cuTauLeaping: A GPU-Powered Tau-Leaping Stochastic Simulator for Massive Parallel Analyses of Biological Systems

Marco S. Nobile, Paolo Cazzaniga, Daniela Besozzi, Dario Pescini, Giancarlo Mauri

Published: March 24, 2014 • DOI: 10.1371/journal.pone.0091963

Correction

26 Jun 2014: The PLOS ONE Staff (2014) Correction: cuTauLeaping: A GPU-Powered Tau-Leaping Stochastic Simulator for Massive Parallel Analyses of Biological Systems. PLoS ONE 9(6): e101858. doi: 10.1371/journal.pone.0101858 | **View correction**

Abstract

Tau-leaping is a stochastic simulation algorithm that efficiently reconstructs the temporal evolution of biological systems, modeled according to the stochastic formulation of chemical kinetics. The analysis of dynamical properties of these systems in physiological and perturbed conditions usually requires the execution of a large number of simulations, leading to high computational costs. Since each simulation can be executed independently from the others, a massive parallelization of tau-leaping can bring to relevant reductions of the overall running time. The emerging field of General Purpose Graphic Processing Units (GPGPU) provides power-efficient high-performance computing at a relatively low cost. In this work we introduce cuTauLeaping, a stochastic simulator of biological systems that makes use of GPGPU computing to execute multiple parallel tau-leaping simulations, by fully exploiting the Nvidia's Fermi GPU architecture. We show how a considerable computational speedup is achieved on GPU by partitioning the execution of tau-leaping into multiple separated phases, and we describe how to avoid some implementation pitfalls related to the scarcity of memory resources on the GPU streaming multiprocessors. Our results show that cuTauLeaping largely outperforms the CPU-based tau-leaping implementation when the number of parallel simulations increases, with a break-even directly depending on the size of the biological system and on the complexity of its emergent dynamics. In particular, cuTauLeaping is exploited to investigate the probability distribution of bistable states in the Schlögl model, and to carry out a bidimensional parameter sweep analysis to study the oscillatory regimes in the Ras/cAMP/PKA pathway in *S. cerevisiae*.

Citation: Nobile MS, Cazzaniga P, Besozzi D, Pescini D, Mauri G (2014) cuTauLeaping: A GPU-Powered Tau-Leaping Stochastic Simulator for Massive Parallel Analyses of Biological Systems. PLoS ONE 9(3): e91963. doi:10.1371/journal.pone.0091963

Editor: Jean Peccoud, Virginia Tech, United States of America

Received: October 24, 2013; **Accepted:** February 17, 2014; **Published:** March 24, 2014

Funding: This work is supported by Research Infrastructure “SYSBIO Centre of Systems Biology” (http://www.sysbio.it/). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Introduction

Nowadays, the use of computational methods represents a valuable and integrative tool to conventional experimental biology, thanks to the promising capability to gain a global-level understanding of the emergent dynamics of biological systems and to elucidate the mechanisms governing their functionality, that most of the times can be hardly determined by laboratory experiments only. Indeed, the computational study of biological systems can present many advantages in terms of cost, ease to use and rapidity, and can support laboratory work by suggesting *ad hoc* designed experiments. In this context, mathematical modeling tools, simulation algorithms and analysis techniques simplify the predictions on the way these complex systems behave in normal conditions and how they react to genetic, chemical or environmental perturbations; moreover, they can facilitate the verification of specific dynamical properties, which can be characterized by non linear or multistable phenomena [1]–[4].

Given a mathematical model describing the interactions between the components of a biological system, computer algorithms allow to validate and analyze the model, giving the possibility to recreate *in silico* a wide spectrum of emergent phenomena; in particular, simulation algorithms are an essential tool to study the temporal evolution of biological systems. Anyway, the shift from the reproduction of the experimental observations to the capability of making predictions on the behavior of the system in unexplored conditions can be limited by the lack or the inaccuracy of available quantitative data (e.g., reaction rates, intracellular concentrations, etc.), which are indispensable to settle a good model parameterization. To cope with these problems, several computational methods can be exploited [1], such as parameter estimation (PE) [5]–[9], sensitivity analysis (SA) [10]–[12], parameter identifiability (PI) [13]–[15], parameter sweep analysis (PSA) [6], reverse engineering (RE) [7], [16], etc. These methods usually require the execution of many simulations to explore the high-dimensional search space of possible model parameterizations, therefore resulting in prohibitive computational costs.

An additional aspect that should be considered when defining mathematical models of biological systems is related to the experimental evidences that most of the cellular regulation networks, especially those involving few amounts of some molecular species, are affected by noise [17]. The randomness occurring at the molecular scale can induce stochastic phenomena at the macromolecular scale, giving rise to non deterministic behaviors. The classical modeling approach based on ordinary differential equations (ODEs) is not able to fully capture all the effects of stochastic processes; in this context, the most remarkable example is the phenomenon of bistability, that can be effectively investigated by means of stochastic approaches [10], [18].

Stochastic modeling of biological systems can rely on the definition of stochastic differential equations (SDEs), like the Langevin equation [19], or on the stochastic formulation of chemical kinetics [20], whereby a biological system is formalized by specifying the set of molecular species which interact through a set of chemical reactions based on mass-action kinetics. These reaction-based models can be simulated by means of Monte Carlo procedures, like Gillespie's stochastic simulation algorithm (SSA) [20], which was proven to be equivalent to the Chemical Master Equation and to generate an exact temporal evolution of well-stirred biochemical systems [21]. Since SSA proceeds by simulating the execution of a single reaction per computation step, it may require a high running time even for small systems. Many improvements to the original SSA procedure were proposed [22]–[24], but all of them still result computationally expensive; among stochastic simulation algorithms, one of the most efficient is tau-leaping [25], which outperforms SSA by allowing the execution of multiple reactions per step, thus providing a relevant reduction of the running time. In the last years, tau-leaping was extended in order to avoid the possibility of generating negative molecular amounts [26], [27], to tackle the problem of stiffness [28], or to keep into account the spatial localization of molecular species [29] and delayed reactions [30]. The present work is based on the modified tau-leaping version proposed in [27].

Despite the computational improvements brought by the tau-leaping algorithm, a typical task for the analysis of stochastic models can still be affected by high computational costs: as a matter of fact, besides requiring many different simulations to explore the space of all possible model parameterizations with PE, SA, PI or PSA analysis, the application of stochastic simulation algorithms needs a congruous number of repetitions of the simulations, under the same conditions, in order to deal with the effects of noise due to stochastic fluctuations and to obtain statistically significant results about the system behavior. Therefore, for the analysis of biological systems based on stochastic modeling approaches, an efficient strategy for the parallelization of multiple tau-leaping executions is necessary to obtain a consistent reduction of the computational costs, and to provide scientists with a powerful tool that may speed up the achievement of new insights into the functioning of biological systems.

The traditional methods to perform parallel executions of an algorithm consist in multithreading [31], distributed computing on clusters [32], [33], or custom circuitry produced with Field Programmable Gate Arrays (FPGA) [34]. The emerging area of General Purpose Graphics Processing Units (GPGPU) computing is a halfway solution that gives access to the huge unexpressed computational power of modern video cards, which reside in almost any personal computer of mid-range price. In addition, GPGPU computing is not only beneficial from a computational point of view, but it also allows to strongly reduce the energy consumption. As a matter of fact, in June 2013, two GPU-powered machines (EURORA and Aurora Tigon) reached the first places in the Green500 list of the most power-efficient supercomputers (http://www.green500.org/lists/green201306). Both machines are based on NVIDIA Tesla K20 GPU.

Previous works presented the parallelization on Graphics Processing Units (GPU) of multiple SSA simulations [35]–[37], also considering reaction-diffusion systems [38], and the fine-grain acceleration of a single tau-leaping execution [39]. The research we describe here represents the first achievement, to the best of our knowledge, in running a huge number of parallel tau-leaping simulations on the GPU for the analysis of biological systems; to this aim, we introduce the novel stochastic simulator called cuTauLeaping. We discuss, in particular, some relevant issues related to the optimization of the tau-leaping implementation on GPU, since a simple and naïve porting to the CUDA architecture of its working process turned out to be inefficient. In cuTauLeaping we introduce a novel restructuring of tau-leaping workflow – consisting in the execution of four different algorithmic phases – which better fits the GPU architecture and avoids the inefficiency drawbacks. We also present the design of *ad hoc* data structures that are necessary for an appropriate memory allocation on the GPU: this is a particularly tricky issue in GPGPU computing, since GPU memories need to be properly used to achieve good computational performances. Another drawback of GPU computing is related to the choice of efficient, parallel and statistically sound random number generators (RNGs). Like any other Monte Carlo-based algorithm, tau-leaping heavily relies on high-quality pseudorandom sequences. In this work, we discuss how to obtain good computational performances by exploiting a proper RNG among the available and out-of-the-box solutions.

In order to compare the computational costs of cuTauLeaping with respect to a standard CPU-based implementation of the original tau-leaping algorithm, we carry out different batches of simulations of four stochastic models of real biological systems: the Michaelis-Menten kinetics [40], a prokaryotic gene regulatory network [35], [41], the Schlögl model [42], and the Ras/cAMP/PKA signal transduction pathway in yeast [6], [43], [44]. In particular, we exploit cuTauLeaping to execute massive parallel simulations to investigate the probability distribution of bistable states in the Schlögl model, and to perform a fine-grain bidimensional parameter sweep analysis for the identification of oscillatory regimes in the Ras/cAMP/PKA pathway in *S. cerevisiae*. In addition, we exploit synthetic systems of increasing size, randomly generated as described in [39], to evaluate the impact of the models size on the computational performance, irrespective of any actual dynamical properties (i.e., oscillations, bistability, etc.) that the systems may present.

We show that even with a limited number of parallel simulations (ranging from 2^2 to 2^7, according to the complexity of the investigated biological system), cuTauLeaping outperforms the CPU implementation of tau-leaping with an empirical speedup ranging from $25\times$ up to $1000\times$, approximately, therefore portending its valuable application for thorough analyses of stochastic biological systems.

Methods

Stochastic modeling and simulation of chemical kinetics

According to the stochastic formulation of chemical kinetics [20], a model of a biological system W is defined by specifying the set of N molecular species $\mathcal{S}=\{S_1,\dots,S_N\}$ which interact through M chemical reactions $r_1,\dots,r_M$; W is assumed to be spatially homogeneous and in thermal equilibrium within a fixed volume V. A generic reaction r_j is defined as

$$r_j\ :\ \alpha_{j1}S_1+\ldots+\alpha_{jN}S_N\rightarrow\beta_{j1}S_1+\ldots+\beta_{jN}S_N, \tag{1}$$

where $\alpha_{ji},\beta_{ji}\in\mathbb{N}$ are the stoichiometric coefficients associated, respectively, to the i-th reactant and to the i-th product of the j-th reaction, for $i=1,\dots,N$ and $j=1,\dots,M$. Reactions $r_1,\dots,r_M$ implicitly define two matrices, $\mathbf{MA},\mathbf{MB}\in\mathbb{N}^{M\times N}$, having α_{ji} and β_{ji} as elements, respectively. We denote by $x_i(t)\in\mathbb{N}$ the number of molecules of species S_i present in W at time t, so that $\mathbf{x}=\mathbf{x}(t)\equiv(x_1(t),\dots,x_N(t))$ represents the state of the system at time t. We denote by $\mathbf{MV}$ the state change matrix associated to W, defined as $\mathbf{MV}\equiv\mathbf{MB}-\mathbf{MA}$. Each row of this matrix, $\mathbf{MV}_j\equiv(v_{j1},\dots,v_{jN})$, is called the state change vector, and consists of elements $v_{ji}=\beta_{ji}-\alpha_{ji}$, $v_{ji}\in\mathbb{Z}$, that represent the stoichiometric change of species S_i due to reaction r_j.

In addition to $\mathbf{MV}$, we define a supplementary state change matrix $\overline{\mathbf{MV}}$, where $\bar{v}_{ji}=0$ for each $S_i\in\mathcal{F}$, for a given $\mathcal{F}\subset\mathcal{S}$ that contains the molecular species whose amounts have to be kept constant during the simulation; the subset $\mathcal{F}$ is used to account for a continuous “feed” of molecules into the system. This condition can be used to mimic, for instance, the non-limiting availability of some chemical resources, or the execution of *in vitro* buffering experiments, in which an adequate supply of some species is introduced in W in order to keep their quantity constant [45].

The traditional way to calculate the stochastic temporal evolution of W consists in solving the so-called Chemical Master Equation (CME), which describes the probability distribution function associated to W [46]. Numerical solution algorithms for the CME are usually based on matrix descriptions of the discrete-state Markov process [47]; anyway, these methods are computationally expensive and not always feasible, especially for systems consisting of many molecular species, for which the number of reachable states is huge or even (countably) infinite. Several analytical solution algorithms for the CME exist, for instance those based on uniformization methods [48]–[50], finite state projection algorithms [51], [52] or the sliding window method [53]; other methods were also introduced for special reaction systems characterized by particular initial conditions (see, e.g., [54] and references therein). A different strategy to solve the CME consists in generating trajectories of the underlying Markov process. A method of this type is the stochastic simulation algorithm (SSA) [20], [55], which provides exact realizations of the associated continuous time, discrete state space jump Markov process $\mathbf{x}$ of a biochemical system $\mathcal{W}$, whose initially conditioned density function is determined by the CME itself; as such, SSA is logically equivalent to the CME [21].

Briefly, starting from the system state $\mathbf{x}$, SSA determines which reaction will be executed during the next time interval $[t,t+\tau)$, by calculating the probability of each reaction r_j to occur in the next infinitesimal time step $[t,t+dt)$. This probability is proportional to $a_j(\mathbf{x})dt$, being $a_j(\mathbf{x})=c_j\cdot d_j(\mathbf{x})$ the *propensity function* of reaction r_j, where $d_j(\mathbf{x})$ is the number of distinct combinations of the reactant molecules occurring in r_j and c_j is a stochastic constant encompassing the physical and chemical properties of r_j [55]. The time τ before a reaction takes place is chosen according to the following equation:

$$\tau=\frac{1}{a_0(\mathbf{x})}\ln\left(\frac{1}{\rho_1}\right),$$

where ρ_1 is a random value sampled in [0,1] with a uniform probability, and $a_0(\mathbf{x})=\sum_{j=1}^{M}a_j(\mathbf{x})$. The index j of the reaction to be executed is the smallest integer in $[1,M]$ such that

$$\sum_{j'=1}^{j}a_{j'}(\mathbf{x})>\rho_2\cdot a_0(\mathbf{x}),$$

where ρ_2 is a random value sampled in [0,1] with a uniform probability.

In [25] an approximate but faster version of SSA, called tau-leaping, was introduced for the purpose of reducing the computational burden typical of SSA. SSA and tau-leaping share the characteristic that, even starting from the same initial state of the system, repeated executions of the algorithms will produce (usually quantitative, but potentially also qualitative) different temporal dynamics, thus reflecting the inherent noise of the system. These two algorithms, anyway, differ with respect to the way reactions are applied at each step: in SSA, only *one* reaction is applied, while with tau-leaping *several* reactions can be applied.

Tau-leaping algorithm

We present here the main features of tau-leaping, that are beneficial to illustrate the choices at the basis of the GPU implementation proposed in this work. We refer to [25], [56] for further details and, especially, to [27], which describes the improved version of the tau-leaping algorithm considered here.

Given a state $\mathbf{x}$ of the system $\mathcal{W}$, let $K_j(\tau,\mathbf{x},t)$ denote the exact number of times that a reaction r_j would be fired in the time interval $[t,t+\tau)$; $\mathbf{K}(\tau,\mathbf{x},t)$ denotes the probability distribution vector having $K_j(\tau,\mathbf{x},t)$ as elements. For arbitrary values of τ, the computation of the values $K_j(\tau,\mathbf{x},t)$ can be as difficult as solving the corresponding CME. On the contrary, if τ is small enough so that the change in the state $\mathbf{x}$ during $[t,t+\tau)$ is so slight that no propensity function will suffer an appreciable change in its value (this is called the *leap condition*), then it is possible to evaluate a good approximation of $K_j(\tau,\mathbf{x},t)$ by using the Poisson random variables with mean and variance $a_j(\mathbf{x})\tau$. So doing, the stochastic temporal evolution of the system is no longer exact (as in the case of SSA); however, the accuracy of tau-leaping can be fixed a priori by means of an error control parameter $\epsilon\in(0,1]$, which is involved in the computation of the changes in the propensity functions and of the time increment τ. The propensity functions change as a consequence of the modification in the molecular amounts of the reactant species, therefore the leap condition must be verified after each state update. This is achieved by evaluating an additional quantity $g_i=g_i(x_i(t))$ for each species S_i, which is related to the highest order $H(i)$ of the reactions in which S_i is involved as a reactant (see [27] for details). This information, along with the number of molecules of S_i involved in all highest-order reactions (given by the system state $\mathbf{x}$), is then used to bound the relative change of $x_i(t)$. Starting from the state $\mathbf{x}$ and choosing a τ value that satisfies the leap condition, the state of the system at time $t+\tau$ is updated according to

$$\mathbf{x}(t+\tau)=\mathbf{x}+\sum_{j=1}^{M}\mathbf{MV}_jP_j(a_j(\mathbf{x}),\tau), \tag{2}$$

where $P_j(a_j(\mathbf{x}),\tau)$ denotes an independent sample of the Poisson random variable with mean and variance equal to $a_j(\mathbf{x})\tau$.

Note that the execution of many reactions per step could lead to negative amounts of the molecular species in $\mathcal{W}$ [25]. To be more precise, if the reactions executed during a step consume a number of reactant molecules greater than those occurring in the system, then negative species amounts would be generated; therefore, the simulation step cannot be executed. To avoid these situations, some reactions are considered as *critical*: a reaction r_j is marked as critical if there are not sufficient reactant molecules to fire it at least n_c times in the next time interval. In this work we use the threshold $n_c=10$, as suggested in [27]. At each iteration of tau-leaping, all reactions are partitioned into the sets of non-critical reactions (R_{nc}) and critical reactions (R_c). Only a single reaction belonging to R_c – selected following the SSA procedure – is allowed to fire during $[t,t+\tau)$.

The length of the step τ satisfying the leap condition is calculated as

$$\tau=\min_{i\in S_{nc}}\left\{\frac{\max\{\epsilon x_i(t)/g_i,1\}}{|\mu_i(\mathbf{x})|},\frac{(\max\{\epsilon x_i(t)/g_i,1\})^2}{\sigma_i^2(\mathbf{x})}\right\}, \tag{3}$$

where S_{nc} is the set of indices of reactant species not involved in critical reactions, and the values $\mu_i(\mathbf{x})$ and $\sigma_i^2(\mathbf{x})$ are computed as follows:

$$\mu_i(\mathbf{x})=\sum_{j\in R_{nc}}v_{ji}a_j(\mathbf{x}),\quad \sigma_i^2(\mathbf{x})=\sum_{j\in R_{nc}}v_{ji}^2a_j(\mathbf{x}),$$
$$\text{for each } i=1,\ldots,N. \tag{4}$$

If the execution of a tau-leaping step would lead to negative amounts of some species, then the τ value is halved and the number of reactions to execute is sampled *ex novo*.

Finally, if τ is smaller than a multiple of $1/a_0(\mathbf{x})$ – which corresponds to the average time increment of SSA – then a certain number of SSA steps is executed because, given the actual state of the system, this will be more accurate and efficient than a tau-leaping step.

General-purpose GPU computing

The emerging field of GPGPU computing allows developers to exploit the great computational power of modern multi-core GPUs, by giving access to the underlying parallel architecture that was conceived for speeding up real-time three-dimensional computer graphics. The GPU implementation of tau-leaping that we propose in this work was developed and optimized for Nvidia's Compute Unified Device Architecture (CUDA), a cross-platform GPGPU library that combines the Single Instruction Multiple Data (SIMD) architecture and multi-threading. CUDA automatically handles the control flow divergence, that is, threads can take different execution paths in a transparent way for the programmer. Nevertheless, conditional branches should be avoided whenever possible as they cause a reduction of performances, due to the serialization of the execution until reconvergence. For this reason, the tau-leaping algorithm required a major reconstruction in order to reduce the need for conditional branches, as will be described in the Results section.

CUDA

Using CUDA's naming conventions, the programmer implements a kernel (that is, a C function) loaded from the host (the CPU) to the devices (one or more GPUs), replicated in many copies named threads. Threads can be organized in three-dimensional structures named blocks which, in turn, are contained in three-dimensional grids. Whenever the host runs a kernel, the GPU creates the corresponding grid and automatically schedules each block on one streaming multiprocessor (SM) available on the GPU, a solution that allows a transparent scaling of performances on different devices (see Figure 1, left side). CUDA poses limitations on the number of threads a block may contain: up to 1024 threads can be distributed in the three dimensions, and each dimension must not exceed 512 threads. The SM organizes scheduled blocks in batches consisting in 32 parallel threads, called warps. Since more than one block can be assigned at once to the same SM, a warp scheduler manages the execution of warps.

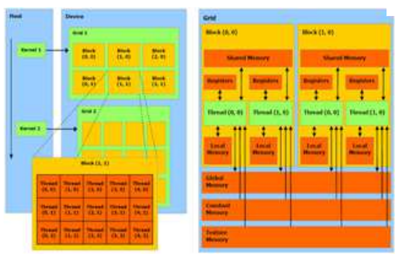


Figure 1. Schematization of CUDA architecture.
Schematic representation of CUDA threads and memory hierarchy. *Left side*. Thread organization: a single kernel is launched from the host (the CPU) and is executed in multiple threads on the device (the GPU); threads can be organized in three-dimensional structures named blocks which can be, in turn, organized in three-dimensional grids. The dimensions of blocks and grids are explicitly defined by the programmer. *Right side*. Memory hierarchy: threads can access data from many different memories with different scopes; registers and local memories are private for each thread. Shared memory let threads belonging to the same block communicate, and has low access latency. All threads can access the global memory, which suffers of high latencies but is cached since the introduction of Fermi architecture. Texture and constant memories can be read from any thread and feature a cache as well. Figures are taken from the Nvidia's CUDA programming guide [58].
doi:10.1371/journal.pone.0091963.g001

Threads can read and write data from different kinds of memories (Figure 1, right side): the global memory (visible from all threads), the shared memory (accessible from threads belonging to the same block), and the local memory (registers and arrays, accessible from the owner thread). Furthermore, all threads can read data from two cached memories: the constant memory and the texture memory. CUDA offers other types of memory, like the page-locked memory, portable memory, and mapped memory; as our implementation does not exploit these additional features, they go beyond the scope of the present paper and will not be further described here.

The global memory is generally very large (up to thousands MBs), but suffers from high access latencies, whilst the shared memory is faster but much smaller (tens of thousands KBs for each SM). Being a very small resource on each multiprocessor, the shared memory poses constraints on the blocks size, thus limiting the number of simultaneous threads that can be executed at once. However, in order to achieve the best performances, the shared memory represents a precious resource that must be exploited as much as possible. These considerations are central in our implementation of tau-leaping and will be described in more detail in the Results section.

Since the introduction of the Fermi architecture, the global memory features a small amount of cache, which makes the use of the texture memory counterproductive [57]. This cache resides on the same on-chip memory (64 KB for each SM) that is used for both cache and shared memories, and gives the programmer the opportunity to balance the two memory amounts. Our GPU implementation of tau-leaping was optimized for the Fermi architecture, since it heavily relies on the availability of shared memory for performance reasons.

Random numbers generation

Stochastic simulation algorithms exploit random numbers generators (RNGs). Nvidia's software development kit [58] contains several libraries and utilities that help developers in the process of creating software for this architecture; CURAND is a RNG library which allows the GPU-based generation of random deviates that can be used both by the host (via memory copy) or directly by the device. CURAND is the only external library that we exploited in cuTauLeaping, while the remaining code was developed in plain vanilla CUDA code.

CURAND from CUDA toolkit v5.0 [59] gives access to many different RNGs that can produce pseudo-random sequences with a very large period: XORWOW [60], MRG32K3A [61] and Mersenne Twister (MT) [62]. Among these RNGs, MT is the one that yields the longest pseudo-random sequences thanks to its 2^{11213} period, while XORWOW and MRG32K3A generate sequences of pseudo-random numbers with a period of $\approx 2^{160}$ and $\approx 2^{191}$, respectively. MT was not used for the implementation of cuTauLeaping because it has three drawbacks: (i) at most 256 threads per block can operate simultaneously [59], (ii) the memory footprint is larger than the other generators [63], (iii) it is much slower than the other two algorithms.

XORWOW is faster than MRG32K3A, but it is known to present statistical flaws [64] and it is rejected by 3 of the 106 tests of the BigCrush statistical test suite [65]. CURAND exploits these RNGs to generate random deviates with uniform or standard normal distributions; since the introduction of the CUDA toolkit v5.0, CURAND libraries offer the possibility to generate the Poisson-distributed random deviates required by tau-leaping. cuTauLeaping offers the possibility to perform the simulations using both RNGs; the results presented in what follows are based on MRG32K3A.

Results

In this section we describe the development of cuTauLeaping and its application to perform parallel stochastic simulations in a massively parallel way, by running multiple independent simulations as parallel CUDA threads. We introduce our GPU-oriented design of tau-leaping, consisting in a four phases workflow, and present the data structures, the memory allocation strategies and the advanced functions exploited on the Fermi architecture.

We compare the computational performances of cuTauLeaping with the CPU implementation of tau-leaping provided in the software COPASI, a well known application for the simulation and the analysis of biochemical networks [66]. To this aim, we exploit as benchmarks four stochastic models of biological systems of increasing complexity, formally described in Text S1. In addition, to analyze the influence of the size of the model (i.e., number of reactions and molecular species) on the performances of cuTauLeaping, we executed several tests on randomly generated synthetic models characterized by different size and various parameterizations.

Finally, we show the advantages of using cuTauLeaping to investigate the effects of systematic perturbations on the system dynamics. To this aim, we performed different parameter sweep analyses (PSA) on the Schlögl and Ras/cAMP/PKA models, in which one (PSA-1D), two (PSA-2D) or three (PSA-3D) parameters were simultaneously varied within given sweep intervals (chosen with respect to a fixed reference value for each parameter). Within these ranges, the numerical values of each varied parameter were determined with a linear sampling for the amounts of molecular species; a logarithmic sampling was instead considered for stochastic constants (if not stated otherwise), in order to uniformly span over many orders of magnitude. All PSA were performed by generating a set of different initial conditions – corresponding to different parameterizations of the model under investigation – and then automatically executing the parallel stochastic simulations with cuTauLeaping.

Design and implementation of cuTauLeaping

In cuTauLeaping, the workflow of the traditional tau-leaping algorithm is partitioned in different phases, which altogether allow a better exploitation of the parallel architecture of the GPU than a monolithic implementation. The rationale behind this choice is that the resources on each SM are limited, thus they would be quickly consumed by the data structures employed by tau-leaping, causing a low occupancy of the GPU that would then result in worse performances. Moreover, since tau-leaping embeds the potential execution of SSA simulation steps, a “fat” kernel responsible for both simulation algorithms would not be convenient, because of the following issues: (i) when the largest permissible time step τ for non-critical reactions is very low, it is faster to forgo tau-leaping and to execute an arbitrary number of SSA steps (see Step 3 in the modified Poisson tau-leaping algorithm presented in [27]); (ii) SSA is simpler and requires fewer resources than tau-leaping (the only thing the two algorithms share is the vector of propensity functions). Therefore, the partitioning of tau-leaping workflow in different phases allows a faster execution of the simulations, thanks to the reduced memory footprint, which yields a higher level of parallelism.

In this section we describe in detail the design and the implementation of the different CUDA kernels that stand at the basis of cuTauLeaping. The four phases that constitute cuTauLeaping, schematized in Figure 2, are:

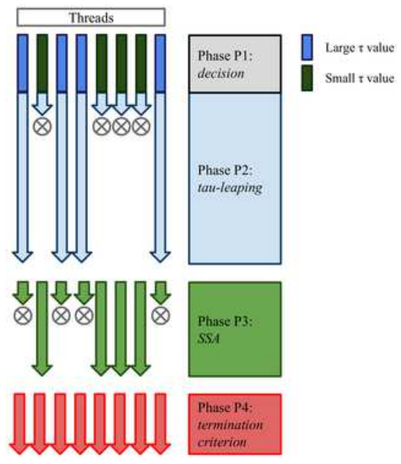


Figure 2. cuTauLeaping workflow.
Simplified scheme of cuTauLeaping workflow: in phase P1 each thread calculates the value τ for the simulation step; in phase P2, the threads whose τ is “large” perform a tau-leaping step **(by executing a set of non-critical reactions and (possibly) one critical reaction)**; the remaining threads perform a fixed number of SSA steps **(where one reaction is executed at each step)** during phase P3. The phases are iterated until all threads have reached t_{max}, a termination criterion verified during phase P4.
doi:10.1371/journal.pone.0091963.g002

phase $\mathrm{P1}$: each thread i, where $i=1,\dots,U$, and $U\in\mathbb{N}$ is specified by the user, determines a tentative value for the length of the time step τ for the non-critical reactions, by using Equation (3);

phase $\mathrm{P2}$: all threads where the length of the time step is such that $\tau\geq\frac{10}{a_0(\mathbf{x})}$ execute a tau-leaping step;

phase $\mathrm{P3}$: the remaining threads execute a fixed number of SSA steps (100 in our default setting);

phase $\mathrm{P4}$: check the termination criterion of the simulation in all threads (cuTauLeaping termination).

Each thread proceeds by applying tau-leaping or SSA steps, which are mutually exclusive, according to the value of a vector $\mathbf{Q}\in\{-1,0,1\}^U$, where for each $i=1,\dots U$ the element $q_i\in\mathbf{Q}$ is set to 0 if the i-th thread must execute SSA, 1 if the i-th thread must execute tau-leaping, while the value -1 corresponds to the signal of terminated simulation.

In cuTauLeaping the first two phases are implemented in a single kernel, so that the tau-leaping step can be executed right after the calculation of the τ value, without the need for a global memory write (e.g., to update $\mathbf{Q}$) or a recalculation of the propensity functions and τ (that could be required, in such a case), which would reduce the performances. In particular, during the first two phases, after the computation of the putative τ value for non-critical reactions, a second putative time step value related to critical reactions is calculated, and the smallest one is used in the current tau-leaping step. If the first putative τ value is used, then only non-critical reactions are sampled from the Poisson distributions and applied; otherwise, besides non-critical reactions, also one critical reaction is selected and applied (as described in [27]).

The four phases are implemented in the following kernels, which are executed in a sequential manner by each thread i:

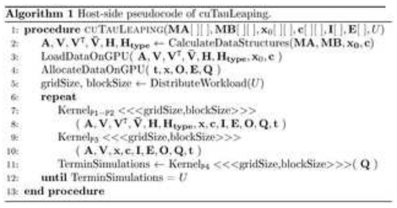
kernel $\mathrm{P1}-\mathrm{P2}$: if $q_i = -1$, then terminate the kernel; otherwise, calculate the τ value for non-critical reactions.

If $\tau < \frac{10}{a_0(\mathbf{x})}$, then $q_i \leftarrow 0$ and terminate the kernel; else $q_i \leftarrow 1$ and execute a tau-leaping step updating the system state $\mathbf{x}$ (according to Equation 2, by executing a set of non-critical reactions and, possibly, one critical reaction) and the global simulation time (by setting $t \leftarrow t+\tau$). If $t > t_{max}$, then $q_i \leftarrow -1$ and terminate the kernel;

kernel $\mathrm{P3}$: if $q_i \neq 0$, then terminate the kernel; otherwise, perform the SSA steps (by executing a single reaction at each step), and update the system state $\mathbf{x}$ (according to Equation 2) and the global simulation time t (by setting, at each SSA step, $t \leftarrow t+\tau$). If $t > t_{max}$ set $q_i \leftarrow -1$ and terminate the kernel;

kernel $\mathrm{P4}$: if $q_i = -1$ (for all threads), then terminate cuTauLeaping; else go back to kernel $\mathrm{P1}-\mathrm{P2}$.

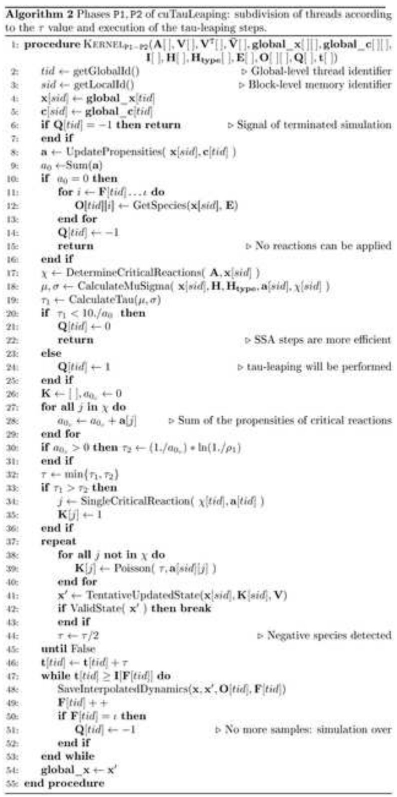
In Figure 3 we report the pseudocode of the host side procedure devoted to invoke the CUDA kernels; in Figures 4, 5, 6 we present the pseudocodes of kernels $\mathrm{P1}-\mathrm{P2}$, $\mathrm{P3}$, $\mathrm{P4}$, respectively. Kernels are iteratively repeated until $t \geq t_{max}$ for all threads. This termination criterion is efficiently verified by kernel $\mathrm{P4}$ that exploits two advanced CUDA functionalities introduced with the Fermi architecture: *synchronizations with predicate evaluation* and *atomic functions*. Synchronization functions are generally used to coordinate the communication between threads, but CUDA allows to exploit these functions to evaluate a predicate for all threads in a block; atomic functions allow to perform read-modify-write operations without any interference from any other thread, therefore avoiding the race condition. A combination of these functionalities allows to determine whether all threads have terminated their execution (i.e., the predicate is $q_i = -1$, for all $i = 1, \ldots, U$). In addition, since both functionalities are hardware-accelerated, the resulting computational complexity is $O(\#\text{blocks})$, making them more efficient than other equivalent methodologies, e.g., parallel reduction [67], whose complexity is $O(\log U)$ (note that, in general, $\#\text{blocks} < \log U$).



Algorithm 1 Host-side pseudocode of cuTauLeaping.

```
procedure CUTAULEAPING($\mathbf{MA}[\,][\,], \mathbf{MB}[\,][\,], \mathbf{x}_0[\,][\,], \mathbf{c}[\,][\,], \mathbf{I}[\,], \mathbf{E}[\,], U$)
    $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\intercal}, \bar{\mathbf{V}}, \mathbf{H}, \mathbf{H_{type}}$ ← CalculateDataStructures($\mathbf{MA}, \mathbf{MB}, \mathbf{x_0}, \mathbf{c}$)
    LoadDataOnGPU( $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\intercal}, \bar{\mathbf{V}}, \mathbf{H}, \mathbf{H_{type}}, \mathbf{x}_0, \mathbf{c}$ )
    AllocateDataOnGPU( $\mathbf{t}, \mathbf{x}, \mathbf{O}, \mathbf{E}, \mathbf{Q}$ )
    gridSize, blockSize ← DistributeWorkload($U$)
    repeat
        Kernel$_{P1-P2}$ <<<gridSize,blockSize>>>
            ( $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\intercal}, \bar{\mathbf{V}}, \mathbf{H}, \mathbf{H_{type}}, \mathbf{x}, \mathbf{c}, \mathbf{I}, \mathbf{E}, \mathbf{O}, \mathbf{Q}, \mathbf{t}$ )
        Kernel$_{P3}$ <<<gridSize,blockSize>>>
            ( $\mathbf{A}, \mathbf{V}, \mathbf{x}, \mathbf{c}, \mathbf{I}, \mathbf{E}, \mathbf{O}, \mathbf{Q}, \mathbf{t}$ )
        TerminSimulations ← Kernel$_{P4}$ <<<gridSize,blockSize>>>( $\mathbf{Q}$ )
    until TerminSimulations = $U$
end procedure
```

Figure 3. Pseudocode of cuTauLeaping – host side.
Host-side pseudocode of cuTauLeaping. As a first step, the stoichiometric information of the reactions is exploited to pre-calculate the data structures needed by the algorithm; all matrices are flattened during this process. Then, once the support memory areas are allocated (e.g., the chunk of global memory where the system dynamics will be stored), the four phases of cuTauLeaping begin and are repeated until all simulations are completed.
doi:10.1371/journal.pone.0091963.g003

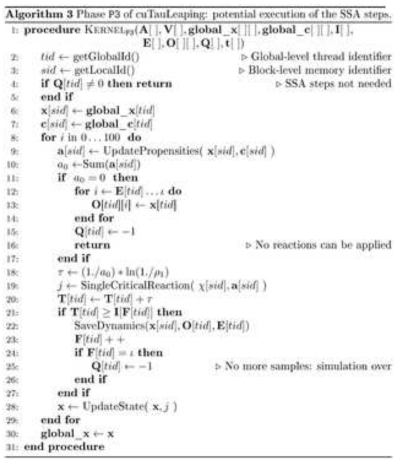


Algorithm 2 Phases P1, P2 of cuTauLeaping: subdivision of threads according to the τ value and execution of the tau-leaping steps.

```
procedure KERNEL$_{P1-P2}$($\mathbf{A}[\,], \mathbf{V}[\,], \mathbf{V}^{\intercal}[\,], \bar{\mathbf{V}}[\,]$, global_x[ ][ ], global_c[ ][ ],
                                   $\mathbf{I}[\,], \mathbf{H}[\,], \mathbf{H_{type}}[\,], \mathbf{E}[\,], \mathbf{O}[\,][\,], \mathbf{Q}[\,], \mathbf{t}[\,]$)
    $tid$ ← getGlobalId()                              ▷ Global-level thread identifier
    $sid$ ← getLocalId()                               ▷ Block-level memory identifier
    $\mathbf{x}[sid]$ ← global_x[$tid$]
    $\mathbf{c}[sid]$ ← global_c[$tid$]
    if $\mathbf{Q}[tid] = -1$ then return              ▷ Signal of terminated simulation
    end if
    $\mathbf{a}$ ← UpdatePropensities( $\mathbf{x}[sid], \mathbf{c}[tid]$ )
    $a_0$ ← Sum($\mathbf{a}$)
    if $a_0 = 0$ then
        for $i \leftarrow \mathbf{F}[tid] \ldots \iota$ do
            $\mathbf{O}[tid][i]$ ← GetSpecies($\mathbf{x}[sid], \mathbf{E}$)
        end for
        $\mathbf{Q}[tid] \leftarrow -1$
        return                                         ▷ No reactions can be applied
    end if
    $\chi$ ← DetermineCriticalReactions( $\mathbf{A}, \mathbf{x}[sid]$ )
    $\mu, \sigma$ ← CalculateMuSigma( $\mathbf{x}[sid], \mathbf{H}, \mathbf{H_{type}}, \mathbf{a}[sid], \chi[sid]$ )
    $\tau_1$ ← CalculateTau($\mu, \sigma$)
    if $\tau_1 < 10./a_0$ then
        $\mathbf{Q}[tid] \leftarrow 0$
        return                                         ▷ SSA steps are more efficient
    else
        $\mathbf{Q}[tid] \leftarrow 1$                  ▷ tau-leaping will be performed
    end if
    $\mathbf{K} \leftarrow [\,], a_{0_c} \leftarrow 0$
    for all $j$ in $\chi$ do
        $a_{0_c} \leftarrow a_{0_c} + \mathbf{a}[j]$    ▷ Sum of the propensities of critical reactions
    end for
    if $a_{0_c} > 0$ then $\tau_2 \leftarrow (1./a_{0_c}) * \ln(1./\rho_1)$
    end if
    $\tau \leftarrow \min(\tau_1, \tau_2)$
    if $\tau_1 > \tau_2$ then
        $j$ ← SingleCriticalReaction( $\chi[tid], \mathbf{a}[tid]$ )
        $\mathbf{K}[j] \leftarrow 1$
    end if
    repeat
        for all $j$ not in $\chi$ do
            $\mathbf{K}[j]$ ← Poisson( $\tau, \mathbf{a}[sid][j]$ )
        end for
        $\mathbf{x}'$ ← TentativeUpdatedState($\mathbf{x}[sid], \mathbf{K}[sid], \mathbf{V}$)
        if ValidState( $\mathbf{x}'$ ) then break
        end if
        $\tau \leftarrow \tau/2$                        ▷ Negative species detected
    until False
    $\mathbf{t}[tid] \leftarrow \mathbf{t}[tid] + \tau$
    while $\mathbf{t}[tid] \geq \mathbf{I}[\mathbf{F}[tid]]$ do
        SaveInterpolatedDynamics($\mathbf{x}, \mathbf{x}', \mathbf{O}[tid], \mathbf{F}[tid]$)
        $\mathbf{F}[tid]$ + +
        if $\mathbf{F}[tid] = \iota$ then
            $\mathbf{Q}[tid] \leftarrow -1$             ▷ No more samples: simulation over
        end if
    end while
    global_x ← $\mathbf{x}'$
end procedure
```

Figure 4. Pseudocode of cuTauLeaping – kernel $\mathrm{P1}-\mathrm{P2}$.
Device-side pseudocode of kernel $\mathrm{P1}-\mathrm{P2}$ in cuTauLeaping, implementing the subdivision of threads according to the τ value and the execution of a tau-leaping step. The kernel starts by loading the vectors **global_x** and **global_c** – which correspond to the current state of the system and to the values of stochastic constants, respectively – from the global memory areas that contain these data for all threads. Since these information are frequently accessed, they are immediately copied into the faster shared memory as vectors **x** and **c**, respectively. The kernel continues by verifying that the q_i value for the running thread i is not equal to the signal of terminated execution (i.e., $q_i = -1$). Then, it calculates the propensity functions of all reactions and accumulates their values in a_0; if $a_0 = 0$, the remaining time instants where the dynamics of the system is

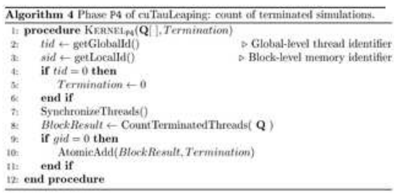
sampled are set to the current state and the simulation is terminated. The kernel concludes the phase P1 by calculating a putative τ value for the tau-leaping step: if τ is smaller than $10/a_0$, then thread i is halted and q_i is set to 0, so that it will perform the SSA steps during the next phase. Otherwise, the tau-leaping algorithm is performed by executing a set of non-critical reactions and (possibly) one critical reaction and, if the simulation has overrun one of the sampling time instants, the state stored in $\mathbf{F}$ is determined by linear interpolation.
doi:10.1371/journal.pone.0091963.g004



Algorithm 3 Phase P3 of cuTauLeaping: potential execution of the SSA steps.

```
procedure KERNEL_P3(A[ ], V[ ], global_x[ ][ ], global_c[ ][ ], I[ ],
                    E[ ], O[ ][ ], Q[ ], t[ ])
    tid ← getGlobalId()                                ▷ Global-level thread identifier
    sid ← getLocalId()                                 ▷ Block-level memory identifier
    if Q[tid] ≠ 0 then return                          ▷ SSA steps not needed
    end if
    x[sid] ← global_x[tid]
    c[sid] ← global_c[tid]
    for i in 0...100 do
        a[sid] ← UpdatePropensities( x[sid], c[sid] )
        $a_0$ ← Sum(a[sid])
        if $a_0 = 0$ then
            for i ← E[tid] ... ι do
                O[tid][i] ← x[tid]
            end for
            Q[tid] ← −1
            return                                     ▷ No reactions can be applied
        end if
        $\tau \leftarrow (1./a_0) * \ln(1./\rho_1)$
        j ← SingleCriticalReaction( χ[sid], a[sid] )
        T[tid] ← T[tid] + τ
        if T[tid] ≥ I[F[tid]] then
            SaveDynamics(x[sid], O[tid], E[tid])
            F[tid] + +
            if F[tid] = ι then
                Q[tid] ← −1                            ▷ No more samples: simulation over
            end if
        end if
        x ← UpdateState( x, j )
    end for
    global_x ← x
end procedure
```

Figure 5. Pseudocode of cuTauLeaping – kernel $\mathrm{P3}$.
Device-side pseudocode of kernel $\mathrm{P3}$ in cuTauLeaping, implementing the execution of the SSA steps. The kernel starts by loading the vectors $\mathbf{global\_x}$ and $\mathbf{global\_c}$ – which correspond to the current state of the system and to the values of stochastic constants, respectively – from the global memory areas that contain these data for all threads. Since these information are frequently accessed, they are immediately copied into the faster shared memory as vectors **x** and **c**, respectively. The kernel continues by verifying that the q_i value for the running thread i is equal to the signal corresponding to SSA (i.e., $q_i = 0$). Then, it performs a fixed number of SSA steps (100 in our default setting), **where a single reaction is executed at each step**, storing the system state at the sampled time instants $\mathbf{I}$.
doi:10.1371/journal.pone.0091963.g005



Algorithm 4 Phase P4 of cuTauLeaping: count of terminated simulations.

```
procedure KERNEL_P4(Q[ ], Termination)
    tid ← getGlobalId()                                ▷ Global-level thread identifier
    sid ← getLocalId()                                 ▷ Block-level memory identifier
    if tid = 0 then
        Termination ← 0
    end if
    SynchronizeThreads()
    BlockResult ← CountTerminatedThreads( Q )
    if gid = 0 then
        AtomicAdd(BlockResult, Termination)
    end if
end procedure
```

Figure 6. Pseudocode of cuTauLeaping – kernel $\mathrm{P4}$.
Device-side pseudocode of kernel $\mathrm{P4}$ in cuTauLeaping, implementing the verification of the termination of all simulations. The verification is performed by means of CUDA's hardware accelerated synchronization and counting features, which allow to count the threads of a block which satisfy a specific predicate. By exploiting CUDA's atomic functions, we accumulate the total number of threads which satisfy the predicate $q_i = -1$: if it is equal to the number of threads, the execution of all parallel simulations is completed.
doi:10.1371/journal.pone.0091963.g006

In order to further improve the performance of the simulation execution, it is better not to code the stoichiometric information by means of matrices. In cuTauLeaping, we flattened the stoichiometric matrices $\mathbf{MA}, \mathbf{MV}$, $\mathbf{MV}^{\mathrm{T}}$ and $\overline{\mathbf{MV}}$ – which are typically sparse matrices – by packing their non-zero elements into arrays of CUDA vector types, named $uchar4$, whose components are accessed by means of $variable.x$, $.y$, $.z$, and $.w$; the vectors corresponding to these matrices are named $\mathbf{A}, \mathbf{V}, \mathbf{V}^{\mathrm{T}}, \bar{\mathbf{V}}$, respectively. Since both $\mathbf{V}^{\mathrm{T}}$ and $\bar{\mathbf{V}}$ vectors can assume negative numbers, we use an offset to store their $.z$ values as *unsigned chars*, and subtract the offset during the calculations on the GPU to yield back the correct negative numbers. An example of this implementation strategy is shown in Figure 7 which schematizes, for the Michaelis-Menten model, the conversion of the matrix $\mathbf{MA}$ into the corresponding flattened representation $\mathbf{A}$. By using this strategy, the complexity of the calculations needed for both SSA and tau-leaping decreases from $\Theta(M \cdot N)$ to $\Theta(Z)$, where $Z = |\{\alpha_{ji} \in \mathbf{MA} | \alpha_{ji} \neq 0\}|$ is the number of non-zero entries in $\mathbf{MA}$. For each non-zero entry, we store into the $.x$ and $.y$ components the corresponding row and column indices of $\mathbf{MA}$, respectively; the $.z$ component is used to store the stoichiometric value. Note that, even though the $.w$ component is left unused, it is more efficient to employ the $uchar4$ vector type rather than $uchar3$, because the former is 4-aligned and takes a single instruction to fetch the whole entry, while the latter is 1-aligned, and would require three memory operations to read each entry of the flattened vector. It is worth noting that the use of an *unsigned char* data type implies that cuTauLeaping could deal with models with up to 256 reactions and molecular species; for larger systems, data types with greater size must be exploited. Anyway, the maximum size of a model is also limited by the shared memory available on the GPU; we provide a detailed analysis of this issue in the Discussion section.

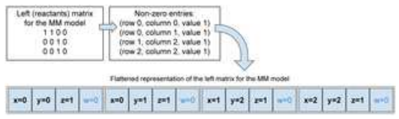

Left (reactants) matrix
for the MM model
1 1 0 0
0 0 1 0
0 0 1 0
Non-zero entries:
(row 0, column 0, value 1)
(row 0, column 1, value 1)
(row 1, column 2, value 1)
(row 2, column 2, value 1)
Flattened representation of the left matrix for the MM model
x=0
y=0
z=1
w=0
x=0
y=1
z=1
w=0
x=1
y=2
z=1
w=0
x=2
y=2
z=1
w=0

Figure 7. Schematization of the flattened representation of the stoichiometric information.

The stoichiometry of chemical reactions is generally represented by (usually sparse) matrices, corresponding to the variation of the species appearing either as reactants or products; however, both tau-leaping and SSA exploit only the non-zero values of these matrices. Each stoichiometric matrix can be pre-processed to identify its non-zero values and discard the remaining ones, thus reducing the number of reading operations required by the two stochastic algorithms. Our strategy to reduce the size of these matrices consists in flattening each matrix as a vector of triples (r,c,v), where r is the row index, c is the column index and v is the non-zero value in (r,c). In our implementation, both r and c indices are 0-based and triples are stored using vectors of CUDA's $uchar4$ data types, that have the advantage of requiring a single instruction to fetch an entry. The top part of this figure shows the values appearing in the 3×4 stoichiometric matrix of reactant species of the Michaelis-Menten model (MM), which consists of 3 reactions over 4 molecular species (see Text S1). Note that only four cells of this matrix have non-zero values; the bottom part of the figure shows the corresponding $uchar4$ vector.
doi:10.1371/journal.pone.0091963.g007

cuTauLeaping is also optimized for what concerns the calculation of $\mu_i(x)$ and $\sigma_i^2(x)$ values (Equation 4). These values are related to each species S_i, and represent an estimate of the change of the propensity functions, based on all possible reactions in which the species S_i is involved. For this reason, the flattened representation of the matrix $\mathbf{MV}$ cannot be exploited here; therefore, to obtain an efficient calculation of these values, we introduced the flattened transposed stoichiometric matrix $\mathbf{V}^{\mathrm{T}}$.

In order to further increase the performances of cuTauLeaping, we also optimized the CUDA code to better exploit the GPU architecture. The first optimization consists in keeping the register pressure low, in order to avoid the *register spilling* into global memory and to increase the occupancy of the GPU. This is achieved by partitioning the tau-leaping algorithm into multiple kernels, allowing a strong reduction of the consumption of hardware resources (i.e., the register pressure). This CUDA optimization technique, known as *branch splitting*, was shown to achieve a relevant gain in performances [68].

Another typical optimization of CUDA is to ensure coalesced access to data, i.e., an aligned and sequential organization of the memory, for all data structures that are updated by each thread. In our implementation we granted coalescence to all data structures that are private to each thread, that is, the system state $\mathbf{x}$, the stochastic constants $\mathbf{c}$, and so on. However, there is some shared information that is not inherently coalesced:

- the stoichiometric flattened vectors $\mathbf{A}$ and $\mathbf{V}$ (used by kernel $\mathrm{P1-P2}$ and kernel $\mathrm{P3}$), $\mathbf{V}^{\mathrm{T}}$ and $\bar{\mathbf{V}}$ (used by kernel $\mathrm{P1-P2}$);
- the vector $\mathbf{H}\in\mathbb{N}^N$ (used by kernel $\mathrm{P1-P2}$), containing the highest order of all reactions in which each molecular species appears as a reactant;
- the vector $\mathbf{H_{type}}\in\mathbb{N}^N$ (used by kernel $\mathrm{P1-P2}$), containing the information about the maximum number of reactant molecules involved in the highest-order reactions.

The data structures used to store the stoichiometric information ($\mathbf{A},\mathbf{V},\mathbf{V}^{\mathrm{T}},\bar{\mathbf{V}}$) are not modified during the simulation and are common to all threads, and can be conveniently loaded into the constant memory. This peculiar CUDA memory is immutable and cached, so that the uncoalesced access pattern does not have any impact on the performances.

Note that, in contrast to other GPU implementations of tau-leaping [39], cuTauLeaping exploits the vector $\mathbf{A}$ to correctly evaluate the propensity functions of the reactions whose reactant species appear also as products in the same reaction: in cases like this, the net balance of the consumed and produced chemicals that is stored in vector $\mathbf{V}$ would not carry sufficient information to distinguish between reactants and products. For instance, given $\mathbf{MV}_j=(-1,-1,1)$, it is impossible to establish whether this state change vector corresponds to a reaction of the form $S_1+S_2\rightarrow S_3$ or $2S_1+2S_2\rightarrow S_1+S_2+S_3$; on the contrary, the information stored in $\mathbf{A}$ allows to discriminate between the two cases.

$\mathbf{H}$ and $\mathbf{H_{type}}$ vectors are necessary to evaluate the length of the step τ (Equation 3), which is determined at each step according to the current state of the system; τ is then exploited to update the simulation time of each thread, denoted by $\mathbf{t}\in\mathbb{R}^U$. Both $\mathbf{H}$ and $\mathbf{H_{type}}$ vectors, anyway, can be calculated offline by preprocessing the stoichiometric matrices $\mathbf{MA}$ and $\mathbf{MB}$ while they are loaded.

In addition, both kernels $\mathrm{P1-P2}$ and $\mathrm{P3}$ exploit the following three vectors:

- $\mathbf{x}\in\mathbb{N}^N$ is the current state of the system;
- $\mathbf{a}\in\mathbb{R}^M$ contains the values of the propensity functions of the reactions;
- $\mathbf{c}\in\mathbb{R}^M$ contains the values of the stochastic constants.

Kernel $\mathrm{P1-P2}$ exploits four additional vectors:

- $\mathbf{x}'\in\mathbb{Z}^N$ is the putative state of the system (note that the elements of this vector might assume negative values);
- $\chi\in\{0,1\}^M$ contains the information about critical reactions, stored as 1 s, and non-critical reactions, stored as 0 s;
- $\mathbf{G}\in\mathbb{R}^N$ contains the auxiliary values used to calculate the length of the time step satisfying the leap condition, which are computed by using the vectors $\mathbf{x}$, $\mathbf{H}$ and $\mathbf{H_{type}}$ (see details in [27]);
- $\mathbf{K}\in\mathbb{N}^M$ contains the samples of the Poisson distributions corresponding to the number of times each reaction will be fired in the current step.

These vectors are coalesced, but frequently exploited by tau-leaping and SSA. In order to minimize the latencies due to the frequent access to the global memory, for each thread we allocate $\mathbf{x}$, $\mathbf{x}'$, $\mathbf{a}$, $\mathbf{c}$, χ and $\mathbf{K}$ into the shared memory. Being an on-chip memory, latencies of the shared memory are about two orders of magnitude lower than that of the global memory; the use of shared memory allows a reduction of the global bandwidth usage [69] and provides a relevant performance boost. In contrast, we memorize $\mathbf{G}$ into the global memory, since its values are used only twice during the simulation step to determine the τ value. Since cuTauLeaping was specifically designed to be embedded into other applications – in particular, the computational tools for PE, PSA and RE that we previously developed [6], [7], [35], which rely on the execution of a large number of simulations – we also copy the stochastic constants into the shared memory vector $\mathbf{c}$.

To obtain a more efficient implementation, an additional strategy consisted in restructuring the tau-leaping algorithm in order to avoid the conditional branches as much as possible. In cuTauLeaping, branches were removed by unrolling loops and by allowing redundant calculations in favor of a uniform control flow.

There are two more relevant facets of cuTauLeaping implementation: the storage of the simulated dynamics and the "feed" of molecular species amounts. The storage of the entire temporal evolution of all species, associated to each thread on the GPU, cannot be realized, since we cannot determine *a priori* how many steps each simulation will take. Indeed, whenever a kernel is launched, the required amount of memory must be statically pre-allocated from the host; it is therefore fundamental to set the number of time instants in which the dynamics is sampled, before each simulation starts. Moreover, the naïve storage of all molecular species, even in the case of a few sampled time instants, may be unfeasible. For instance, in the case of the Ras/cAMP/PKA pathway, storing the dynamics of 2^{12} simulations, considering 1000 samples stored as single-precision floating point values, would require 4(bytes)$\times 1000$(samples)$\times 33$(species)$\times 2^{12}$(simulations)$\simeq 0.5$ GB, which is very close to the memory amount of an average GPU. Therefore, we make use of four additional global memory vectors:

- $\mathbf{I}\in\mathbb{R}^{\imath}$ contains the $\imath\in\mathbb{N}$ time instants in which the temporal evolution of the system is stored;
- $\mathbf{E}\in\mathbb{N}^{\kappa}$ contains the indices of the $\kappa\in\mathbb{N}$ molecular species whose amounts are stored (for some $\kappa\leq N$);
- $\mathbf{O}\in\mathbb{N}^{\kappa\times\imath\times U}$ contains the U simulated dynamics of the molecular species in $\mathbf{E}$, sampled during the time instants in $\mathbf{I}$;
- $\mathbf{F}\in\mathbb{N}^{U}$ stores, for each thread, the pointer to the next time instant in $\mathbf{O}$, i.e., when the u-th simulation reaches the i-th sampling time instant, $\mathbf{F}_u$ is set to $i+1$.

Finally, in order to fully exploit the SM cache, also the values of $\imath$, κ, M, N, n_c, ϵ, t_{max} and the size of the flattened vectors $\mathbf{A},\mathbf{V},\mathbf{V}^{\mathrm{T}}$ and $\bar{\mathbf{V}}$, are stored as constants into the constant memory of the GPU.

Computational results

In this section we present a comparison of the computational effort of GPU and CPU for the simulation of four stochastic models of increasing size and complexity: the Michaelis-Menten kinetics (MM) [40], a prokaryotic gene regulatory network (PGN) [35], [41], the Schlögl model [70], [71] and the Ras/cAMP/PKA signal transduction pathway in yeast [6], [43], [44]. The definition of each model, as well as the values of the initial molecular amounts and of the stochastic constants used to run simulations, are given in Text S1.

To analyze the performances of cuTauLeaping, the same simulations executed on GPU were carried out on a CPU architecture by exploiting the tau-leaping algorithm implemented in the software COPASI (version 4.8 Build 35, running on Windows 7 64-bit) [66]. COPASI has been recently integrated with a server-side tool, named Condor [33], that handles COPASI jobs, automatically splits them in sub-jobs and distributes the calculations on a cluster of heterogeneous machines; in the present work we do not use this possibility as we are interested in COPASI as a single-node CPU-bound reference implementation, which is currently single-threaded and does not exploit the physical and logical cores of the CPU.

The GPU used for the tests is a Nvidia GeForce GTX 590, a dual-GPU video card equipped with 2×16 SMs for a total of 1024 cores (cuTauLeaping automatically distributes the workload on the available SMs); the performances were compared with a quad-core CPU Intel Core i7-2600 with a clock rate of 3.4 GHz.

In all simulations, the total simulation time t_{max} for MM, PGN and Schlögl models was set to 5.0, 50.0 and 10.0 a.u., respectively, while for the Ras/cAMP/PKA model it was set to 150.0 a.u.; for each simulation, we stored 100 samples of all the molecular species occurring in the system. The value of the error control parameter of tau-leaping was set to $\epsilon=0.03$, as suggested in [27].

The results of the comparison between cuTauLeaping and COPASI CPU tau-leaping are summarized in Table 1, which reports the running time (in seconds) obtained by executing different batches of simulations of each model; these results were obtained using the RNG MRG32K3A in cuTauLeaping. Table 1 clearly show the advantage of cuTauLeaping as the number of simulations increases. Interestingly, because of the architectural differences and the different clock rates, a single run of cuTauLeaping may be slower than the CPU counterpart, and it becomes fully profitable only by running multiple simulations. For instance, in the case of the Ras/cAMP/PKA model (Figure 8d), the break-even is reached around 2^7 simulations. Thus, when less than ≈100 simulations of this specific pathway are needed, the use of a CPU implementation may be more convenient. Nonetheless, statistical analyses of stochastic temporal evolutions of biological systems require large batches of simulations (usually $\gg100$) to derive statistically significant measures of the analyzed system dynamics. Note that, in general, the analytical determination of the break-even for an arbitrary model is a hard task, because it depends on its size (the number of reactions and molecular species) as well as on its parameterization that might lead to stiffness phenomena, able to affect the running time of the used simulation algorithm. Moreover, if a biological system is characterized by multistability or very large fluctuations in the dynamics of some molecular species (e.g., those occurring in low amounts), simulations running in different threads can be characterized by a high divergence of the execution flow, thus resulting in reduced parallelism and, consequently, in worse performances. For instance, if two threads simulating the same system reach very different system states, they can take different branches within the code (e.g., due to a rejection of an invalid putative state $\mathbf{x}'$ of the system), an event that can greatly affect the performance of the GPU.

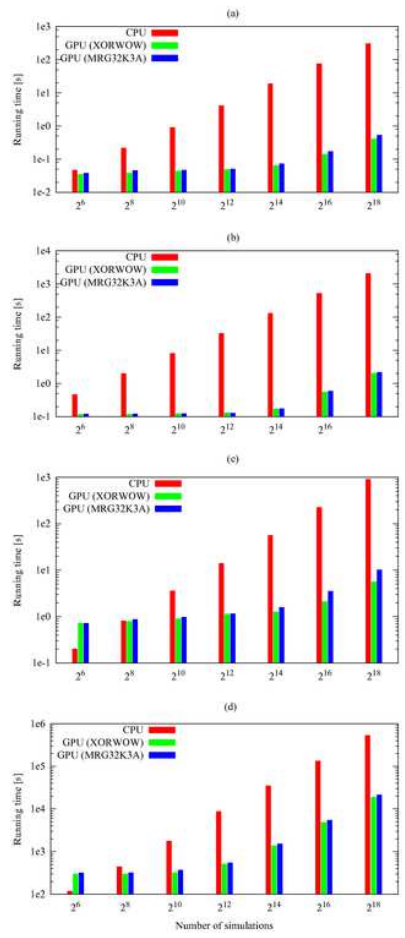
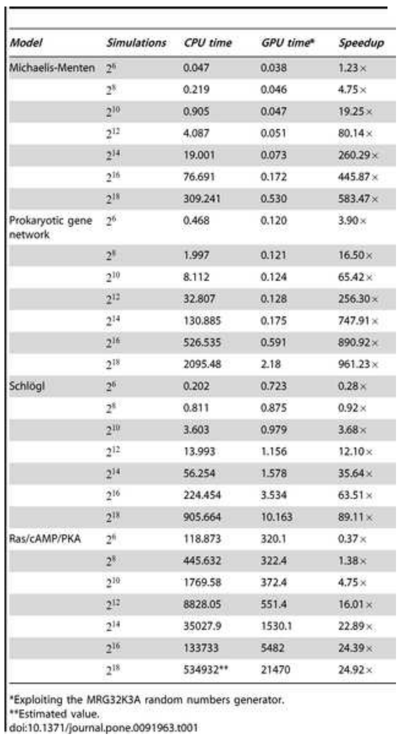


Figure 8. Comparison of the computational time of CPU tau-leaping and cuTauLeaping.
Comparison between the computational time taken by cuTauLeaping and COPASI CPU tau-leaping to execute different batches of stochastic simulations of the Michaelis-Menten (MM) model (**a**), the Prokaryotic Gene Network (PGN) model (**b**), the Schlögl model (**c**), and the Ras/cAMP/PKA pathway (**d**) (see Text S1 for models definitions). For each model, cuTauLeaping becomes more profitable than the CPU counterpart when a certain number of parallel simulations is run, with a break-even that depends on the complexity of the system: for the MM and PGN models, cuTauLeaping is more effective when around 2^4 parallel simulations are run, while for the Schlögl model and the Ras/cAMP/PKA pathway the break-even is around 2^7 simulations. Considering the speedup, the best results achieved with cuTauLeaping – with respect to COPASI – are around 583× for the MM model, 961× for the PGN model, 90× for the Schlögl model, and 25× for the model of the Ras/cAMP/PKA pathway (see also Table 1).
doi:10.1371/journal.pone.0091963.g008



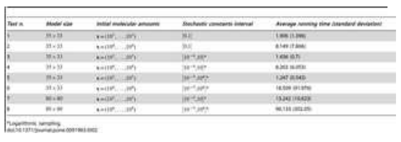
Model	Simulations	CPU time	GPU time*	Speedup
Michaelis-Menten	2^6	0.047	0.038	1.23×
	2^8	0.219	0.046	4.75×
	2^{10}	0.905	0.047	19.25×
	2^{12}	4.087	0.051	80.14×
	2^{14}	19.001	0.073	260.29×
	2^{16}	76.691	0.172	445.87×
	2^{18}	309.241	0.530	583.47×
Prokaryotic gene network	2^6	0.468	0.120	3.90×
	2^8	1.997	0.121	16.50×
	2^{10}	8.112	0.124	65.42×
	2^{12}	32.807	0.128	256.30×
	2^{14}	130.885	0.175	747.91×
	2^{16}	526.535	0.591	890.92×
	2^{18}	2095.48	2.18	961.23×
Schlögl	2^6	0.202	0.723	0.28×
	2^8	0.811	0.875	0.92×
	2^{10}	3.603	0.979	3.68×
	2^{12}	13.993	1.156	12.10×
	2^{14}	56.254	1.578	35.64×
	2^{16}	224.454	3.534	63.51×
	2^{18}	905.664	10.163	89.11×
Ras/cAMP/PKA	2^6	118.873	320.1	0.37×
	2^8	445.632	322.4	1.38×
	2^{10}	1769.58	372.4	4.75×
	2^{12}	8828.05	551.4	16.01×
	2^{14}	35027.9	1530.1	22.89×
	2^{16}	133733	5482	24.39×
	2^{18}	534932**	21470	24.92×

*Exploiting the MRG32K3A random numbers generator.
**Estimated value.
doi:10.1371/journal.pone.0091963.t001

Table 1. Comparison of computational time of COPASI CPU tau-leaping and cuTauLeaping.
doi:10.1371/journal.pone.0091963.t001

For the sake of completeness, we also plot in Figure 8 the running times of cuTauLeaping obtained by using both RNGs available in cuTauLeaping, XORWOW and MRG32K3A, and compare these results with the running time of COPASI CPU tau-leaping (see also Table 2 in Text S2). To fully compare these two RNGs, in Text S2 we show an example of the frequency distribution of the amount of the molecular species X in the Schlögl model, obtained executing different batches of parallel simulations using either MRG32K3A or XORWOW, and present the results of the Kolmogorov-Smirnov test. Altogether, our results show that, despite XORWOW presents statistical flaws and therefore should not be safely exploited to carry out Monte Carlo simulations, the frequency distributions of X obtained with XORWOW and MRG32K3A can be considered equivalent; in addition, XORWOW allows to achieve a higher speedup with respect to MRG32K3A.

Finally, in order to investigate the influence of the size of the simulated system on cuTauLeaping performances, we executed several tests on randomly generated synthetic models (RGSM). In particular, we analyzed six distinct parameterizations of 100 different RGSM, each one consisting of 35 reactions and 33 species. The rationale behind the choice of this model size and various initial conditions was to compare the computational costs of cuTauLeaping for the simulation of RGSM on the one side, and the Ras/cAMP/PKA model on the other side. RGSM were generated according to the methodology proposed by Komarov *et al.* [39], and following two different strategies for the selection of the values of the stochastic constants. The results of these tests are given in Table 2 where, for each synthetic model, the average running times (given in seconds) were evaluated by executing 2^{10} parallel simulations with $t_{max} = 150$ a.u., to allow a direct comparison with the results listed in Table 1.



Test n.	Model size	Initial molecular amounts	Stochastic constants interval	Average running time (standard deviation)
1	35 × 33	$x = (10^3, \ldots, 10^3)$	[0,1]	1.806 (1.266)
2	35 × 33	$x = (10^4, \ldots, 10^4)$	[0,1]	8.149 (7.866)
3	35 × 33	$x = (10^3, \ldots, 10^3)$	[illegible]*	1.436 (0.7)
4	35 × 33	$x = (10^4, \ldots, 10^4)$	[illegible]*	8.202 (6.053)
5	35 × 33	$x = (10^3, \ldots, 10^3)$	[illegible]*	1.247 (0.543)
6	35 × 33	$x = (10^4, \ldots, 10^4)$	[illegible]*	16.509 (91.976)
7	[illegible]	$x = (10^3, \ldots, 10^3)$	[illegible]*	13.242 (16.423)
8	[illegible]	$x = (10^4, \ldots, 10^4)$	[illegible]*	90.133 (302.05)

*Logarithmic sampling.
doi:10.1371/journal.pone.0091963.t002

Table 2. Running times of cuTauLeaping for the simulation of randomly generated synthetic models.
doi:10.1371/journal.pone.0091963.t002

In tests 1 and 2 we randomly selected the values of stochastic constants of the RGSM with a uniform probability in $[0,1]$; the initial molecular amounts were set to 10^3 (test 1) and 10^4 (test 2) for all species appearing in the systems. Both tests show that, on average, the computational time required for 2^{10} parallel simulations of RGSM is much lower than the time needed for the same number of parallel simulations of the Ras/cAMP/PKA model. In addition, we observe that the initial molecular amounts considered in the parameterizations actually influence the results; in general, higher quantities lead to higher average running times.

In tests 3 to 6 we exploited a modified strategy to select the values of stochastic constants, which were logarithmically sampled in the given range in order to uniformly span over different orders of magnitude. To obtain more realistic parameterizations, in tests 3 and 4 this range was defined according to the smallest and to the highest values of stochastic constants appearing in the Ras/cAMP/PKA model (see Table 6 in Text S1); in tests 5 and 6, the range was widened to six orders of magnitude larger than the values used in tests 3 and 4. Also in these cases, the average running time required for the simulation of the RGSM is lower than the time needed to execute the same number of parallel simulations of the Ras/cAMP/PKA model. Moreover, we observe that the running time is mainly influenced by the initial molecular amounts rather than the values of the stochastic constants used in the different parameterizations.

In addition, we carried out 2^{10} parallel stochastic simulations of RGSM whose sizes are approximately $4\times$ bigger than the Ras/cAMP/PKA model (tests 7 and 8). Even considering the worst result (test 8), the average running time required to simulate these larger synthetic models is still low, suggesting that the system size has not a direct impact on the performances of cuTauLeaping, while the complexity of the system – such as the presence of positive or negative feedbacks possibly leading to oscillatory dynamics (as in the Ras/cAMP/PKA model), or reactions that lead to stiffness (as in the Schlögl model) – is much more relevant.

The analysis of bistability in the Schlögl model

Bistability is a capacity exhibited by many biological systems, consisting in the possibility of switching between two different stable steady states in response to some chemical signaling (see, e.g., [72]–[74] and references therein). The Schlögl model is one of the simplest prototypes of chemical systems presenting a bistable dynamic behavior [70], [71]. This system is characterized by the fact that, starting from the same initial conditions, its dynamics can reach either the low or the high steady state; switches between the two steady states can also occur due to stochastic fluctuations.

cuTauLeaping can be efficiently exploited for the execution of a massive number of simulations of the Schlögl model, with the same initial parameterization, in order to produce a frequency distribution of the molecular species that exhibits the bistable behavior. Generally speaking, this kind of investigation allows the implicit identification of attractors and multiple steady states of a system, and helps to (empirically) determine the probability of reaching a particular state during the dynamical evolution of the system itself.

To this aim, we performed 2^{18} simulations of the Schlögl model, keeping track of the molecular amount of species X in each simulation, sampled in 100 time instants uniformly distributed over the simulation time. In particular, to detect the initial jump either to the low or to the high steady states, that takes place in the first time instants of the simulations, we performed 2^{17} simulations with $t_{max} = 3$ a.u.; then, for a deeper investigation of the bistable switching behavior of the system, the other simulations were executed considering $t_{max} = 150$ a.u.. We used the results of the simulations to calculate the histograms of the molecular amount of X, that were then exploited to realize a heatmap showing the frequency distribution of this species between the two stable steady states. In Figure 9a we show the initial transient of the dynamics, where a slightly higher probability to reach the low steady state can be observed, starting from the initial configuration of the Schlögl system (described in Tables 4 and 5 in Text S1). Figure 9b shows the frequency distribution of reaching either the low or the high steady state (around 100 and 600 molecules of species X, respectively), highlighting a larger variance concerning the high steady state.

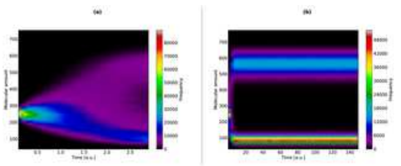


Figure 9. Frequency distribution of bistable states in the Schlögl model.
Frequency distribution of the molecular amount of molecular species X in the Schlögl model, calculated using a total of 2^{18} parallel simulations executed by cuTauLeaping. (**a**) Plot of the frequency distribution of X considering $t_{max} = 3$ a.u., to detect the bistable switching behavior that takes place in the first time instants of the dynamics; a slightly higher probability to reach the low steady state can be observed, starting from the initial state of the Schlögl system (described in Text S1). (**b**) Plot of the frequency distribution of X considering $t_{max} = 150$ a.u., to investigate the stability of the two steady states of the system; the heatmap highlights the two stable states (around 100 and 600 molecules of species X), and shows larger stochastic fluctuations around the high steady state.
doi:10.1371/journal.pone.0091963.g009

In Figure 10, we show the frequency distribution of molecular amounts of X in perturbed conditions of the Schlögl model, evaluated by exploiting a PSA-1D in which the value of the stochastic constant c_3 is uniformly varied in the interval $[6.9 \cdot 10^{-4}, 1.4 \cdot 10^{-3}]$. The frequency distribution was calculated according to 2^{18} simulations, where the dynamics was sampled at the single time instant $t = 10$ a.u., according to ten different values of the stochastic constant c_3 in the chosen sweep range. The total running time to execute this PSA-1D was 34.92 seconds with cuTauLeaping, and 1013.51 seconds with COPASI, thus achieving a 116× speedup. Figure 10 shows that increasing values of c_3 induce a decrease (increment, respectively) in the frequency distribution of X concerning the low (high, respectively) steady state, whereas for intermediate values of c_3 the system is characterized by an effective bistable behavior.

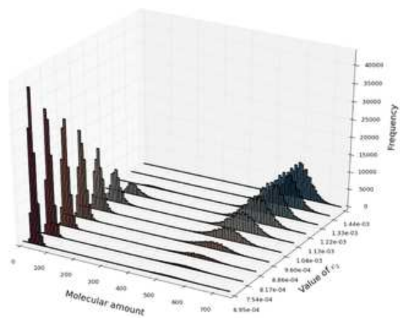


Figure 10. Parameter sweep analysis of the Schlögl model.
Results of a PSA-1D on the Schlögl model, in which the value of the stochastic constant c_3 is varied in the interval $[6.9 \cdot 10^{-4}, 1.4 \cdot 10^{-3}]$ (the set of reactions and the values of all other parameters are given in Tables 4 and 5 in Text S1). Each frequency distribution is calculated according to 2^{18} simulations executed by cuTauLeaping, measuring the amount of the molecular species X at the time instant $t = 10$ a.u., considering ten different values of the stochastic constant c_3 within the sweep interval. The figure shows that increasing values of c_3 induce a decrease (increment) in the frequency distribution of X concerning the low (high) steady state, with intermediate values of c_3 characterized by an effective bistable behavior.
doi:10.1371/journal.pone.0091963.g010

Finally, in Figure 11 we show the results of a PSA-3D performed by simultaneously varying the values of the stochastic constants c_1, c_2 and c_3 in the ranges $[2.9 \cdot 10^{-7}, 3.1 \cdot 10^{-7}]$, $[8.0 \cdot 10^{-5}, 1.1 \cdot 10^{-4}]$ and $[6.0 \cdot 10^{-4}, 1.2 \cdot 10^{-3}]$, respectively; for each stochastic constant, taken independently from the others, the chosen range corresponds to a condition of effective bistability of the Schlögl model. The values of the three stochastic constants were uniformly sampled in a $16 \times 16 \times 16$ three-dimensional lattice; for each sample, we executed 256 simulations (for a total of 2^{20} simulations) and evaluated the frequency distribution of the amount of species X at the time instant $t = 10$ a.u.. This set of values was then partitioned according to the reached (low or high) stable steady state; in Figure

11, the red (blue, respectively) region corresponds to the parameterizations of the model which yield the high (low, respectively) steady state most frequently. The green region represents a set of conditions whereby both steady states are equally reached.

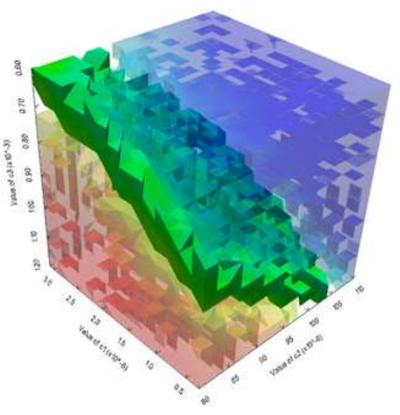


Figure 11. Three-dimensional parameter sweep analysis of the Schlögl model.
Results of a PSA-3D on the Schlögl model, performed by varying the stochastic constants c_1, c_2 and c_3 in the intervals $[2.9\cdot 10^{-7}, 3.1\cdot 10^{-7}]$, $[8.0\cdot 10^{-5}, 1.1\cdot 10^{-4}]$ and $[6.0\cdot 10^{-4}, 1.2\cdot 10^{-3}]$, respectively. The values of the stochastic constants were uniformly sampled in a $16\times 16\times 16$ three-dimensional lattice; for each sample, we executed 256 simulations with cuTauLeaping (for a total of 2^{20} simulations) and evaluated the frequency distribution of the amount of the molecular species X at the time instant $t=10$ a.u.. This set of values was then partitioned according to the reached (low or high) stable steady state; in the plot, the red (blue) region corresponds to the parameterizations of the model which yield the high (low) steady state most frequently. The green region represents a set of conditions whereby both steady states are equally reached.
doi:10.1371/journal.pone.0091963.g011

Parameter sweep analysis of the Ras/cAMP/PKA model

In this section we present the results of a PSA carried out on the stochastic model of the Ras/cAMP/PKA pathway [6], [44]. In *S. cerevisiae*, this pathway plays a major role in the regulation of metabolism, in stress resistance and to control the cell cycle progression [75], [76]. In [6], it was shown that intrinsic noise within the Ras/cAMP/PKA pathway can enhance the robustness of the system in response to different perturbations of the model parameters, ensuring the presence of stable oscillatory regimes, as previously investigated for other biological systems (see [77] and references therein). Indeed, stochastic simulations of the functioning of this pathway showed that yeast cells might be able to respond appropriately to an alteration of some basic components – such as the intracellular amount of pivotal proteins, that can be related to the stress level [78], [79] – fostering the maintenance of stable oscillations during the signal propagation. This behavior might suggest a stronger adaptation capability of yeast cells to various environmental stimuli or endogenous variations.

In [6], [44], in particular, we showed that the intracellular pool of guanine nucleotides (GTP, GDP), as well as the molecular amounts of protein Cdc25 – that positively regulates the activation of Ras protein, and that is negatively regulated by PKA – are both able to govern the establishment of oscillatory regimes in the dynamics of the second messengers cAMP and of protein PKA. In turn, this behavior can influence the dynamics of downstream targets of PKA, such as the periodic nucleocytoplasmic shuttling of the transcription factor Msn2 [78], [80]. In addition, in [6], [44] we highlighted that stochastic and deterministic simulations of the Ras/cAMP/PKA pathway can yield qualitatively different outcomes: in some conditions, characterized by very low amounts of pivotal proteins in this pathway (e.g., Cdc25), the stochastic approach provides stable oscillatory regimes of cAMP, while the deterministic approach shows damped oscillations. Therefore, these results remark the role played by noise in the Ras/cAMP/PKA pathway and the usefulness of executing stochastic simulations.

To deeply investigate the role played by guanine nucleotides and Cdc25, in this work we further analyzed the extended version of the Ras/cAMP/PKA model presented in [6], [44], where the reactions responsible for the occurrence of oscillatory behaviors were included. To this aim, we performed a PSA-2D to simulate the system dynamics in perturbed conditions, where we simultaneously varied the amount of GTP in the interval $[1.9\cdot 10^{4}, 5.0\cdot 10^{6}]$ molecules (corresponding to a reduced nutrient availability, up to a normal growth condition) and the amount of Cdc25 in the interval $[0, 600]$ molecules (ranging from the deletion to a 2-fold overexpression of these regulatory proteins). A total of 2^{16} different initial parameterizations were uniformly distributed over this bidimensional parameter space.

In Figure 12a we plot the amplitude of cAMP oscillations in each of these initial conditions, where an amplitude value equal to zero corresponds to a non oscillating dynamics; the amplitude values of cAMP oscillations were calculated as described in [44]. This figure shows that oscillatory regimes are established for basically any value of GTP when the amount of Cdc25 is at normal condition or slightly lower, while if the amount of Cdc25 increases, no oscillations of cAMP occur when GTP is high, but oscillatory regimes are still present if GTP is low. In order to compare the advantage of using cuTauLeaping to perform this stochastic analysis with respect to a CPU implementation, in Figure 12b we present the previous analysis performed on the CPU [6], which was obtained with a comparable computational time, albeit in the case of the CPU-based analysis only 2^{8} parameterizations of the Ras/cAMP/PKA pathway (corresponding to 2^{8} independent simulations) could be analyzed.

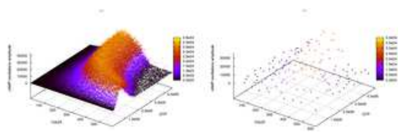


Figure 12. Bidimensional parameter sweep analysis of the Ras/cAMP/PKA model.
Results of a PSA-2D on the Ras/cAMP/PKA model by varying the amount of GTP in the interval $[1.9\cdot 10^{4}, 5.0\cdot 10^{6}]$ molecules (ranging from a reduced nutrient availability to a normal growth condition), and the amount of Cdc25 in the interval $[0, 600]$ molecules (ranging from the deletion to a 2-fold overexpression of this GEF proteins). The figure shows the amplitude of cAMP oscillations, evaluated as described in [44]; an amplitude value equal to zero corresponds to a non oscillating dynamics. (**a**) Plot of the results obtained by running 2^{16} parallel simulations with cuTauLeaping; (**b**) plot of the results obtained by running 2^{8} sequential simulations, performed on the CPU. The two batches of parallel and sequential simulations

were executed with a comparable computational time.
doi:10.1371/journal.pone.0091963.g012

These computational results can suggest possible interesting behaviors of the biological system under investigation. In this case, for instance, the establishment of oscillatory regimes in the above mentioned conditions can be due to the fact that Ras proteins are more frequently in their inactive state (that is, loaded with GDP instead of GTP) when the ratio GTP/GDP decreases. Since in normal growth conditions the concentration of GTP is 3 to 5 times higher than GDP, the decreased activity of Ras proteins in the considered perturbed conditions – which are characterized by a favored unproductive binding/unbinding with GDP – can induce the establishment of an oscillatory regime (see also [6] for more details).

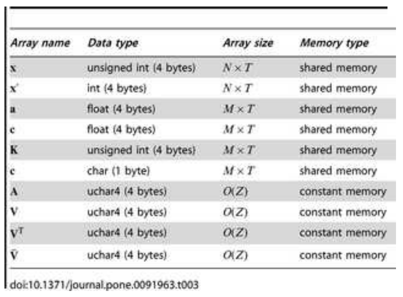
Discussion

To reduce the computational costs related to the analyses of mathematical models of real biological systems, two conceptually simple ways can be considered to parallelize stochastic simulations. The easiest solution consists in generating multiple threads on multi-core workstations, but it immediately turns out to be undersized, since the number of cores on high-end machines can be far lower than the number of simulations required for computational analysis as PE, PSA, SA and RE. The other way consists in distributing the stochastic simulations on a cluster of machines, which may as well result inadequate for several problems. First of all, it is economically expensive and very power-demanding; secondly, it takes a dedicated software infrastructure to handle workload balancing, network communication and the possible errors due to nodes downtime or server-node communication issues; thirdly, if the nodes of the cluster are heterogeneous, the slowest machines may represent a bottleneck for the whole task. In addition, a cluster implementation may not always scale well because of two problems: on the one hand, the speedup is approximately proportional to the number of independent simulations that run on a dedicated node (i.e., a million nodes for common tasks like PE and SA); on the other hand, the running time of each simulation can be larger than the overhead requested for server-node communication.

An alternative methodology to perform multiple and massively parallel simulations consists in exploiting the GPGPU architecture. The modern GPU of mid-range price contains thousands of cores that – as long as the computational task can be subdivided and optimized for a SIMD architecture – allow an impressive peak of computational power, and also a higher energetic efficiency with respect to an equivalent CPU-based solution.

Taking into account all these aspects, we developed cuTauLeaping, an implementation of tau-leaping algorithm as a set of strongly optimized CUDA kernels, able to simultaneously execute multiple independent simulations on a single machine. The specific design of cuTauLeaping presents some additional advantages: it avoids any memory transfer to and from the host, thus reducing the overall running time, and it can be embedded into the GPU-based software framework that we developed for PE [35], [81] and RE [7]. As a matter of fact, the modularity of our implementation and the mutual independence of the multiple simulations allows to easily wrap cuTauLeaping with any other methodology that needs or can benefit from these massive parallel executions.

To achieve even better performances, the tau-leaping algorithm was redesigned to (i) avoid the conditional branches, thus exploiting the underlying SIMD architecture as much as possible, and to (ii) capitalize on the CUDA's memory hierarchy, by pre-calculating some of the needed data structures and by allocating the most used ones into the fastest, yet smallest, memories. Indeed, one the biggest limiting factor for a good occupancy of the CUDA resources is a large use of the shared memory, which can improve the overall performances at the cost of reducing the theoretical occupancy of the SM. Table 3 lists the data type and size of the vectors used by cuTauLeaping. For performances reasons, these vectors are stored into the high-performance memories: vectors containing information that change during the simulation are allocated into the shared memory, while the other information are stored into the constant memory. The dimensions of these vectors are proportional to the number of threads forming a block (T), the number of reactions (M), the number of molecular species present in the system (N) and the number of non-zero entries (Z) of the corresponding non-flattened stoichiometric matrix.



Array name	Data type	Array size	Memory type
$\mathbf{x}$	unsigned int (4 bytes)	$N \times T$	shared memory
$\mathbf{x}'$	int (4 bytes)	$N \times T$	shared memory
$\mathbf{a}$	float (4 bytes)	$M \times T$	shared memory
$\mathbf{c}$	float (4 bytes)	$M \times T$	shared memory
$\mathbf{K}$	unsigned int (4 bytes)	$M \times T$	shared memory
$\mathbf{c}$	char (1 byte)	$M \times T$	shared memory
$\mathbf{A}$	uchar4 (4 bytes)	$O(Z)$	constant memory
$\mathbf{V}$	uchar4 (4 bytes)	$O(Z)$	constant memory
$\mathbf{V}^T$	uchar4 (4 bytes)	$O(Z)$	constant memory
$\bar{\mathbf{V}}$	uchar4 (4 bytes)	$O(Z)$	constant memory

doi:10.1371/journal.pone.0091963.t003

Table 3. tau-leaping data structures residing in CUDA high-performance memories.
doi:10.1371/journal.pone.0091963.t003

Specifically, according to the memory structures described in Table 3, in cuTauLeaping the exact shared memory consumption per SM for a model consisting of N molecular species and M chemical reactions, simulated by T threads per block, is equal to $T \times [13(bytes) \times M(reactions) + 8(bytes) \times N(species)]$. Since the shared memory is a limited resource on the GPU, it follows that the maximum size of a block is proportional to the size of the system, leading to the upper bound

$$T \leq \lfloor \frac{MAX_{shared}}{13M + 8N} \rfloor, \tag{5}$$

where MAX_{shared} corresponds to the amount of shared memory available on each SM for the specific architecture. According to Equation (5), on a GPU based on the Fermi architecture, the maximum size of a block for the MM, PGN, Schlögl and Ras/cAMP/PKA models corresponds to 692, 381, 646 and 63 threads, respectively. More generally, if we consider a theoretical, very large stochastic model composed by 100 reactions and 100 species, the maximum value for T would be 23; multiple blocks can be launched and run on different SMs, if these are available on the GPU, thus allowing a further level of parallelism. For instance, the GeForce GTX 590 that we used in our tests is equipped with 32 SMs and could therefore execute $23 \times 32 = 736$ simultaneous threads for this theoretical model. Since the shared memory represents a limiting factor for the parallelism, a subset of the data structures listed in Table 3 might be moved from the shared memory to the slower global memory, thus increasing the T value at the cost of higher latencies in the access to data and of higher computational costs. Being this a relevant aspect of our implementation, we are currently working on the optimization of these data structures, to the purpose of increasing the level of parallelism and further reducing the computational time.

In order to analyze the boost of performances of cuTauLeaping, we compared it with a standard CPU implementation (using the software COPASI [66] as reference), by running several identical simulations of four biological models of increasing size and complexity. Our results showed that tau-leaping running on GPU yields much better results and becomes particularly profitable when a large number of simulations have to be performed. Interestingly, when the number of simulations is limited (ranging from a few units to around one hundred, for the four models we tested), the CPU version may result more efficient than the GPU, being the break-even between the two implementations directly dependent on the complexity of the system and on its emergent dynamics. It is worth noting that, although the computational speedup achieved with cuTauLeaping might be improved by exploiting a faster RNG, such as XORWOW, the results obtained by using the more reliable RNG MRG32K3A still show a relevant reduction of running times with respect to COPASI CPU tau-leaping (see Table 2 in Text S2). Therefore, cuTauLeaping represents an advantageous tool to carry out thorough computational analyses of stochastic biological systems that usually require a huge number of simulations.

In addition, we performed different tests on randomly generated synthetic models, suggesting that the inherent complexity of the system and the chosen parameterization are more important than the model size, and they can greatly affect the performances of the simulation algorithm. The variation of the initial parameterization, which is indispensable to carry out a perturbation analysis, usually induces quantitatively and qualitatively distinct dynamical behaviors; even more important is the fact that different parameterizations generally result in different running times, leading to potentially huge and surely unfeasible computational costs, especially when standard CPU executions of stochastic simulation algorithms are performed. As an example to explain this important matter, we compared the running time of cuTauLeaping and of COPASI CPU tau-leaping when varying a single parameter in the Ras/cAMP/PKA model over 5 orders of magnitude (namely, we executed 2^{10} simulations varying the value of the stochastic constant c_3 in the sweep interval $[1.5\cdot10^{-3},1.5\cdot10]$). Figure 13 shows that, in this situation, the computational cost of tau-leaping running on CPU rapidly increases; this behavior could become prohibitive if several independent simulations need to be executed. On the contrary, cuTauLeaping shows a very moderate increase in the running times, although following the same growth trend of the CPU counterpart, and outperforms the CPU implementation. An explanation for this behavior is that, being the CPU sequential, a simulation can start as long as the previous one terminated, whilst in the case of GPU the overall running time roughly corresponds to the running time of the slowest simulation. This is particularly relevant in the case of parameterizations leading to high running times on the CPU (e.g., when $c_3=1.5\cdot10$ in Figure 13), where the speedup granted by the use of cuTauLeaping is $25\times$, compared to the $4.75\times$ speedup (see Table 1) achieved with the reference parameterization of the Ras/cAMP/PKA model (Table 6 in Text S1).

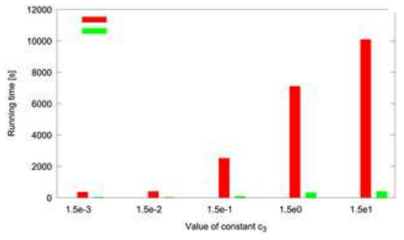


Figure 13. Performance comparison of CPU tau-leaping and cuTauLeaping for a PSA of the Ras/cAMP/PKA model.
Running times of cuTauLeaping and COPASI CPU tau-leaping to execute a PSA-1D of the Ras/cAMP/PKA model, where the stochastic constant c_3 was varied in the interval $[1.5\cdot10^{-3},1.5\cdot10]$ and a total of 2^{10} simulations were executed. The plot shows how the computational cost of tau-leaping running on CPU rapidly increases; this behavior can become prohibitive if several independent simulations need to be executed. On the contrary, cuTauLeaping shows a very moderate increase in the running times and outperforms the CPU implementation of tau-leaping.
doi:10.1371/journal.pone.0091963.g013

A *fine-grain* GPU parallelization of tau-leaping was previously proposed in [39], to the aim of accelerating the execution of single runs of tau-leaping. In that work, the computational performances were discussed in relation to very large systems of molecular interactions (with better gains achieved for 10^5 reaction channels), and tested over a *synthetically* generated network consisting of $M=N=1000$ reactions and species. Taking into account the above mentioned issues related to the model parameterization, there is a remarkable aspect that should not be left out when assessing the effective performances of fine-grain implementations of this type. Namely, the synthetic network used in [39] was characterized by a homogeneous initial parameterization (i.e., the values of all stochastic constants were randomly selected with a uniform distribution in $[0,1]$), therefore largely limiting the biochemical meaning of the distinct reaction rates that very different molecular interactions – transcription rates, post-translational modification rates, diffusion rates, catalyzed processes, etc. – do actually present in real cellular systems [4], [82]. An arbitrary modification of these values, that are of pivotal importance in the definition and in the analysis of validated models of *real* biological systems, might possibly result in different computational performances of such fine-grain GPU accelerations whenever tested on other well assessed mathematical models of biological systems [83]. This is corroborated by the results that we obtained from the analyses of randomly generated synthetic models, which altogether highlight the impact of the model parameterization on the computational performances: in particular, in cuTauLeaping the highest running times with large standard deviation values were obtained in tests 6 and 8 presented in Table 2, which are characterized by the largest intervals for the choice of stochastic constants and by the highest initial molecular amounts.

As a matter of fact, stochastic modeling and simulation methods are usually assumed to be suitable for relatively small systems (such as signaling pathways), consisting of a few tens of reactions and species, defined according to a bottom-up modeling approach whereby a mechanistic description of the most relevant molecular interactions is provided [18], [84]. The rationale behind this is that a good initial parameterization for models of this type cannot be usually settled by using either literature data or experimental measurements – especially for reaction constants, which are always difficult or even impossible to measure in living systems – and thus a large batch of simulations are generally required not only to analyze the dynamical behavior of the system, but also to corroborate the choice of the initial parameters. Therefore, despite the noticeable boost that a fine-grain GPU-based parallelization of stochastic algorithms can have in terms of single simulations, most of the times the effective requirements for the analysis of real models naturally rely upon *coarse-grain* and massively parallel executions of a large number of simulations, as proposed in this work.

Despite all these possible optimizations, stochastic simulations of complex biological systems still remain a computationally intensive task, especially when some molecular species occur in very low amounts and other in very large amounts – or also when the values of reaction constants span over different orders of magnitude – possibly inducing a slowdown of running time because of stiffness and multi-scale problems. Therefore, future efforts need to be focused on the implementation of GPU-based hybrid simulation algorithms [18], [85]–[87], able to automatically choose the subset of reactions to be simulated with a stochastic algorithm and the subset of reactions for which a deterministic simulation method is more adequate in the analysis of the same biological model.

The cuTauLeaping software is available from the authors upon request.

Supporting Information

Text S1.

Stochastic biological models.
doi:10.1371/journal.pone.0091963.s001
(PDF)

Text S2.

Comparison of the computational costs of cuTauLeaping using the random numbers generators XORWOW and MRG32K3A.
doi:10.1371/journal.pone.0091963.s002
(PDF)

Author Contributions

Conceived and designed the experiments: MSN PC DB. Performed the experiments: MSN PC. Analyzed the data: MSN PC DB. Contributed reagents/materials/analysis tools: MSN PC DP. Wrote the paper: MSN PC DB. Designed and implemented the code: MSN PC. Critically read the manuscript and contributed to the discussion of the whole work: DP GM.

References

1. Aldridge B, Burke J, Lauffenburger D, Sorger P (2006) Physicochemical modelling of cell signalling pathways. Nat Cell Biol 8: 1195–203. doi: 10.1038/ncb1497
View Article • PubMed/NCBI • Google Scholar

2. Hyduke D, Palsson B (2010) Towards genome-scale signalling network reconstructions. Nat Rev Genet 11: 297–307. doi: 10.1038/nrg2750
View Article • PubMed/NCBI • Google Scholar

3. Kitano H (2002) Computational systems biology. Nature 420: 206–210. doi: 10.1038/nature01254
View Article • PubMed/NCBI • Google Scholar

4. Papin J, Hunter T, Palsson B, Subramaniam S (2005) Reconstruction of cellular signalling networks and analysis of their properties. Nat Rev Mol Cell Biol 9: 99–111. doi: 10.1038/nrm1570
View Article • PubMed/NCBI • Google Scholar

5. Chou I, Voit E (2009) Recent developments in parameter estimation and structure identification of biochemical and genomic systems. Math Biosci 219: 57–83. doi: 10.1016/j.mbs.2009.03.002
View Article • PubMed/NCBI • Google Scholar

6. Besozzi D, Cazzaniga P, Pescini D, Mauri G, Colombo S, et al. (2012) The role of feedback control mechanisms on the establishment of oscillatory regimes in the Ras/cAMP/PKA pathway in *S. cerevisiae*. EURASIP J Bioinform Syst Biol 10. doi: 10.1186/1687-4153-2012-10
View Article • PubMed/NCBI • Google Scholar

7. Nobile MS, Besozzi D, Cazzaniga P, Pescini D, Mauri G (2013) Reverse engineering of kinetic reaction networks by means of Cartesian Genetic Programming and Particle Swarm Optimization. In: 2013 IEEE Conference on Evolutionary Computation. volume 1, pp. 1594–1601.

8. Dräger A, Kronfeld M, Ziller MJ, Supper J, Planatscher H, et al. (2009) Modeling metabolic networks in *C. glutamicum*: a comparison of rate laws in combination with various parameter optimization strategies. BMC Syst Biol 3. doi: 10.1186/1752-0509-3-5
View Article • PubMed/NCBI • Google Scholar

9. Moles CG, Mendes P, Banga JR (2003) Parameter estimation in biochemical pathways: a comparison of global optimization methods. Genome Res 13: 2467–2474. doi: 10.1101/gr.1262503
View Article • PubMed/NCBI • Google Scholar

10. Gunawan R, Cao Y, Petzold LR, Doyle FJ (2005) Sensitivity analysis of discrete stochastic systems. Biophys J 88: 2530–2540. doi: 10.1529/biophysj.104.053405
View Article • PubMed/NCBI • Google Scholar

11. Plyasunov S, Arkin A (2007) Efficient stochastic sensitivity analysis of discrete event systems. J Comput Phys 221: 724–738. doi: 10.1016/j.jcp.2006.06.047
View Article • PubMed/NCBI • Google Scholar

12. Zhang HX, Goutsias J (2011) Reducing experimental variability in variance-based sensitivity analysis of biochemical reaction systems. J Chem Phys 134: 114105. doi: 10.1063/1.3563539
View Article • PubMed/NCBI • Google Scholar

13. Chis OT, Banga J, Balsa-Canto E (2011) Identifiability of systems biology models: A critical comparison of methods. PLOS ONE 6: e27755. doi: 10.1371/journal.pone.0027755
View Article • PubMed/NCBI • Google Scholar

14. Raue A, Becker V, Klingmüller U, Timmer J (2010) Identifiability and observability analysis for experimental design in nonlinear dynamical models. Chaos 20: 045105. doi: 10.1063/1.3528102
View Article • PubMed/NCBI • Google Scholar

15. Srinath S, Gunawan R (2010) Parameter identifiability of power-law biochemical system models. J Biotechnol 149: 132–140. doi: 10.1016/j.jbiotec.2010.02.019
View Article • PubMed/NCBI • Google Scholar

16. Kitagawa J, Iba H (2003) Identifying metabolic pathways and gene regulation networks with evolutionary algorithms. In: Evolutionary Computation in Bioinformatics, Morgan Kaufmann, chapter 12. pp. 255–278.

17. Elowitz M, Levine A, Siggia E, Swain P (2002) Stochastic gene expression in a single cell. Science 297: 1183–1186. doi: 10.1126/science.1070919
View Article • PubMed/NCBI • Google Scholar

18. Wilkinson D (2009) Stochastic modelling for quantitative description of heterogeneous biological systems. Nat Rev Genet 10: 122–133. doi: 10.1038/nrg2509
View Article • PubMed/NCBI • Google Scholar

19. Gillespie DT (2000) The chemical Langevin equation. J Chem Phys 113: 297. doi: 10.1063/1.481811
View Article • PubMed/NCBI • Google Scholar

20. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81: 2340–2361. doi: 10.1021/j100540a008
View Article • PubMed/NCBI • Google Scholar

21. Gillespie DT (1992) A rigorous derivation of the chemical master equation. Physica A 188: 404–425. doi: 10.1016/0378-4371(92)90283-v
View Article • PubMed/NCBI • Google Scholar

22. Gibson M, Bruck J (2000) Efficient exact stochastic simulation of chemical systems with many species and many channels. J Phys Chem A 104: 1876–1889. doi: 10.1021/jp993732q
View Article • PubMed/NCBI • Google Scholar

23. Li H, Petzold LR (2006) Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. Technical report, Department of Computer Science, University of California Santa Barbara.

24. Slepoy A, Thompson A, Plimpton S (2008) A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical networks. J Chem Phys 128: 205101. doi: 10.1063/1.2919546
View Article • PubMed/NCBI • Google Scholar

25. Gillespie DT (2001) Approximate accelerated stochastic simulation of chemically reacting systems. J Chem Phys 115: 1716–1733. doi: 10.1063/1.1378322
View Article • PubMed/NCBI • Google Scholar

26. Tian T, Burrage K (2004) Binomial leap methods for simulating stochastic chemical kinetics. J Chem Phys 121: 10356. doi: 10.1063/1.1810475
View Article • PubMed/NCBI • Google Scholar

27. Cao Y, Gillespie DT, Petzold LR (2006) Efficient step size selection for the tau-leaping simulation method. J Chem Phys 124: 044109. doi: 10.1063/1.2159468
View Article • PubMed/NCBI • Google Scholar

28. Cao Y, Gillespie DT, Petzold LR (2007) The adaptive explicit-implicit tau-leaping method with automatic tau selection. J Chem Phys 126: 224101. doi: 10.1063/1.2745299
View Article • PubMed/NCBI • Google Scholar

29. Marquez-Lago TT, Burrage K (2007) Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics. J Chem Phys 127: 104101. doi: 10.1063/1.2771548
View Article • PubMed/NCBI • Google Scholar

30. Leier A, Marquez-Lago T, Burrage K (2008) Generalized binomial *t* -leap method for biochemical kinetics incorporating both delay and intrinsic noise. J Chem Phys 128: 205107. doi: 10.1063/1.2919124
View Article • PubMed/NCBI • Google Scholar

31. Tian T, Burrage K (2005) Parallel implementation of stochastic simulation of large-scale cellular processes. In: 8th International Conference on High-Performance Computing in Asia-Pacific Region. pp. 621–626.

32. Burrage K, Burrage P, Hamilton N, Tian T (2006) Compute-intensive simulations for cellular models. In: Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies. pp. 79–119.

33. Kent E, Hoops S, Mendes P (2012) Condor-COPASI: high-throughput computing for biochemical networks. BMC Syst Biol 6: 91. doi: 10.1186/1752-0509-6-91
View Article • PubMed/NCBI • Google Scholar

34. Macchiarulo L (2008) A massively parallel implementation of Gillespie algorithm on FPGAs. In: International Conference of the IEEE on Engineering in Medicine and Biology Society. pp. 1343–1346.

35. Nobile MS, Besozzi D, Cazzaniga P, Mauri G, Pescini D (2012) A GPU-based multi-swarm PSO method for parameter estimation in stochastic biological systems exploiting discrete-time target series. In: Giacobini M, Vanneschi L, Bush W, editors, Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, Springer, volume 7246 of *LNCS*. pp. 74–85.

36. Li H, Petzold LR (2010) Efficient parallelization of the stochastic simulation algorithm for chemically reacting systems on the Graphics Processing Unit. Int J High Perform Comput Appl 24: 107–116. doi: 10.1177/1094342009106066
View Article • PubMed/NCBI • Google Scholar

37. Klingbeil G, Erban R, Giles M, Maini PK (2011) STOCHSIMGPU: parallel stochastic simulation for the Systems Biology Toolbox 2 for MATLAB. Bioinformatics 27: 1170–1171. doi: 10.1093/bioinformatics/btr068
View Article • PubMed/NCBI • Google Scholar

38. Vigelius M, Lane A, Meyer B (2010) Accelerating reaction-diffusion simulations with General-Purpose Graphics Processing Units. Bioinformatics 27: 288–290. doi: 10.1093/bioinformatics/btq622
View Article • PubMed/NCBI • Google Scholar

39. Komarov I, D'Souza RM, Tapia J (2012) Accelerating the Gillespie *t* -leaping method using Graphics Processing Units. PLOS ONE 7: e37370. doi: 10.1371/journal.pone.0037370
View Article • PubMed/NCBI • Google Scholar

40. Nelson D, Cox M (2004) Lehninger Principles of Biochemistry. W. H. Freeman Company.

41. Wang Y, Christley S, Mjolsness E, Xie X (2010) Parameter inference for discretely observed stochastic kinetic models using stochastic gradient descent. BMC Syst Biol 4: 99. doi: 10.1186/1752-0509-4-99
View Article • PubMed/NCBI • Google Scholar

42. Schlögl F (1972) Chemical reaction models for non-equilibrium phase transitions. Z Physik 253: 147–161. doi: 10.1007/bf01379769
View Article • PubMed/NCBI • Google Scholar

43. Cazzaniga P, Pescini D, Besozzi D, Mauri G, Colombo S, et al. (2008) Modeling and stochastic simulation of the Ras/cAMP/PKA pathway in the yeast *Saccharomyces cerevisiae* evidences a key regulatory function for intracellular guanine nucleotides pools. J Biotechnol 133: 377–385. doi: 10.1016/j.jbiotec.2007.09.019
View Article • PubMed/NCBI • Google Scholar

44. Pescini D, Cazzaniga P, Besozzi D, Mauri G, Amigoni L, et al. (2012) Simulation of the Ras/cAMP/PKA pathway in budding yeast highlights the establishment of stable oscillatory states. Biotechnol Adv 30: 99–107. doi: 10.1016/j.biotechadv.2011.06.014
View Article • PubMed/NCBI • Google Scholar

45. Cho Y, Ramakrishnan N, Cao Y (2008) Reconstructing chemical reaction networks: data mining meets system identification. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 142–150.

46. van Kampen N (2001) Stochastic Processes in Physics and Chemistry. Elsevier Science, Amsterdam.

47. Stewart W (2010) Introduction to the Numerical Solution of Markov Chains. Princeton University Press.

48. Zhang J, Watson L, Cao Y (2007) A modified uniformization method for the solution of the Chemical Master Equation. Technical Report TR-07-31, Computer Science, Virginia Tech.

49. Sidje R, Burrage K, MacNamara S (2007) Inexact uniformization method for computing transient distributions of Markov Chains. SIAM J Sci Comput 29: 2562–2580. doi: 10.1137/060662629
View Article • PubMed/NCBI • Google Scholar

50. Hellander A (2008) Efficient computation of transient solutions of the Chemical Master Equation based on uniformization and quasi-Monte Carlo. J Chem Phys 128: 154109. doi: 10.1063/1.2897976
View Article • PubMed/NCBI • Google Scholar

51. Burrage K, Hegland M, MacNamara F, Sidje B (2006) A Krylov-based finite state projection algorithm for solving the Chemical Master Equation arising in the discrete modelling of biological systems. In: Langville A, Stewart W, editors, Proceedings of the Markov 150th Anniversary Conference. Boston Books, Charleston, South Carolina, pp. 21–38.

52. Munsky B, Khammash M (2006) The finite state projection algorithm for the solution of the Chemical Master Equation. J Chem Phys 124: 044104. doi: 10.1063/1.2145882
View Article • PubMed/NCBI • Google Scholar

53. Wolf V, Goel R, Mateescu M, Henzinger TA (2010) Solving the Chemical Master Equation using sliding windows. BMC Syst Biol 4: 42. doi: 10.1186/1752-0509-4-42
View Article • PubMed/NCBI • Google Scholar

54. Jahnke T, Huisinga W (2007) Solving the Chemical Master Equation for monomolecular reaction systems analytically. J Math Biol 54: 1–26. doi: 10.1007/s00285-006-0034-x
View Article • PubMed/NCBI • Google Scholar

55. Gillespie DT (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J Comput Phys 22: 403–434. doi: 10.1016/0021-9991(76)90041-3
View Article • PubMed/NCBI • Google Scholar

56. Gillespie DT (2008) Simulation methods in Systems Biology. In: Bernardo M, Degano P, Zavattaro G, editors, Formal Methods for Computational Systems Biology, Springer, volume 5016 of *LNCS*. pp. 125–167.

57. NVIDIA (2011) Tuning CUDA applications for Fermi.

58. NVIDIA (2012) Nvidia CUDA C Programming Guide. Available: http://docs.nvidia.com/cuda/-cuda-c-programming-guide/index.html.

59. NVIDIA (2012) CUDA Toolkit 5.0 CURAND Guide.

60. Marsaglia G (2003) Xorshift RNGs. J Stat Softw 8: 1–6.
View Article • PubMed/NCBI • Google Scholar

61. L'Ecuyer P, Simard R, Chen EJ, Kelton WD (2002) An object-oriented random-number package with many long streams and substreams. Oper Res 50: 1073–1075. doi: 10.1287/opre.50.6.1073.358
View Article • PubMed/NCBI • Google Scholar

62. Matsumoto M, Nishimura T (1998) Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans Model Comput Simul 8: 3–30. doi: 10.1145/272991.272995
View Article • PubMed/NCBI • Google Scholar

63. Nandapalan N, Brent RP, Murray LM, Rendell AP (2012) High-performance pseudo-random number generation on Graphics Processing Units. In: Parallel Processing and Applied Mathematics, Springer. pp. 609–618.

64. Hill DRC, Mazel C, Passerat-Palmbach J, Traore MK (2013) Distribution of random streams for simulation practitioners. Concurr Comp-Pract E 25: 1427–1442. doi: 10.1002/cpe.2942
View Article • PubMed/NCBI • Google Scholar

65. L'Ecuyer P, Simard R (2007) Testu01: A C library for empirical testing of random number generators. ACM T Math Software 33. doi: 10.1145/1268776.1268777
View Article • PubMed/NCBI • Google Scholar

66. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, et al. (2006) COPASI - a COmplex PAthway SImulator. Bioinformatics 22: 3067–3074. doi: 10.1093/bioinformatics/btl485
View Article • PubMed/NCBI • Google Scholar

67. Wen-mei WH (2011) GPU Computing Gems Jade Edition. Morgan Kaufmann doi: 10.1016/b978-0-12-385963-1.00046-0
View Article • PubMed/NCBI • Google Scholar

68. Carrillo S, Siegel J, Li X (2009) A control-structure splitting optimization for GPGPU. In: Proc. of the 6th ACM conference on Computing frontiers. New York, NY, USA: ACM, CF '09, pp. 147–150.

69. Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, et al.. (2008) Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proc. of the 13th ACM SIGPLAN Symposium on principles and practice of parallel programming. New York, NY, USA: ACM, PPoPP '08, pp. 73–82.

70. Vellela M, Qian H (2009) Stochastic dynamics and non-equilibrium thermodynamics of a bistable chemical system: the Schlogl model revisited. J R Soc Interface 6: 925–940. doi: 10.1098/rsif.2008.0476
View Article • PubMed/NCBI • Google Scholar

71. Wilhelm T (2009) The smallest chemical reaction system with bistability. BMC Syst Biol 3: 90. doi: 10.1186/1752-0509-3-90
View Article • PubMed/NCBI • Google Scholar

72. Craciun G, Tang Y, Feinberg M (2006) Understanding bistability in complex enzyme-driven reaction networks. PNAS 103: 8697–8702. doi: 10.1073/pnas.0602767103
View Article • PubMed/NCBI • Google Scholar

73. Pomerening JR (2008) Uncovering mechanisms of bistability in biological systems. Curr Opin Biotech 19: 381–388. doi: 10.1016/j.copbio.2008.06.009
View Article • PubMed/NCBI • Google Scholar

74. Widder S, Macía J, Solé R (2009) Monomeric bistability and the role of autoloops in gene regulation. PLOS ONE 4: e5399. doi: 10.1371/journal.pone.0005399
View Article • PubMed/NCBI • Google Scholar

75. Santangelo GM (2006) Glucose signaling in *Saccharomyces cerevisiae*. Microbiol Mol Bio Rev 70: 253–282. doi: 10.1128/mmbr.70.1.253-282.2006
View Article • PubMed/NCBI • Google Scholar

76. Zaman S, Lippman SI, Schneper L, Slonim N, Broach JR (2009) Glucose regulates transcription in yeast through a network of signaling pathways. Mol Syst Biol 5: 1–14. doi: 10.1038/msb.2009.2
View Article • PubMed/NCBI • Google Scholar

77. Steuer R, Zhou C, Kurths J (2003) Constructive effects of fluctuations in genetic and biochemical regulatory systems. BioSystems 72: 241–251. doi: 10.1016/j.biosystems.2003.07.001
View Article • PubMed/NCBI • Google Scholar

78. Medvedik O, Lamming D, Kim K, Sinclair D (2007) MSN2 and MSN4 link calorie restriction and TOR to Sirtuin-mediated lifespan extension in *Saccharomyces cerevisiae*. PLoS Biol 5: 2330–2341. doi: 10.1371/journal.pbio.0050261
View Article • PubMed/NCBI • Google Scholar

79. Wang L, Renault G, Garreau H, Jacquet M (2004) Stress induces depletion of Cdc25p and decreases the cAMP producing capability in *Saccharomyces cerevisiae*. Microbiology 150: 3383–91. doi: 10.1099/mic.0.27162-0
View Article • PubMed/NCBI • Google Scholar

80. Garmendia-Torres C, Goldbeter A, Jacquet M (2007) Nucleocytoplasmic oscillations of the yeast transcription factor Msn2: Evidence for periodic PKA activation. Current Biol 17: 1044–9. doi: 10.1016/j.cub.2007.05.032
View Article • PubMed/NCBI • Google Scholar

81. Nobile MS, Besozzi D, Cazzaniga P, Mauri G, Pescini D (2012) Estimating reaction constants in stochastic biological systems with a multi-swarm PSO running on GPUs. In: Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference Companion. New York, NY, USA: ACM, GECCO Companion '12, pp. 1421–1422.

82. Lipkow K, Andrews SS, Bray D (2005) Simulated diffusion of phosphorylated CheY through the cytoplasm of *Escherichia coli*. J Bacteriol 187: 45–53. doi: 10.1128/jb.187.1.45-53.2005
View Article • PubMed/NCBI • Google Scholar

83. Li C, Donizelli M, Rodriguez N, Dharuri H, Endler L, et al. (2010) BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. BMC Syst Biol 4: 92. doi: 10.1186/1752-0509-4-92
View Article • PubMed/NCBI • Google Scholar

84. Stelling J (2004) Mathematical models in microbial systems biology. Curr Opin Microbiol 7: 513–518. doi: 10.1016/j.mib.2004.08.004
View Article • PubMed/NCBI • Google Scholar

85. Salis H, Kaznessis Y (2005) Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. J Chem Phys 122: 054103. doi: 10.1063/1.1835951
View Article • PubMed/NCBI • Google Scholar

86. Pahle J (2009) Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. Brief Bioinform 10: 53–64. doi: 10.1093/bib/bbn050
View Article • PubMed/NCBI • Google Scholar

87. Besozzi D, Caravagna G, Cazzaniga P, Nobile MS, Pescini D, et al.. (2013) GPU-powered simulation methodologies for biological systems. In: Graudenzi A, Caravagna G, Mauri G, Antoniotti M, editors, Proceedings of Wivace 2013 - Italian Workshop on Artificial Life and Evolutionary Computation. volume 130 of *EPTCS*, pp. 87–91.