Massively-Parallel Simulation of Biochemical Systems

Jens Ackermann Paul Baecher Thorsten Franzel
Michael Goesele* Kay Hamacher†

Technische Universität Darmstadt

Abstract:
Understanding biological evolution prompts for a detailed understanding of the realized phenotype. Biochemical and gene regulatory dynamics are a cornerstone for the physiology of the cell and must therefore be regarded as one of the major aspects of such a phenotype. Experimental insight into molecular parameters is, however, hard to come by. Model development therefore requires computational parameter estimation. At the same time, design of cellular dynamics is highly efficient when done in-silico. We therefore developed a computational approach to allow for massively parallel simulation of biological molecular networks that leverage the massively-parallel computing power of modern graphics cards and other many-core programming paradigms. Our system can automatically compile standard SBML files into CUDA code, using analytic derivatives, and computing standard measures of complex dynamics like the Lyapunov exponent.

1 Introduction

In the post genome area it became common wisdom that genomic information alone is not sufficient for understanding cellular function, organismic evolution, or ecological dynamics. Instead, we need to also look at dynamical features, e.g., the cellular dynamics [Iye09]. At the same time, biology is on its way to become a pure quantitative science [BW06]. Together, these two developments created a need for new approaches.

Accordingly, the network paradigm [DM03] has in recent years dominated models and concepts in theoretical biology [vR06]. These networks describe the topology of the underlying biochemical dynamics, which in fact are ordinary differential equations (ODEs). Such models are nowadays accessible in the SBML modeling language [HFS+03] from databases such as the EMBL BioModels database [LNBB+06].

In particular, understanding the available parameter space of these ODEs [DCS+09] is of greatest interest in order to understand the evolution of cellular physiology [Lyn07] and to design biotechnological applications [IRN+04]. Such parameter spaces are generally high-dimensional and pose therefore a difficult optimization problem [Ham05].

*Corresponding author: GRIS, Fraunhoferstrasse 5, TU Darmstadt, Darmstadt, Germany
†Corresponding author: Bioinformatics and Theo. Biology, Institute for Genetics & Microbiology, TU Darmstadt, Darmstadt, Germany

Both questions prompt for efficient procedures to sample the parameter space—either adaptively to understand evolutionary dynamics or by optimization to design cellular circuits. Currently, there are two types of tools available ot sample the parameter space. On one hand, tools with graphical user interfaces such as Copasi [HSG+06] or BioNessie [LG09] allow the unexperienced modeler to perform simulations for a couple of parameter sets. They are, however, not suitable for efficient sweeping of large parameter ranges. On the other hand, command line-based tools such as the SBML ODE Solver Library [MFM+06] can perform such sweeps but require large compute clusters for a sufficient sampling.

Since evaluating an ODE for a large number of parameters is an embarrassingly parallel problem (i.e., all evaluations are completely independent) that can be solved in a single instruction multiple data (SIMD) fashion [MSM04], it is ideally suited for a massively-parallel implementation on recent general purpose graphics processors (GPUs). This has been demonstrated by two systems for numerical integration of ODEs, which were recently introduced [JK09, LP09].

Our goal is therefore to automatically create all the necessary code to run simulations given a description of the ODE system in an XML language such as SBML, simulate a large number of systems with varying parameters, and obtain measures that are of interest to the evolutionary biologist and the bio-engineer. We demonstrate here a first step, which specialializes on the automatic creation of CUDA [NVI08] code and simple, non-adaptive parameter space scans. We describe our experiences and hint at possible future developments.

In particular, our contributions are:

- Given the description of a system of ODEs in SBML, we automatically differentiate the rate equations within the model. Based on these derivatives, we then automatically construct massively-parallel CUDA code which is able to simulate the system.
- We analyze online the computed complex dynamics of the system via the leading Lyapunov exponent. Simulations are performed in a combinatorial fashion over the parameter hypercube.
- We demonstrate the efficiency of the resulting CUDA code empirically and analyze the speedup compared to a CPU implementation of the system.

2 Related Work

2.1 SBML-based ODE Solvers

In recent years, developing ODE solvers for networks of biological and biochemical entities has been an active area of research in computational biology [LCPG08]. While there are general numerical approaches to solve differential equations, typically based on hand-written code [W.H95, JK09, LP09], we focus here on systems that have a direct interface

to SBML [HFS+03].

Copasi [HSG+06] can import SBML files and offers a sophisticated graphical user interface (GUI). It enables even unexperienced users to simulate systems intuitively on a desktop computer. Similarly, BioNessie [LG09] is a GUI-based, platform-independent biochemical network simulator. Its compute core is provided by the SOSlib (SBML ODE Solver Library) [MFM+06]. This library includes also command-line driven tools to analyze systems of ordinary differential equations derived from chemical reaction networks. Simulations for large sets of parameters can thus be performed even on large clusters that provide sufficient compute power. In contrast, we propose to use the compute power of modern GPUs which offer high performance at moderate cost.

2.2 Automatic Code Generation

Automatic code generation has a long tradition in computer science and is an active area of research. Some of the tools mentioned above, in particular Copasi [HSG+06] and the SOSlib [MFM+06], are capable of transforming SBML files into C code. Their approach has, however, two shortcomings.

First, the generated code is not CUDA-ready and can therefore not directly execute on a GPU. There exist, however, examples of automatic CUDA code generation [HB09] for specific tasks outside of systems biology.

Second, these approaches do not generate efficient, analytical derivatives for analysis procedures such as the computation of Lyapunov exponents. E.g., Copasi uses a numerical approach [WSSV85] to compute Lyapunov exponents. In contrast, we employ a computer algebra system [RED] to automatically compute analytical derivates from the SBML input. This yields the additional benefit that it can in principle leverage any optimization and complexity reduction techniques of the computer algebra system to improve efficiency of the resulting code.

3 Methods

We implemented a pipeline that automatically creates an executable from a model described in the Systems Biology Markup Language (SBML) [HFS+03]. This executable uses NVIDIA's CUDA framework [NVI08] to simulate thousands of dynamic systems in parallel.

A biological model contained in a SBML file is first transformed into CUDA code with some additional keywords that control a computer algebra system. This augmented CUDA code is processed by the computer algebra system which replaces the special keywords with CUDA code for the Jacobian matrix of the system. Together with a CUDA based ODE solver, this is compiled into the final program that executes the simulation for a multitude of parameter configurations in parallel. The implemented pipeline is shown

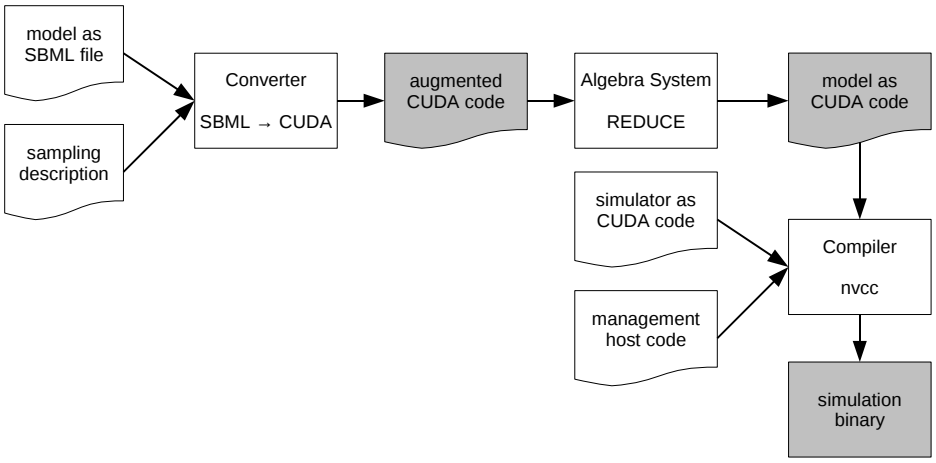

Figure 1: Overview over our pipeline. Gray boxes indicate automatically created files, while white parts are fixed elements.

in Figure 1 and a snipplet of code and its transformation from SBML to CUDA code is depicted in Figure 2.

3.1 Conversion to CUDA

To achieve a highly efficient simulation on the GPU, the equations describing the model have to be directly integrated into the CUDA program. Furthermore, the computation of leading Lyapunov exponents requires the Jacobi matrix of the system for each time step of the simulation. Like the differential equations, this has to be evaluated on the GPU, but a numerical differentiation based on finite differences is neither stable nor efficient enough. We therefore propose a preprocessing step to compute analytic derivatives.

In more detail, the first stage of our implementation uses the SBML API [BKJH08] to process a SBML file and extract the mathematical formulas. These are internally represented as abstract syntax trees that we automatically convert into instructions for the REDUCE [RED] computer algebra system. These instructions represent the converted mathematical formulas and commands to compute the analytic derivatives.

The translation tool generates a skeleton `model.cu_red` of a CUDA file and inserts the REDUCE statements framed by some special keywords. This skeleton includes function definitions as well as initial values for the model that can be extracted from the SBML file. Furthermore, it contains a description of the parameter space sampling that will be performed. Since the SBML code does not include such sampling information, it must be supplied by the user as a separate file.

A script then loads the augmented skeleton file into the REDUCE system and calls the

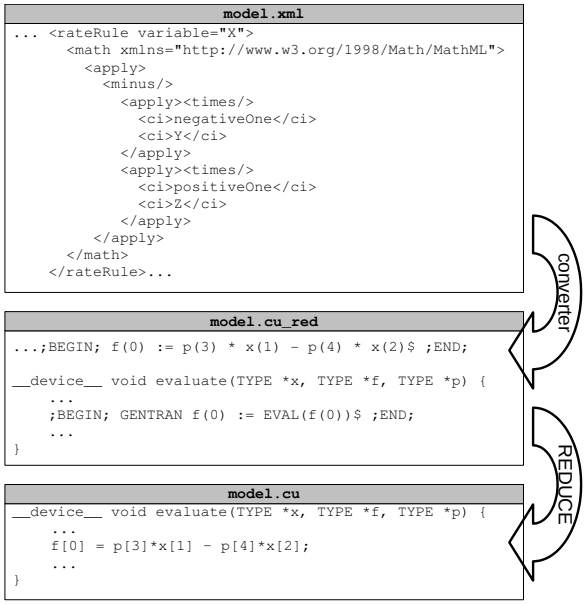

Figure 2: Flow of code through the various steps of our pipeline.

GENTRAN package, which is part of the REDUCE distribution. This package replaces the keywords with the results from the REDUCE commands and translates them into C code. As we pay attention to only insert REDUCE instructions at places, where CUDA is equal to C code, this results in a pure CUDA file `model.cu`.

3.2 Building the Simulation Executable

The problem of simulating a multitude of dynamical systems that all correspond to shared mathematical expressions (ODEs) just with a different set of parameters needs to be mapped to the CUDA architecture (please refer to the CUDA manual [NVI08] for a detailed description). In our approach, we designate one thread for each parameter set, i.e., each thread simulates the complete model for a fixed parameter vector.

We integrate the sytem with an Euler scheme and compute a deviation vector to determine the Lyapunov exponent with the algorithm described in Benettin et al. [BGS76] and Alligood et al. [ASY97]. All computations are hereby performed on the GPU. In contrast to the automatically created code from SBML, both the Euler method and the calculation of the Lyapunov exponent are independent of the specific model. They are therefore supplied as a fixed part of the pipeline along with some management code running on the CPU.

All fixed and automatically created components are compiled into a single executable. Several parameters such as the stepsize of the integration and number of time steps can

be set on the command line. Recompilation only becomes necessary, when a different parameter space sampling is needed.

3.3 CUDA Execution Configuration

In the CUDA programming model, threads are organised as blocks that are themselves grouped into grids [NVI08]. As the threads have to share resources like registers and shared memory on the multiprocessor, the number of threads per block is determined by the resource usage of the individual threads and the capabilities of the graphics device.

The amount of registers and memory needed per thread depends heavily on the number of ODEs describing the biological model. Therefore, the general simulation code has to provide some means to automatically detect the prefered block size for the actual model that it is used upon. The host code currently accomplishes this by decreasing the block size at runtime until the kernel is successfully started.

To make use of modern graphics card configurations featuring multiple GPUs on the same board, the host part of our implementation divides the entire set of samples S into smaller sets $S_{\mathrm{GPU}1}, \ldots, S_{\mathrm{GPU}K}$ that are distributed among all GPUs. Given the blocksize and the set $S_{\mathrm{GPU}i}$ of samples for a single GPU, the number of threads needed may still exceed the maximum grid size. To cope with this, we further divide $S_{\mathrm{GPU}i} = S_{\mathrm{GPU}i,1} \cup \ldots \cup S_{\mathrm{GPU}i,l}$ and sequentially start grids for all the $S_{\mathrm{GPU}i,j}$. Supporting multiple GPUs also has the advantage of simple adaption to a possible cluster of CUDA enabled computers.

3.4 Implementation Details

We currently use REDUCE 3.8 as of 27.02.2009 and the GENTRAN package shipped with that release. The conversion tool supports SBML Level 2 Version 3 via the API version 3.2.0.

Evaluations are performed on two distinct computers. Both are equipped with Intel Xeon Quad-Core processors (E5430, 64 bit, 2.66 GHz) and run OpenSuSE 11.1. One machine has a GeForce GTX 280 as graphics card and gcc 4.1.2 (20061115, prerelease) as compiler. The other features a GeForce 9800 GX2 and uses gcc 4.3.2 (revision 141291) for compiling. The GeForce 9800 GX2 contains internally two separate devices and appears to the host PC as double GPU system. We use nvcc release 2.1 (V0.2.1221) for CUDA compilation on both machines and driver version 180.22.

4 Results

For the purpose of this paper, we simulated an abstract system proposed by Rössler [Rös77], which ultimatively leads to the ordinary differential equation system of the well-known

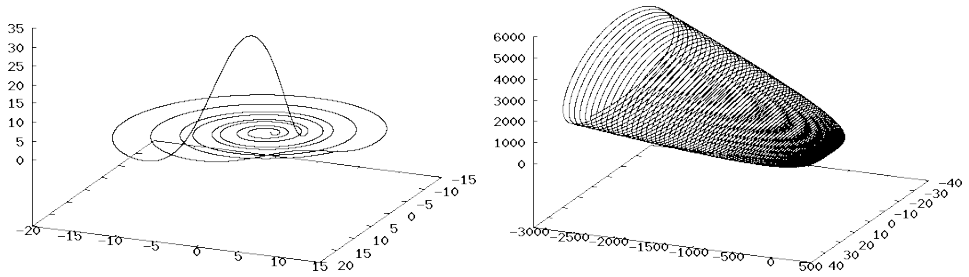

Figure 3: Time series plots with 10^6 steps for the Rössler attractor shown in Equation 1 computed on the GPU. Only a subset of the computed points is shown. *Left:* Plot for the original parameter set $(a, b, c) = (0.15, 0.2, 10.0)$ as proposed by Rössler [Rös77]. *Right:* Plot for an arbitrary chosen parameter set $(a, b, c) = (0.2, 0.0, 0.0)$.

Rössler attractor

$$\frac{dx}{dt} = -y - z \qquad \frac{dy}{dt} = x + ay \qquad \frac{dz}{dt} = b + z(x - c). \tag{1}$$

The parameters a, b, and c can be interpreted as effective reaction rates for biochemical reaction networks. Figure 3 shows some time series obtained on the GPU for two different parameter sets examplifying the rich dynamics of this chaotic system.

In the following, we will integrate this system using massively-parallel computations over a hypercube in parameter-space. This enables the efficient investigation of parameter influences on the underlying system dynamics. In particular, we compute the largest Lyapunov exponent of the N-step simulated trajectories to quantify the degree of chaotic behavior observed in this system. The Lyapunov exponent is computed using the direct method of Alligood et al. [ASY97].

We note that the solution of the equation system (1) is non-bounded for sections of the parameter space. Consequently, we observed numerical problems such as NaNs for parameter sets (a, b, c) in these ranges. We also found deviations at the beginning of the simulated trajectory between the CPU and the GPU version. This is in accordance with the different implementations of float-pointing numbers on GPUs and CPUs [NVI08].

Figure 4 shows iso-surfaces of Lyapunov-exponents $\mathcal{L}$ in the parameter space of the Rössler attractor of Equation 1. Of particular interest is the iso-surface $\mathcal{L} = 0$, which divides parameter sets resulting in chaotic behavior from those that lead to non-chaotic, predictable dynamics. The iso-surfaces for other values of the Lyapunov exponent ($\mathcal{L} = -0.1$ and $\mathcal{L} = 0.2$) are consistent with the boundary between chaos and non-chaos since they are "below" and "above" the iso-surface for $\mathcal{L} = 0$.

5 Performance Evaluation

Table 1 summarizes the compute time required by several versions of our code. In all cases, we simulated 10^6 different Rössler systems on the GPU with $5 \cdot 10^6$ iterations

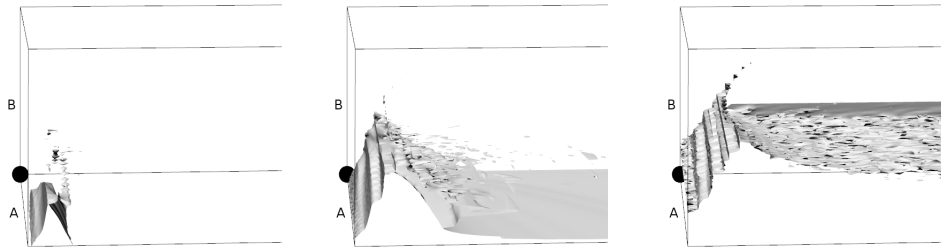

Figure 4: Iso-surfaces in parameter space of the Rössler attractor for different thresholds. *Left:* Threshold $\mathcal{L} = -0.1$. *Middle:* Threshold $\mathcal{L} = 0$ marking the boundary between chaotic and non-chaotic behavior. *Right:* Threshold $\mathcal{L} = 0.2$. In all images, the sphere marks the origin of the coordinate system. The c axis has been scaled down by a factor of 10 relative to the other axes.

code	**GPU/CPU**	**number of multiprocessors**	**threads per block**	**time** $\times 10^3$ **s**
a	9800 GX2 single	16	256	12.22
b	9800 GX2 full	2×16	256	6.13
c	9800 GX2 full	2×16	312	5.69
d	9800 GX2 full	2×16	320	5.55
e	9800 GX2 full	2×16	128	6.14
f	9800 GX2 full, shared memory	2×16	32	6.26
g	GTX 280	30	512	5.81
CPU	Xeon 2.66 GHz	—	—	351.42

Table 1: Timings for various execution configurations on different GPUs and for CPU reference code. The GeForce 9800 GX2 contains two physical GPUs with 16 multiprocessors per GPU. Both GPUs are used except for Scenario a. The GeForce GTX 280 contains 30 multiprocessors on a single GPU but belongs to a newer hardware generation.

per parameter set and computed the Lyapunov exponent of the trajectory. The parameter ranges are 100 samples for both, a and b, evenly spaced between 0 and 0.99; and 100 samples for c evenly spaced between 0 and 49.5.

The CPU version was implemented also in single precision and we did not leverage SIMD/SMT techniques, but instead compared to and compute speedups for single-core results. We used the code on the CPU as we compiled for the GPU, but omitted CUDA statements, thus comparing the same extent of code optimization. The codes differed only in initialization and I/O parts.

Table 1a shows our standard code running on the GeForce 9800 GX2 using only one of its devices. Using both devices (Table 1b) doubles the speed. All parameters were selected fully automatically. In addition, we manually ran the code with different execution configurations (Table 1c–e) but observed no significant difference in performance. This holds also for Table 1f where we additionally used the shared memory available on the multiprocessor. Overall, the execution configuration seems to have an almost negligible influence on the performance of the system (with a slight preference for larger block sizes)

and can therefore be chosen as necessary.

The remaining entries in Table 1 show the timing for the GTX 280, a newer generation GPU, and a CPU implementaion. The speedup between the CPU version and the GPU version is ~ 59.3 (calculated with the average speed of the GPU implementations). This confirms our assumption that the GPU provides an efficient and cost-effective simulation environment compared to a standard PC.

Note that - using SIMD extensions and all four cores of a recent CPU - one would obtain a theoretical, maximal speedup of 16.

6 Conclusion

We developed an automatic SBML-to-CUDA pipeline that allows for massively-parallel simulation of sufficiently large portions of the parameter spaces of differential equations relevant to systems biology and biochemical reaction networks. Using Lyapunov exponents as a standard way to analyze time series we are able to scan the parameter space of the Rössler attractor and find the boundary surface between chaotic and non-chaotic dynamics. The computation of the Lyapunov exponents uses analytical derivatives of the underlying ODEs and is thus highly stable and efficient.

The parallelization on the GPU showed a speedup of about factor 59 compared to a CPU implementation executed on a standard PC. The performance did only depend to a small extend on the execution configuration which is most likely caused by the nature of the problem which is well suited for computation on the GPU.

We note that the main computational costs arise from evaluating non-linear terms in the ODE systems. Therefore no additional omptimization is to be expected from employing more efficient parallelization schemes based on parallelizing the Euler method itself.

6.1 Limitations

This paper presents a proof-of-concept implementation of our proposed pipeline which has two main limitations.

First, the conversion tool currently only supports a small subset of SBML. In addition, many SBML files (e.g., files created by other tools) violate the SBML standard and omit information such as units for all mathematical entities. We plan to address these issues in the future by extending the supported subset of SBML and by increasing the robustness of the converter (e.g., by guessing reasonable default values for undefined units).

Second, our system is currently limited to models with eight or less ODEs and about five to twenty parameters. Larger dimensions can be simulated as well but result in higher register usage per thread. Variables are then stored in local memory which can yield a significant performance loss (depending on how well CUDA can hide memory latency). While this

is a principle problem of our approach, we expect that future hardware generations will provide more local resources per multiprocessor enabling efficient simulation of larger systems.

7 Future Work

The proposed system is now capable of automatically transforming SBML files to CUDA and naivly scanning the parameter space. In the future, we plan to perform further optimizations including optimization of the generated C and CUDA code and simplification of the mathematical expressions. We also plan to apply the converter to larger systems and to investigate the achievable speedup for such cases. Lastly, we will incorporate improved schemes to sample the parameter space more efficiently. This includes, e.g., global optimization schemes [HW99, Ham06, Ham07] or adaptive mesh generation in parameter space.

Acknowledgements

JA and MG are supported by the Deutsche Forschungsgemeinschaft through the Emmy Noether program under code GO1752/3-1. KH gratefully acknowledges financial support by the Fonds der Chemischen Industrie für den Hochschullehrernachwuchs.

References

[ASY97] Kathleen T. Alligood, Tim D. Sauer, and James A. Yorke. *Chaos : An Introduction to Dynamical Systems*. Springer, New York, 1997.

[BGS76] G. Benettin, L. Galgani, and J.M. Strelcyn. Kolmogorov entropy and numerical xperiments. *Phys. Rev. A*, 14:2338–2344, 1976.

[BKJH08] B. J. Bornstein, S. M. Keating, A. Jouraku, and M. Hucka. LibSBML: An API Library for SBML. *Bioinformatics*, 24(6):880–881, 2008.

[BW06] F. J. Bruggeman and H. V. Westerhoff. Approaches to biosimulation of cellular processes. *J Biol Phys*, 32(3-4):273–288, Jan 2006.

[DCS+09] Adel Dayarian, Madalena Chaves, Eduardo D Sontag, Anirvan M Sengupta, and Adam P Arkin. Shape, Size, and Robustness: Feasible Regions in the Parameter Space of Biochemical Networks. *PLoS Computational Biology*, 5(1):e1000256, Jan 2009.

[DM03] S.N. Dorogovtsev and J.F.F. Mendes. *Evolution of Networks*. Oxford, Oxford, 2003.

[Ham05] K. Hamacher. On Stochastic Global Optimization of one-dimensional functions. *Physica A*, 354:547–557, 2005.

[Ham06] K. Hamacher. Adaptation in stochastic tunneling global optimization of complex potential energy landscapes. *Europhys. Lett.*, 74(6):944–950, 2006.

[Ham07] K. Hamacher. Adaptive Extremal Optimization by Detrended Fluctuation Analysis. *J.Comp.Phys.*, 227(2):1500–1509, 2007.

[HB09] Simon L. Harding and Wolfgang Banzhaf. Distributed Genetic Programming on GPUs using CUDA. *Submitted to Genetic Programming and Evolvable Machines*, 2009.

[HFS+03] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[HSG+06] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jurgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[HW99] K. Hamacher and W. Wenzel. The Scaling Behaviour of Stochastic Minimization Algorithms in a Perfect Funnel Landscape. *Phys. Rev. E*, 59(1):938–941, 1999.

[IRN+04] Nobuyoshi Ishii, Martin Robert, Yoichi Nakayama, Akio Kanai, and Masaru Tomita. Toward large-scale modeling of the microbial cell for computer simulation. *J Biotechnol*, 113(1-3):281–94, Sep 2004.

[Iye09] Ravi Iyengar. Why We Need Quantitative Dynamic Models. *Sci. Signal.*, 2(64):eg3–, 2009.

[JK09] M Januszewski and M Kostur. Accelerating numerical solution of Stochastic Differential Equations with CUDA. *arxiv:0903.3852v1*, pages 1–14, Mar 2009.

[LCPG08] Hong Li, Yang Cao, Linda R Petzold, and Daniel T Gillespie. Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnol Prog*, 24(1):56–61, Jan 2008.

[LG09] Xuan Liu and David Gilbert. BioNessie - A Biochemical Networks Simulation Environment. 2009. Accepted for publication in the Journal of Cell Research.

[LNBB+06] Nicolas Le Novere, Benjamin Bornstein, Alexander Broicher, Melanie Courtot, Marco Donizelli, Harish Dharuri, Lu Li, Herbert Sauro, Maria Schilstra, Bruce Shapiro, Jacky L. Snoep, and Michael Hucka. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucl. Acids Res.*, 34(suppl_1):D689–691, 2006.

[LP09] H Li and L Petzold. Efficient Parallelization of Stochastic Simulation Algorithm for Chemically Reacting Systems on the Graphics Processing Unit. *International Journal of High Performance Computing Applications*, pages 1–27, Mar 2009.

[Lyn07] Michael Lynch. The evolution of genetic networks by non-adaptive processes. *Nat Rev Genet*, 8(10):803–813, Oct 2007.

[MFM+06] Rainer Machné, Andrew Finney, Stefan Müller, James Lu, Stefanie Widder, and Christoph Flamm. The SBML ODE Solver Library: a native API for symbolic and fast numerical analysis of reaction networks. *Bioinformatics*, 22(11):1406–1407, 2006.

[MSM04] Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingill. *Patterns for Parallel Programming*. Addison-Wesley Longman, 2004.

[NVI08] NVIDIA Corporation. *NVIDIA CUDA Programming Guide 2.1*, 2008.

[RED] At the time of writing, information about REDUCE was available from Anthony C. Hearn, RAND, Santa Monica, CA 90407-2138, USA. E-mail: reduce@reduce-algebra.com.

[Rös77] O.E. Rössler. Chaos in Abstract Kinetics: Two Prototypes. *Math. Biol.*, 39:275–289, 1977.

[vR06] Natal A. W van Riel. Dynamic modelling and analysis of biochemical networks: mechanism-based models and model-based experiments. *Briefings in Bioinformatics*, 7(4):364–374, Jan 2006.

[W.H95] W.H. Press et al. *Numerical Recipies in C*. Cambridge University Press, Cambridge, 1995.

[WSSV85] A. Wolf, J. B. Swift, H. Swinney, and J. A. Vastano. Determining Lyapunov exponents from a time series. *Physica*, 16D:285–317, 1985.