GPU-powered model analysis with PySB/cupSODA

Leonard A. Harris1,2,*, Marco S. Nobile3,4,*, James C. Pino2,5,*, Alexander L. R. Lubbock1,2, Daniela Besozzi3,4, Giancarlo Mauri3,4, Paolo Cazzaniga4,6,† and Carlos F. Lopez1,2,†

1Department of Cancer Biology, Vanderbilt University, Nashville, TN, USA
2Quantitative Systems Biology Center, Vanderbilt University, Nashville, TN, USA
3Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milan, Italy
4SYSBIO.IT Centre of Systems Biology, Milan, Italy
5Chemical and Physical Biology Graduate Program, Vanderbilt University, Nashville, TN, USA
6Department of Human and Social Sciences, University of Bergamo, Bergamo, Italy

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: Dr. Jonathan Wren

ABSTRACT

Summary: A major barrier to the practical utilization of large, complex models of biochemical systems is the lack of open-source computational tools to evaluate model behaviors over high-dimensional parameter spaces. This is due to the high computational expense of performing thousands to millions of model simulations required for statistical analysis. To address this need, we have implemented a user-friendly interface between cupSODA, a GPU-powered kinetic simulator, and PySB, a Python-based modeling and simulation framework. For three example models of varying size, we show that for large numbers of simulations PySB/cupSODA achieves order-of-magnitude speedups relative to a CPU-based ordinary differential equation integrator.

Availability and Implementation: The PySB/cupSODA interface has been integrated into the PySB modeling framework (version 1.4.0), which can be installed from the Python Package Index (PyPI) using a Python package manager such as pip. cupSODA source code and precompiled binaries (Linux, Mac OS/X, Windows) are available at github.com/aresio/cupSODA (requires an Nvidia GPU; developer.nvidia.com/cuda-gpus). Additional information about PySB is available at pysb.org.

Contact: c.lopez@vanderbilt.edu; paolo.cazzaniga@unibg.it

Supplementary information: Supplementary data are available at Bioinformatics online.

1 INTRODUCTION

Kinetic modeling of complex biochemical systems is central to the emerging field of systems biology (Kitano, 2002; Le Novère, 2015). Kinetic models require definition of numerous free parameters, usually obtained by calibration to experimental data, that specify initial species concentrations and kinetic rate constants. Once calibrated, a model should be analyzed for its sensitivity and predictive power over ranges of parameter values (Fisher and Henzinger, 2007). Both model calibration and analysis can require thousands to millions of model simulations for statistical convergence and significance (Gutenkunst et al., 2007; Eydgahi et al., 2013). In many cases, the computational expense of simulation at this scale makes detailed model analysis infeasible. Recently, efforts have been made to leverage the highly parallel structure of graphics processing units (GPUs) to accelerate scientific computations (Dematte and Prandi, 2010; Nobile et al., 2016). GPUs are well suited for applications in which the same arithmetic operations are applied to many independent data elements, e.g., solving independently parameterized systems of ordinary differential equations.
differential equations (ODEs). GPU-based kinetic simulators thus hold great promise for accelerating tasks such as model calibration and analysis, but are challenging for non-experts to use because they require specialized settings and inputs.

To address this problem, we have created a user-friendly interface between the GPU-based kinetic simulator cupSODA (Nobile et al., 2013, 2014) and PySB, a Python-based modeling and simulation platform (Lopez et al., 2013). cupSODA is built around the well-known adaptive stiff/non-stiff ODE integrator LSODA (Petzold, 1983). It is designed to perform thousands of parallel simulations, each independently parameterized, of mass-action kinetic models by leveraging the high-performance memories on the GPU, specifically the cached and non-mutable constant memory and the low-latency on-chip shared memory. PySB is a rule-based modeling (Chylek et al., 2014, 2015) platform for constructing and analyzing complex models of biochemical systems. Models can be constructed in native Python code or imported from various formats, including the Systems Biology Markup Language (SBML) (Hucka et al., 2003). PySB leverages powerful libraries within the Python ecosystem, such as NumPy, SymPy, and SciPy (Perez and Harris, 2011). To run simulations with PySB/cupSODA, one must construct multiple input files containing, e.g., the reaction stoichiometries and the initial species concentrations and rate parameter values for each specified simulation. Numerous simulator-specific parameters must also be defined, such as the number of CUDA “blocks” to use and the desired cupSODA memory configuration (see Supplementary Information). The number of simulations that cupSODA can run in parallel is limited by the number of CUDA “cores” on the GPU (usually a few thousand; see Supplementary Table S1), but the number of simulations that can be loaded onto the GPU at one time is usually many more than this, limited by the available memory (Nobile et al., 2013, 2014).

The PySB/cupSODA interface simplifies and streamlines the use of cupSODA via a CupSodaSimulator class, available within the PySB package. The class constructor accepts the following arguments:

- **model**: A PySB model object (required)
- **tspan**: A list of output time points (default: None)
- **initials**: A list or dictionary of initial species concentrations for each simulation (default: None)
- **param_values**: A list or dictionary of rate parameter values for each simulation (default: None)
- **verbose**: Verbose output (default: False)

The CupSodaSimulator constructor also recognizes numerous keyword arguments (kwargs), such as `n_blocks`, the number of CUDA blocks, and `memory_usage`, the desired memory configuration. Importantly, default values are defined for each kwarg, removing the need for user input. For example, if a user-defined value is not provided, the number of CUDA blocks is automatically calculated by querying the specifications of the GPU in use.

The CupSodaSimulator.run() method performs the simulations by constructing the cupSODA input files and invoking cupSODA as a subprocess (the method takes `tspan`, `initials`, and `param_values` as optional arguments). Additionally, the method reads into a three-dimensional array the results of the simulations (species time courses), which cupSODA outputs to (typically thousands of) separate text files. The user then has the ability to analyze and/or visualize the results using tools available within the Python ecosystem, e.g., plotting the time courses using the Matplotlib library (Perez et al., 2011). For convenience, a run_cupSODA wrapper function has also been implemented that combines invocations of the CupSodaSimulator constructor and run method into a single step. A workflow diagram and example Python script using the run_cupSODA function are provided in Supplementary Figs. S1 and S2, respectively.

### 2 FEATURES AND IMPLEMENTATION

cupSODA is designed to exploit the massive parallelism of the CUDA architecture (Nickolls et al., 2008). To run simulations with cupSODA, one must construct multiple input files containing, e.g., the reaction stoichiometries and the initial species concentrations and rate parameter values for each specified simulation. Numerous simulator-specific parameters must also be defined, such as the number of CUDA “blocks” to use and the desired cupSODA memory configuration (see Supplementary Information). The number of simulations that cupSODA can run in parallel is limited by the number of CUDA “cores” on the GPU (usually a few thousand; see Supplementary Table S1), but the number of simulations that can be loaded onto the GPU at one time is usually many more than this, limited by the available memory (Nobile et al., 2013, 2014).

The PySB/cupSODA interface simplifies and streamlines the use of cupSODA via a CupSodaSimulator class, available within the PySB package. The class constructor accepts the following arguments:

- **model**: A PySB model object (required)
- **tspan**: A list of output time points (default: None)
- **initials**: A list or dictionary of initial species concentrations for each simulation (default: None)
- **param_values**: A list or dictionary of rate parameter values for each simulation (default: None)
- **verbose**: Verbose output (default: False)

The CupSodaSimulator constructor also recognizes numerous keyword arguments (kwargs), such as `n_blocks`, the number of CUDA blocks, and `memory_usage`, the desired memory configuration. Importantly, default values are defined for each kwarg, removing the need for user input. For example, if a user-defined value is not provided, the number of CUDA blocks is automatically calculated by querying the specifications of the GPU in use.

The CupSodaSimulator.run() method performs the simulations by constructing the cupSODA input files and invoking cupSODA as a subprocess (the method takes `tspan`, `initials`, and `param_values` as optional arguments). Additionally, the method reads into a three-dimensional array the results of the simulations (species time courses), which cupSODA outputs to (typically thousands of) separate text files. The user then has the ability to analyze and/or visualize the results using tools available within the Python ecosystem, e.g., plotting the time courses using the Matplotlib library (Perez et al., 2011). For convenience, a run_cupSODA wrapper function has also been implemented that combines invocations of the CupSodaSimulator constructor and run method into a single step. A workflow diagram and example Python script using the run_cupSODA function are provided in Supplementary Figs. S1 and S2, respectively.

### 3 RESULTS

In Fig. 1A–C and Supplementary Fig. S3, we compare the run time efficiency of PySB/cupSODA to the CPU-bound ODE integrator LSODA, available in the Python package SciPy (Oliphant, 2007), for three example models listed in Table 1 (see Supplementary Information for descriptions). These include models of the eukaryotic cell cycle (Tyson, 1991), the Ras/cAMP/PAK signaling pathway in Saccharomyces cerevisiae (Besozzi et al., 2012), and extrinsically induced apoptosis in mammalian cells (EARM: extrinsic apoptosis reaction model) (Lopez et al., 2013). Run time comparisons show that in all cases SciPy/LSODA is faster for small numbers of simulations but PySB/cupSODA overtakes it for large numbers of simulations, achieving a maximum speedup of approximately one order of magnitude. Comparable speedups are achieved for other memory settings and GPUs (Supplementary Information and Supplementary Figs. S4 and S5).

For each model in Table 1, we also performed sensitivity analyses (Fig. 1D and Supplementary Figs. S6–S12) by quantifying changes in defined model outputs to variations (±20%) in initial protein concentrations around a set of reference values (Supplementary Tables S2–S4; see Supplementary Information for further details).
The ability to efficiently perform such analyses is critical since non-genetic variability within isogenic cell populations has been attributed to significant variations in protein concentrations across cells (Spencer et al., 2009). In Fig. 1D and Supplementary Fig. S11, we analyze the sensitivity in “time-to-death” in EARM (defined as the time at which Smac reaches 50% cleavage; see Supplementary Information and Supplementary Fig. S10) for a specific set of rate parameters. Our results show that time-to-death is sensitive to the initial levels of six of the 21 proteins considered. Of particular interest is the sensitivity to Bak. The same analysis performed for a different set of rate parameters (Supplementary Fig. S12) shows insensitivity to Bak but sensitivity to Bax. This indicates that the model harbors at least two alternative pathways to apoptosis induction. The analysis took ~11 min with PySB/cupSODA and ~35 min with SciPy/LSODA (for both parameter sets). Similar accelerations were seen for the cell cycle and Ras/cAMP/PKA models (Supplementary Information).

4 CONCLUSION
The PySB/cupSODA interface provides the modeling community with a high-performance GPU-based kinetic simulator, that can run thousands of parallel simulations on a common desktop workstation, within the easy-to-use framework of a full-fledged, open-source programming and analysis environment in Python. This will greatly accelerate and streamline the process of analyzing complex biochemical models for systems biology applications.

ACKNOWLEDGMENT
We thank Jeremy L. Muhlich for useful feedback during the implementation of this work.

Funding: This work was supported by the National Science Foundation [MCB-1411482]; the VICC Young Ambassador Award to C.F.L.; Vanderbilt Center for Quantitative Sciences and Kleberg Foundation Funds to C.F.L.; and the National Institutes of Health Vanderbilt Biomedical Informatics Training Program [NLM 5T15-LM007450-14 to L.A.H.]

REFERENCES