecule $i$ at time $t$ is specified by a discrete random variable $X_i(t)$, and a system is expressed by a vector of random variables $\mathbf{X}$. The evolution is then modelled by a joint
probability distribution $P(X, t)$, expressing the probability that at time $t$ there are $X_i$ molecules of the first species, $X_2$ molecules of the second species and so on. A common form for $P(X, t)$ is the chemical master equation (CME) [27]

$$P(X, t + \Delta t) = P(X, t) \left(1 - \sum_{j=1}^{m} \alpha_j \Delta t\right) + \sum_{k=1}^{m} \beta_k \Delta t,$$

where $m$ is the number of reactions that can be fired in the system, $\alpha_j \Delta t$ is the probability that reaction $j$ will occur in interval $[t, t + \Delta t]$ given that the system is $X$ at time $t$, and $\beta_k \Delta t$ is the probability that a reaction $j$ will bring the system in state $X$ (from another state) within $[t, t + \Delta t]$. Unfortunately, it is not often possible to solve the CME, and therefore other tools are needed.

### Sequential stochastic simulation

Stochastic simulation algorithms are computer programs that generate a trajectory (i.e. a possible solution) of a stochastic equation. The approach was pioneered nearly 30 years ago by Gillespie [25] with his Stochastic Simulation Algorithm (SSA).

The SSA is referred to biochemical systems consisting of a well-stirred mix of molecular species that chemically interact, through so-called reaction channels, inside some fixed volume and at a constant temperature. Based on the CME, a propensity function is defined for each reaction $j$, giving the probability that a reaction $j$ will occur in the next infinitesimal interval. Then, relying on standard Monte Carlo method reactions are stochastically selected and executed forming in that a way a (simulated) trajectory in the discrete state-space corresponding to the CME. Several variants of the SSA exist, but all of them are based on the following common template:

(i) Initialise the data structures of the system;
(ii) randomly select a reaction;
(iii) execute the selected reaction;
(iv) update the data structures; and
(v) go to Step 2 or Terminate.

The different instances of the SSA vary in how the next reaction is selected and in the data structures used to store chemical species and reactions. In particular, the Next Reaction Method [28] is based on the so-called dependency graph, a direct graph whose nodes represent reactions, and whose arcs denote dependencies between reactions (i.e. an arc between reactions $i$ and $j$ exists iff the execution of reaction $i$ changes the propensity function of reaction $j$). Another research direction aims at integrating SSA with spatial information [29]. The Next-Subvolume Method (NSM) [30] simulates both reaction events and diffusion of molecules within a given volume; the algorithm partitions the volume into cubical sub-volumes that represent well-stirred systems. Steps 2 and 3 of the template algorithm above are modified by considering usual reactions as well as diffusion among sub-volumes, characterised by a diffusion coefficient that gives a measure of the ‘speed’ of the diffusion process [31].

### Parallel stochastic simulation

Stochastic simulation approaches, such as the one implemented by the SSA, are not new in systems biology; however only in relatively recent times they have received much attention, as an increasing number of studies revealed a fundamental characteristic of many biological systems. It has been observed, in fact, that most key reactant molecules are present only in a small amount in living systems (e.g. [32]), making the stochastic approach sound in life-sciences. This renewed attention also showed the main limit of SSA: it does not scale well with large systems. The resource requirements of SSA could be reduced either by using approximate algorithms or through parallelisation. The latter research line is a really recent one, but some interesting proposals are emerging.

A first improvement can be achieved by considering that many independent running of SSA are needed to compute statistics about a discrete and stochastic model. It is straightforward to run different simulations on different processes, but much attention has to be paid to the generation of random numbers [33]. This kind of parallelism is called parallelism across the simulation. The use of GRID architectures to run many independent simulations is promising because of inherent scalability [34].

Parallelism across the simulation is an effective technique when many simulations are needed, but there are instances where a single simulation of a large system (think for example to a colony of cells) is required. In this case, the researches are only at the very beginning. Basically, there are two approaches to distribute computation to the processing units: one is based on weighted graph partitioning and the other on geometric partitioning.
The Distributed-based Stochastic Simulation Algorithm or DSSA [35] is developed on the intuition that the main computational requirement of any SSA variants comes from Steps 2 and 3, namely the random selection and the execution of the next reaction. The DSSA relies on cluster architecture to tackle the complexity of these steps. In particular, a cluster processing unit, termed server, coordinates the activities of the other processing units (or clients) in the cluster, resulting in the following workflow:

(i) the server partitions the set of reactions and distributes them among a number of client processing units;
(ii) each client then performs locally a random selection and communicates it to server that determines the next reaction to the execute; and
(iii) the server communicates to the clients the information to update the local information of the clients.

The partitioning algorithm employed in Step 1 uses the dependency graph as a weighted graph to minimise communications between the server and the clients; in particular, not all the clients need to be updated after a reaction is selected by the server. The authors outline some experimental and performance analysis showing that the performance improvement with respect to SSA is linearly dependent on the number of client nodes.

Another approach that is receiving great attention is based on geometric clustering. A pioneer work is [36], but the algorithm reached a good maturation only with the recent efforts in integrating SSA with space information. In particular, in [37] the NSM is parallelised by using a geometric clustering algorithm to map set of sub-volumes to processing units. The algorithm scales well on a cluster architecture, where the main limit is the linear relation between the diffusion coefficient and the number of messages exchanged among the processing units. The authors also test a promising GRID version of the algorithm, but the overhead due to the synchronisation among processing units requires more investigations.

Finally, we refer a couple of applications of non-standard parallel hardware to speed up stochastic simulation [38]. In [39], an FPGA is configured so to perform a reaction event of SSA at every clock cycle. By comparison, a usual CPU requires thousand of clock cycles for a single SSA reaction. Another work [40] exploits the high parallel structure of modern GPUs to obtain parallelism across the simulation without the costs of a computer cluster. These applications are really promising, but they require on the road testing.

MODEL CHECKING

In the late 1970s/early 1980s, a novel approach to the formal verification of discrete-state models appeared in the computer science community under the name of model-checking [41, 42]. It was initially targeted to the verification of computer hardware designs, but it soon spread to several areas such as software verification, communication protocols verification, reliability analysis, game theory and, in recent years, it has been applied to life sciences as well. Model checking is based on a fairly simple principle: given a model \( M \) and a property \( \varphi \), an algorithm is defined that verifies whether \( \varphi \) is a property of \( M \) (denoted \( M \models \varphi \)). What makes model checking so appealing is that the verification procedure is automatic and the produced result is exact (up to the exactness with which the model represents the behaviour of the considered system) as it is obtained through an exhaustive search of the state space of the model. Model-checking approaches can be classified according to the type of model considered and the associated formal language used for formulating properties. Linear temporal logic (LTL) and computational tree logic (CTL) are languages used to state properties of qualitative models. Examples of qualitative properties are safety properties (‘something bad is not going to happen’), liveness properties (‘something good is eventually going to happen’) and fairness properties (‘in a competing situation every process is guaranteed to get its turn’). CTL model checking has been considered for the verification of properties of biological systems [43]. It should be noted that application of model-checking verification to a continuous and deterministic model of a biological system entails a discretisation procedure by means of which the original model is turned into a discrete-state one. For instance, the BIOCHAM tool [44] supports CTL model checking on the qualitative model obtained from a discretisation of a system which is originally expressed as a set of reaction-rate equations (the discretisation being obtained by conversion of molecules concentrations into...
 amounts and by disregarding kinetic rates of the chemical equations). The Genetic Network Analyzer (GNA) [43] supports CTL verification for models obtained through abstraction of a system of piecewise linear differential equations (PLDEs): the phase-space corresponding to the considered system of PLDEs is split into a number of subspaces corresponding to specific combinations of the sign of the derivatives in the system of PLDEs. CTL formulae are then used to express patterns characterising relevant trends for the modelled species (e.g. 'species X steadily grows until a threshold has been reached and then species Y starts decreasing').

In the 1990s, the idea of model checking was extended to quantitative models. The probabilistic CTL (PCTL) [45] is a language to express properties referred to Discrete Time Markov Chain models, whereas its dual, the continuous stochastic logic (CSL) [46, 47], refers to continuous time Markov chain models. Markov chain models can be thought of as state graphs with stochastic information attached to the arcs. Verification of properties against a Markov chain model is quantitative in two respects: because the output of a probabilistic model checker is a measure of the probability that the model exhibits a given property (rather than a yes/no answer) and also because time boundaries can be associated to properties to indicate that the behaviour of interest has to happen within a time frame. Verification of PCTL/CSL formulae also entails the application of numerical methods to the solution of the considered Markov chain model (i.e. computation of the steady-state distribution and transient-state distribution [48]). Recently, effort has been put in the application of probabilistic model checking to the verification of relevant biological case studies (e.g. [49, 50]), as well as in terms of tool support [51].

The main issue with the model-checking approach is given by the explosion of the model's dimension. The number of states in a model can easily reach a level which goes well beyond the storage capability of currently available computational resources. This is even truer with modelling of biology where systems often consist of complex networks of signals and large populations and, as a consequence, model-checking verification, in many cases, is simply not applicable. Several techniques aimed to tackle the state-space explosion problem have been developed over the last decades. On-the-fly, verification algorithms [52, 53] establish the truth of a qualitative property by progressively exploring/building the state space (an approach that in most cases avoids the construction of the whole state space). Partial order reduction techniques allow for a reduction of the state space dimension based on exploitation of the commutativity of concurrently executed transitions [54]. Binary decision diagrams (BDDs) and multi-terminal BDDs (MTBDD) provide a form of memory-efficient encoding of a model state-space, for non-probabilistic, respectively, probabilistic models. They have been extensively studied yielding so-called symbolic model-checking algorithms (most popular model-checking tools, such as nuSMV [55] and PRISM [56], to mention a couple, are based on symbolic approaches). Despite the advances given by techniques for the efficient representation of large models, the state space explosion remains a major limiting factor to the application of model-checking verification to complex systems. A promising path of research is that of parallel model-checking approaches.

Parallel model checking

Classical model-checking algorithms establish the truth of a formula through exploration of the model's state space (a process often referred to as reachability analysis). Parallelisation of such algorithms is about looking at ways for distributing the reachability-based verification of a formula over a multi-processor architecture. In practice, however, model-checking algorithms differ with respect to the type of logic they are referred to, thus the parallelisation problem is different with respect to different type of model checking. With LTL model checking, an automata-based type of verification, checking the truth of a formula corresponds to checking the emptiness of the language recognised by a specific (Büchi) automaton. This, in turn, is proved to be equivalent to finding a cycle containing an accepting state (in the graph corresponding to the automaton, see [57]). Hence LTL model checking boils down to cycle detection, a well-studied subject in graph theory. There are a plethora of algorithms for cycle detection mostly based on a depth-first search (DFS) exploration of the graph. Several approaches for the optimisation of DFS-based cycle detection have been proposed [58, 59], however DFS algorithms are known to be hard to parallelise. In [60], authors propose the so-called Negative Cycle detection (NC) algorithm, by means
of which cycle detection is reduced to detection of negative cycles on a directed graph whose edges are labelled (with real-valued lengths) in the following way: length $-1$ is given to edges outgoing from accepting states, length $0$ is given to all remaining edges. The advantage of the NC algorithm, as opposed to DFS ones, is that it is suitable for parallelisation. The DiVinE model checker [61], recently applied to the verification of genetic regulatory networks [57], incorporates a number of algorithms for parallel verification of LTL properties, which have been shown to improve the performance of popular sequential model checkers such as SPIN [53].

On the other hand, parallelisation of model checking for probabilistic models regards looking at methods for a decomposed solution of Markov chain models. It can be showed that verification of reachability properties as well as of steady-state properties against a Markovian model corresponds to solving systems of linear equations (SLEs). Distributed approaches to the solution of SLEs may be distinguished into those that operate on explicit data structures, an overview of which can be found in [62], as opposed to those that work on symbolic representations (i.e. MTBDD) of the state space [63]. The MTBDDs-based parallelisation proposed in [63] has been implemented in the probabilistic model-checker PRISM and tested on a dual-processor shared-memory architecture showing an average speed up of 1.8 for the verification of some benchmark models. To the best of our knowledge, fully parallelised approaches targeted to cluster or GRID architectures have not yet been realised to date on existing tools for probabilistic model checking.

CONCLUSIONS

In this article, we have provided an overview about the application of parallel computing paradigms to systems biology. Parallel approaches are ‘biologically motivated’ due to the enormous complexity (of the modelled systems) which the exponential improvements in molecular biology understanding has recently exposed. In fact, one of the main limitations in managing large biological systems comes from the fundamental difference between the high parallelism expressed by the biochemical reactions within a system and the sequential environments employed for the analysis and the simulations of these reactions. Such a limitation is cross-method, namely it affects both continuous/deterministic and discrete/stochastic models, and it undermines the applicability of simulation as well as of other analysis methods. It seems therefore natural to move towards parallel computation and architectures. We have organised our discussion about parallel computing with respect to three complementary subjects, namely: the solution of systems of ODEs, the development of simulation approaches to discrete stochastic models and, finally, the application of model-checking techniques to the verification of discrete-state models of biological systems.

With regards to methods for the parallel solution of differential equations, we have seen that a vast literature has been produced and a large set of well-established software libraries are available nowadays. Furthermore, the increasing availability of low-cost parallel architectures (such as multi-core) is extending the users of these tools.

On the other hand, parallelisation of stochastic simulation algorithms is still emerging as one of the key techniques to tackle down the scalability issue of most simulation algorithms. We have seen that most works in the area are still in a developing phase, but some results are promising. In that respect, the possibility of using a GRID architecture (i.e. a potentially extremely large number of parallel computing nodes) to perform several simultaneous and independent simulations increases the precision of the statistics that can be calculated through simulation. Clearly, this solution does not allow a scale up in the size of the models manageable. The two classes of parallel algorithms we discussed, one based on weighted graph partitioning, and one on geometric partitioning, go in this direction. Both proposals are promising but require on the road testing on real case studies to establish their real applicability. Notably, the use of alternative computing devices is promising and deserves more research efforts.

Finally, we considered model checking, a well-established technique for the automatic verification of a model of a system which has, in recent times, attracted the attention of the systems biology research community. We presented an historical overview of the model-checking method, distinguishing between qualitative versus quantitative verification of models. We described popular approaches to tackling the infamous state-space explosion problem which undermines the applicability of model checking to complex systems.
We described some applications of model-checking to systems biology and together with existing software frameworks targeted to modelling of biological systems which feature model-checking verification. We then considered the issue of parallelising model-checking algorithms, in the qualitative settings as well as in the quantitative parallelising model-checking algorithms, in the verification. We then considered the issue of logical systems which feature model-checking software frameworks targeted to modelling of systems biology and together with existing tools.

Key Points

- Analysis of biological pathways require great computational power.
- Parallelisation is an effective technique for tackling computational requirements.
- Efficient parallel ODEs solvers are embedded into biological tools.
- Parallel stochastic simulation is still in its infancy but promising works are emerging.
- Application of (parallel) model-checking approaches to systems biology is still an emerging discipline and is not yet well understood whether it can be tailored to the needs of biologists.

References


40. Li H, Petzold L. In silico simulation of biological network dynamics.


