GPU accelerated biochemical network simulation

Yanxiang Zhou 1,*, Juliane Liepe 1, Xia Sheng 1,2, Michael P.H. Stumpf 1,2,3, †
and Chris Barnes 1,2, †

1Centre for Bioinformatics, Division of Molecular Biosciences, Imperial College London, London, UK, 2Institute of Mathematical Sciences, Imperial College London, London, UK and 3Centre for Integrative Systems Biology, Imperial College London, London, UK.

ABSTRACT

Motivation: Mathematical modelling is central to systems and synthetic biology. Using simulations to calculate statistics or to explore parameter space is a common means for analysing these models and can be computationally intensive. However, in many cases, the simulations are easily parallelizable. Graphics processing units (GPUs) are capable of efficiently running highly parallel programs and outperform CPUs in terms of raw computing power. Despite their computational advantages, their adoption by the systems biology community is relatively slow, since differences in hardware architecture between GPUs and CPUs complicate the porting of existing code.

Results: We present a Python package, cuda-sim, that provides highly parallelized algorithms for the repeated simulation of biochemical network models on NVIDIA CUDA GPUs. Algorithms are implemented for the three popular types of model formalisms: the LSODA algorithm for ODE integration, the Euler-Maruyama algorithm for SDE simulation and the Gillespie algorithm for MJP simulation. No knowledge of GPU computing is required from the user. Models can be specified in SBML format or provided as CUDA code. For running a large number of simulations in parallel, up to 360-fold decrease in simulation runtime is attained when compared to single CPU implementations.

Availability: http://cuda-sim.sourceforge.net/

Contact: christopher.barnes@imperial.ac.uk
m.stumpf@imperial.ac.uk

Supplementary information: Supplementary data are available at Bioinformatics online.

1 INTRODUCTION

Mathematical modelling is an integral part of systems and synthetic biology. Ordinary differential equations (ODEs) are the most commonly used methodology, but due to increasing appreciation of the importance of stochasticity in biological processes, stochastic differential equations (SDEs) and Markov jump processes (MJPs) are also applied. Since most models are non-linear, they generally cannot be solved analytically and therefore require numerical treatment. Furthermore, in order to understand behaviour across high dimensional parameter space or to perform simulation based inference (Toni et al., 2009; Liepe et al., 2010), a very large number of simulations is required, making analysis of even the simplest models extremely time consuming.

For computationally expensive calculations, graphics processing units (GPUs) can be used. GPUs are many core, multi-threaded chips that are capable of several hundred GFLOPS (Kirk and Hwu, 2010). In terms of raw computing power, a single GPU in a desktop PC is comparable to a CPU cluster, but is much cheaper. However, due to their single instruction multiple data (SIMD) architecture, only highly parallelized processes can be efficiently run on GPUs. Even though platforms for general purpose GPU computing like CUDA® (Compute Unified Device Architecture) from NVIDIA® which provides a CUDA API exist, it remains difficult and time consuming to port existing algorithms that were designed for execution on CPUs. There have been developments in porting biochemical network simulators to GPUs (reviewed in Dematte and Prandi, 2010) but there does not currently exist a general purpose simulation tool that integrates multiple algorithms within the same interface.

Here, we present a new Python package called cuda-sim which provides highly parallelized algorithms for large scale simulations of biochemical network models. It is compatible with all NVIDIA GPUs that support CUDA. Absolutely no knowledge of CUDA and GPU computing is needed for the user to access the simulation algorithms: (1) The integration of ODEs is carried out using a GPU implementation of LSODA (Hindmarsh, 1983), (2) SDE simulations are provided via the Euler-Maruyama algorithm (Kloeden and Platen, 1999) and (3) simulations from a MJP (or Master equation) are performed using the Gillespie algorithm (Gillespie, 1976). All functionality can be accessed via a Python interface that hides the implementation details.

2 IMPLEMENTATION

The cuda-sim package is implemented in Python using PyCUDA (Klöckner et al., 2009) to access the GPU. PyCUDA acts as a wrapper for the CUDA API and provides abstractions that facilitate programming. As an additional layer between code and hardware, PyCUDA can also accommodate future changes in GPU architecture and will help ensuring that cuda-sim remains available.
Two different pseudo random number generators (RNG) are used for the SDE and MJP simulations. In the MJP simulations, each thread carries out different numbers of simulation steps and therefore needs different numbers of random numbers that are provided by one Mersenne Twister RNG (Matsumoto and Nishimura, 1998) per thread. In the completely parallel SDE simulations, each thread requires the same number of random numbers that are local to a group of threads known as a warp (Thomas, 2010).

The user accesses the package by specifying SBML models that are parsed in cuda-sim using libSBML (Bornstein et al., 2008) and then converted to CUDA code modules. These code modules are automatically incorporated into the GPU kernels and run from within cuda-sim. Using a provided Python script, the user can specify model parameters and run the simulations with cuda-sim. Alternatively the algorithms can be called directly from Python, allowing incorporation into other software projects.

It is advisable to use dedicated general purpose GPUs like the Tesla C2050 that we used for our timing studies. GPUs that provide graphical output have a time limitation on the execution length of programs and therefore will not be compatible with larger simulations run with cuda-sim.

3 TIMING COMPARISONS

Using a model of p53-Mdm2 oscillations which contains 3 species, 6 reactions and 8 parameters (Geva-Zatorsky et al., 2006 and Supplementary Figure 1) we simulated different numbers of time series of length 100 hours and performed timing studies comparing the runtime on the GPU and the runtime on a single CPU (Figure 1). For the ODE integrations to yield different results for each thread, instead of using fixed parameters, we drew parameters from uniform random distributions in the interval between 0 and 2.

The CPU versions of the Gillespie and Euler-Maruyama algorithms were written in C++ and compiled with GCC using the -O3 optimisation flag. For the LSODA algorithm comparisons, the CPU implementation in the SciPy Python module was used. The testing was done on an Intel Core i7-975 Extreme Edition Processor 3.33 GHz machine with 12 GB of RAM and one Tesla C2050 GPU. All points are averages over three runs. Since the CPU runtimes scale linearly, the total CPU time for large numbers of simulations can extrapolated using a linear model.

For the LSODA, the Euler-Maruyama and the Gillespie algorithm, speed-ups of 47-fold, 367-fold and 12-fold are attained, respectively, for large numbers of simulations. Only for small numbers of simulations, are the CPU implementations of the three algorithms faster than the GPU versions (Figure 1 A–C). This is due to the fact that the initialisation on the GPU takes substantially longer than on the CPU. But since in most applications of these algorithms, either in order to explore the parameter space or to perform inference, at least thousands of simulations will be needed for which the GPU outperforms the CPU even for the rather simple p53-Mdm2 model.

We also compared the cuda-sim implementations of the LSODA and Gillespie algorithms with implementations in the Matlab package SBTOOLBOX2 (Schmidt and Jirstand, 2006) and our Euler-Maruyama implementation with the native sde function within Matlab. Since stochastic simulation in SBTOOLBOX2 supports only mass-action models, we used a model of enzyme kinetics (Supplementary Figure 2). We obtained similar timing to the p53-Mdm2 model when using our CPU implementations (Supplementary Figure 3) and speed-ups of between three and four orders of magnitude when compared to Matlab (Supplementary Figure 4).

4 CONCLUSIONS

GPUs offer a powerful and cost effective solution for parallel computing. cuda-sim provides a Python interface for biochemical network simulations using ODEs, SDEs and MJPs on NVIDIA CUDA GPUs and significantly reduces computation time. The package can be used as a standalone tool, or incorporated into other Python packages.

Funding: C.B. and M.P.H.S. acknowledge funding from the BBBSRC, J.L. acknowledges financial support from the Wellcome Trust, Y.Z. acknowledges financial support from the DAAD and Imperial College London, X.S. acknowledges funding from Kwoks’ Foundation of Sun Hung Kai Properties. M.P.H.S. is a Royal Society Wolfson Research Merit Award holder.
REFERENCES


