J Supercomput (2014) 69:17–24
DOI 10.1007/s11227-014-1208-8

GPU-accelerated simulations of mass-action kinetics models with cupSODA

Marco S. Nobile · Paolo Cazzaniga · Daniela Besozzi · Giancarlo Mauri

Published online: 23 May 2014
© Springer Science+Business Media New York 2014

Abstract In the last years, graphics processing units (GPUs) witnessed ever growing applications for a wide range of computational analyses in the field of life sciences. Despite its large potentiality, GPU computing risks remaining a niche for specialists, due to the programming and optimization skills it requires. In this work we present cupSODA, a simulator of biological systems that exploits the remarkable memory bandwidth and computational capability of GPUs. cupSODA allows to efficiently execute in parallel large numbers of simulations, which are usually required to investigate the emergent dynamics of a given biological system under different conditions. cupSODA works by automatically deriving the system of ordinary differential equations from a reaction-based mechanistic model, defined according to the mass-action kinetics, and then exploiting the numerical integration algorithm, LSODA. We

M. S. Nobile (✉) · G. Mauri
Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336, 20126 Milano, Italy
e-mail: nobile@disco.unimib.it

G. Mauri
e-mail: mauri@disco.unimib.it

P. Cazzaniga
Dipartimento di Scienze Umane e Sociali, Università degli Studi di Bergamo, Piazzale S. Agostino 2, 24129 Bergamo, Italy
e-mail: paolo.cazzaniga@unibg.it

D. Besozzi
Dipartimento di Informatica, Università degli Studi di Milano, Via Comelico 39, 20135 Milano, Italy
e-mail: besozzi@di.unimi.it

M. S. Nobile · P. Cazzaniga · D. Besozzi · G. Mauri
SYSBIO Centre for Systems Biology, Milano, Italy

show that cupSODA can achieve a 86× speedup on GPUs with respect to equivalent executions of LSODA on the CPU.

Keywords CUDA · Graphics processing unit · cupSODA · Biochemical simulation · Systems biology

1 Introduction

Since the introduction of general-purpose graphics processing units (GPUs), the adoption of graphics engines experienced a great boost in Bioinformatics, Systems Biology and Computational Biology [4,6], where central processing units (CPUs) traditionally represented the standard workhorses. The use of GPUs for the investigation of complex biological systems is motivated by the need for performing large numbers of computational analyses, which necessitate a computing power that usually overtakes the capability of standard desktop computers. Despite the considerable advantage in terms of computational speedup, scientific applications on GPUs risk remaining a niche for few specialists, since GPU-based programming substantially differs from CPU-based programming and usually requires the re-design of existing algorithms or the development of ad hoc procedures. Indeed, a challenge of GPU computing is that the direct porting of an application may be unfeasible, or may not fully exploit the computational power and massive parallelism of GPUs because of their innovative architecture and intrinsic limitations [5]. To overcome these problems, several software tools for the analysis of biological systems were recently released (see, e.g., [6,13,17]), so that also users with no knowledge of GPUs hardware and programming can easily access their computing power.

In this work we present cupSODA, a GPU-powered simulator for biological systems that allows to efficiently execute a large number of parallel deterministic simulations at a considerable reduced computational cost with respect to CPUs. In particular, cupSODA exploits the massive parallelism of CUDA architecture to execute different and independent simulations in each thread, therefore giving access to tera-scale computing on a common workstation and obtaining a low-cost and energy-wise alternative to the traditional high-performance computing infrastructures (i.e., cluster of machines).

Given a mathematical model of a biological system, simulation algorithms can be exploited to analyze the model and to make predictions on the way the system behaves in normal or perturbed conditions: the intensive exploration of high-dimensional parameter spaces allows to devise the different emergent behaviors that the system can present [1,3,15]. In the case of deterministic models, a system of ordinary differential equations (ODEs) is used to describe how the concentration of each chemical species varies in time; according to the law of mass action [9], these equations can be derived from a given mechanistic reaction-based model of the biological system under investigation. cupSODA is a simulator for models based on mass-action kinetics, whose peculiarity is that it automatically derives the system of ODEs—and perform their numerical integration using the LSODA algorithm [12]—simply starting from a set of biochemical reactions. This way, any user—having or not either GPU programming skills or mathematical modeling expertise—will be able to run parallel simulations

of mass-action kinetics systems at reduced costs. Indeed, when a large number of mutually independent simulations has to be performed, LSODA turns out to be very time consuming if the simulations are run in a sequential manner on the CPU. In what follows we show that our GPU implementation of LSODA turns out to be extremely advantageous.

2 Implementation of cupSODA

cupSODA relies on a C version of LSODA [12], ported and adapted to the CUDA architecture in order to be run on the GPU. cupSODA is designed to be a cross-platform implementation, so that it can be run on Microsoft Windows, Linux, and Apple OSX-based operating systems. Nevertheless, to exploit CUDA's massive parallelism for the execution of different and independent simulations in each thread, a Nvidia GPU is required; therefore, cupSODA itself requires a Nvidia video card to be executed.

In [12], LSODA was designed to solve differential systems in the canonical form, whereby the user is supposed to specify the system of ODEs, that must be implemented as a custom C function that is then passed to the algorithm. In order to speed up the computation when dealing with stiff systems, the Jacobian matrix must be implemented as a custom C function as well.

cupSODA, on the other hand, is designed to the purpose of being a *black-box* simulator, that can be easily used without any programming skills. In particular, cupSODA consists in a tool that automatically converts the mechanistic reaction-based model of a biological system—which can be indifferently given according to a deterministic or a stochastic formulation[1]—into the corresponding system of ODEs, in conformity with the mass-action kinetics [16], and then it encodes the ODEs and its corresponding Jacobian matrix as C arrays.

These two arrays are loaded into the GPU, parsed and implemented as custom functions. In order to encode each term of each equation into a linear data structure, without any loss of information, both arrays contain the following data: a signed multiplicative factor of the term; the index of the kinetic constant associated to the term; the number of the chemical species involved in the term and their corresponding indices (see an example in [10], Fig. 2). The terms of all ODEs are linked together in these arrays. To efficiently parse the arrays inside the GPU, we use two additional arrays storing the offsets of each equation (i.e., the indices of each term of each equation); the parsing algorithm consists in the function given in Fig. 1. A similar function is defined to parse the Jacobian matrix.

The cupSODA simulator was designed to speed up the time-consuming computational tasks typical of Systems Biology [1,3], which rely on the repetition of large numbers of simulations in perturbed conditions, generally realized by varying the initial concentrations of chemical species or the value of the kinetic constants. To fulfill this requirement, cupSODA is able to launch multiple threads which run inde-

[1] cupSODA automatically executes the conversion from the stochastic to the deterministic formulation of both reaction constants and initial molecular amounts, given that the volume of the modeled biological system is specified.

```
function decode_ODEs ( encoded_ODE, ODE_offset ):
  dX = [0, ..., 0]
  position = 0
  for i = 0 to ODE_offset.length do:
    ode_value = 0
    while( position<ODE_offset[i] ) do:
      term = encoded_ODE[position]
      term *= k[encoded_ODE[position+1]]
      species = encoded_ODE[position+2]
      for s = 0 to species do:
        term *= X[encoded_ODE[position+3+s]]
      end for
      position += (3+species)
      ode_value += term
    end while
    dX[i] = ode_value
  end for
  return dX
end function
```

Fig. 1 Pseudocode of the parsing algorithm, exploited device-side by cupSODA for the decoding of ODEs. A similar code is used for the decoding of the Jacobian matrix

pendent parallel simulations of the same model, with each thread exploiting its own parameterization and initial conditions. To this aim, parameters and initial conditions are contained in coalesced arrays, a strategy that allows a faster fetching of data from the *global memory*.

Besides the global memory (accessible from all threads), cupSODA also exploits the *shared memory* (accessible from threads belonging to the same block), the *local memory* (registers and arrays, accessible from owner thread), and the *constant memory* (cached and not modifiable). In particular, being the data transfer between host and device very time consuming, all temporary results are allocated on the memories of the GPU, and they are read back as soon as the simulations are over. To obtain a further reduction of the memory latencies, the current state and time of each simulation are stored into the shared memory, while all the constants values (e.g., number of reactions and chemical species in the model, length of ODEs and Jacobian arrays) and LSODA settings are stored into the constant memory.

Since the amount of shared memory is limited on each streaming multiprocessor—and poses a limitation on the blocks' size—cupSODA automatically calculates the number of threads per block and blocks per grid. We exploit double precision floating point values to allocate the states of the system and the current time, so that the consumption of shared memory of each thread is $M = 8 \times (N + 1)$ bytes of shared memory during the ODEs integration, where N is the number of chemical species in the system. It is possible to determine the threads-per-block value as $T_{\text{pb}} = \left\lfloor \frac{\text{SM}}{M} \right\rfloor$, where SM is the shared memory available on the GPU, so that the number of resulting blocks is $B = \left\lfloor \frac{T_{\text{tot}}}{T_{\text{pb}}} \right\rfloor$, where T_{tot} is the number of total threads requested by the user.

LSODA requires additional parameters for its functioning, the most relevant being the absolute and relative error tolerance values (denoted by AET and RET, respectively), and the maximum number of internal steps for each integration interval. AET and RET values can be either scalar values or specific vectors for each ODE: cupSODA accepts all combinations. Moreover, these values can be specified for each individual thread, allowing the user to simulate the same system with different tolerances and compare their outcomes.

cupSODA can be also easily employed to compare the outcome of simulations with any available experimental data (e.g., the concentration of some chemical species measured at sampling instants $t_0, \ldots, t_F$), for instance to validate the model under investigation or to perform a parameter estimation. To this aim, cupSODA invokes the LSODA kernel $F - 1$ times: each time the kernel is run over a (simulated) time interval of length $\Delta t = t_i - t_{i-1}$, $i = 1, \ldots, F$, and the concentration values of the output species are stored at the end of each Δt. Once the concentrations are stored, cupSODA provides a set of metrics (e.g., root mean square) that the user can exploit to evaluate the distance between each simulation outcome and the target dynamics.

3 Results

To show the effectiveness of cupSODA, we compare its performances against the COmplex PAthways SImulator (COPASI [7]), which we consider here as reference sequential simulator. To show the suitability of cupSODA, we performed different batches of independent deterministic simulations of three biological models characterized by an increasing complexity: the Michaelis–Menten (MM) enzymatic kinetics [9]; a simple model of gene expression in prokaryotic organisms (PGN) [14]; the signaling pathway Ras/cAMP/PKA in the yeast *S. cerevisiae* [2]. During each test we stored the dynamics of all chemical species consisting of 100 time instants—uniformly sampled over the whole simulation time—keeping track of the overall running time.

The GPU used for the tests is a Nvidia GeForce GTX 590, a video card with Fermi architecture equipped with 2×16 streaming multiprocessors for a total of 1,024 cores. The performances of the GPU were compared with a quad-core CPU Intel Core i7-2600 with a clock rate of 3.4 GHz. A direct comparison of GPUs and CPUs is a hard task, because of their architectural differences (e.g., the multiple cache levels of CPUs). Moreover, nominal peak capabilities of GPUs (2.48 TFlops in single precision, in the case of GTX 590) can be achieved only by implementing kernels completely adhering to the underlying SIMD architecture (e.g., by avoiding any conditional branch in the code). In addition, the occupancy of GPUs is affected by the usage of registers and shared memory, which are both limited resources on each streaming multiprocessor. cupSODA suffers from a high register pressure, which poses a limitation on the number of simultaneous blocks that can be executed by each multi-core processor: currently, our implementation is limited to 2 blocks on Fermi GPUs and 4 on Kepler GPUs. Hence, the theoretical computational power peak is hard to be reached; nonetheless, we compare here the performances of these two hardware devices since they are well representative of hardware components typically found in personal computers. All

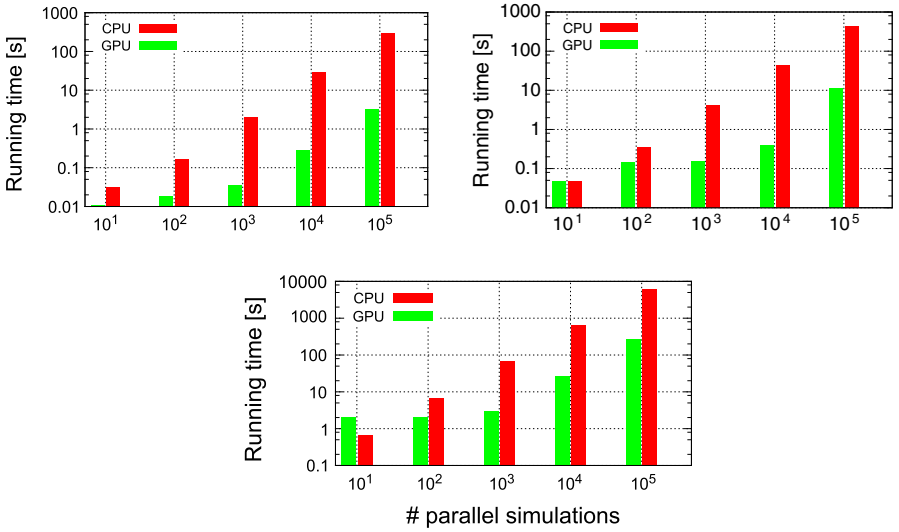

Fig. 2 Comparison between the computational time of cupSODA (*green histograms*) and COPASI (*red histograms*) for simulating the MM model (*top left*), the PGN model (*top right*), and the Ras/cAMP/PKA model (*bottom*). The y-axis are in logarithmic scale; RET= 10^{-10}, AET= 10^{-10} for MM and PGN models, RET= 10^{-10}, AET= 10^{-14} for the Ras/cAMP/PKA model. For all tests the maximum number of internal steps allowed during each call of LSODA was set to 10,000 (color figure online)

tests were executed on a system running the OS Microsoft Windows 7, CUDA version 5.0, COPASI 4.8 (build 35).

We show in Fig. 2 a comparison of the overall running times of COPASI and cupSODA, obtained by performing an increasing number of simulations with LSODA for the three test models. Our results show that cupSODA largely outperforms the LSODA algorithm implemented in COPASI and, in the case of 10^5 simulations (i.e., parallel threads) for the MM model (Fig. 2, top left), the computational cost on the GPU is nearly two orders of magnitude smaller than the CPU, namely, 3.358 vs. 289.335 s, which corresponds to a 86× speedup. In the case of the PGN model (Fig. 2, top right), the execution of 10^5 simulations takes 437 s on the CPU, while it takes just 6.5 s on the GPU, resulting in a 67×. Vice versa, a small number of simulations does not yield better performances, since the running time for 10 simulations of this model is similar on the two architectures: 0.047 (CPU) vs. 0.046 s (GPU). The execution of 10^5 simulations of the Ras/cAMP/PKA model (Fig. 2, bottom) takes 6133.3 s on the CPU and just 268.58 s on the GPU, resulting in a 23× speedup. This result clearly states that cupSODA is convenient when more than 10 simulations are required, since the computational cost for a small number of simulations is lower on the CPU.

Finally, we investigated the impact of the use of different memories on cupSODA, by executing 10^5 simulations of the Ras/cAMP/PKA model exploiting either the shared memory or the global memory to store the current state and time of the simulations. The running times were, respectively, 27.501 and 54.93 s: cupSODA is twice as fast as the naive porting of LSODA when the low latency memories are intensively exploited; in addition, it is worth noting that these running times are always lower than the CPU (651.741 s).

4 Conclusions

Computational methods for *in silico* analysis of biological systems has heightened the need for novel and efficient algorithms, to carry out fast simulations and to better explore the emergent behavior of these complex systems. GPUs are suitable for this kind of problems, providing cheap means to access tera-scale performances on common workstations. The bottleneck to the wide adoption of GPUs resides in the required programming skills for the development of GPU-based algorithms, and to handle specific features of GPU computing, such as the efficient usage of memory or the communication bandwidth between GPU and CPU. As a matter of fact, algorithms must be heavily restructured or purposely designed to fully exploit the underlying SIMD architecture and the memory hierarchy. Moreover, a direct porting from the CPU source-code to CUDA is usually unfeasible, because of the different architectures and the limited programming capabilities allowed by GPU kernels.

In this work we presented cupSODA, a GPU-powered simulator of biochemical system based on mass-action kinetics, designed to offer a black-box solution and capable of automatically translating the reaction-based model of a biological system into a system of ODEs. cupSODA relies on a numerical integrator for ODEs, called LSODA, which we implemented as a CUDA kernel in order to exploit the massive parallel capabilities of modern GPUs, thus achieving a relevant reduction of the computational time usually required to execute a huge number of independent simulations. The mutual independence of the simulations allows to fully exploit the underlying SIMD architecture; moreover, cupSODA benefits from an additional speedup, thanks to our choice of storing each system state into the low-latency shared memory.

A previous GPU implementation of LSODA algorithm was proposed in the cuda-sim library [17], a Python package providing GPU-accelerated biochemical simulations. The aim and design of cuda-sim are very different from cupSODA, as the latter allows to perform a massive number of simulations without writing any source code. cuda-sim also relies on a *just-in-time* technique, whereby the code for LSODA that will be executed on the GPU is automatically created and compiled at run-time: this is indeed a flexible and elegant solution, but adds a relatively long compilation time and requires the availability of a CUDA compiler and the CUDA toolkit on the running machine. During the development of cupSODA we opted for the encoding of ODEs and the Jacobian matrix into linear arrays which are processed device-side, without the need for any intermediate recourse to the CUDA driver API or any metaprogramming techniques. Hence, with cupSODA it is possible to immediately simulate any biological system modeled according the mass-action kinetics, a characteristics that is particularly appealing when the model itself is repeatedly modified, for instance when it undergoes a reverse engineering process [3,8,11]: in such a case, cupSODA only needs to update the GPU representation of the ODEs and the Jacobian matrix, in order to be ready to run a massive number of simulations of the new model.

In this work, we tested the performances of cupSODA on three biological models of increasing complexity and we showed that cupSODA becomes more convenient than the CPU counterpart when a consistent number of parallel simulations has to be run, with a break-even that depends on the complexity of the system under investigation. Indeed, when performing demanding computational analysis such as, e.g., parameter

sweep, parameter estimation or sensitivity analysis, the outstanding advantage of novel softwares as cupSODA clearly comes to light.

The cupSODA software is available from the authors upon request.

References

1. Aldridge B, Burke J, Lauffenburger D et al (2006) Physicochemical modelling of cell signalling pathways. Nat Cell Biol 8:1195–1203
2. Besozzi D, Cazzaniga P, Pescini D 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 Bioinf Syst Biol 2012:10
3. Chou I, Voit E (2009) Recent developments in parameter estimation and structure identification of biochemical and genomic systems. Math Biosci 219(2):57–83
4. Dematté L, Prandi D (2010) GPU computing for systems biology. Brief Bioinform 11(3):323–333
5. Farber R (2011) Topical perspective on massive threading and parallelism. J Mol Graphics Modell 30:82–89
6. Harvey MJ, De Fabritiis G (2012) A survey of computational molecular science using graphics processing units. WIREs Comput Mol Sci 2(5):734–742
7. Hoops S, Sahle S, Gauges R et al (2006) COPASI: a COmplex PAthway SImulator. Bioinformatics 22(24):3067–3074
8. Koza J, Mydlowec W, Lanza G et al (2007) Automatic computational discovery of chemical reaction networks using genetic programming. In: Džeroski S, Todorovski L (eds) Computational discovery of scientific knowledge, LNCS, vol 4660, pp 205–227
9. Nelson D, Cox M (2004) Lehninger principles of biochemistry. W. H. Freeman Co, New York
10. Nobile MS, Besozzi D, Cazzaniga P et al (2013) cupSODA: a CUDA-powered simulator of mass-action kinetics. In: Malyshkin V (ed) Proceedings of 12th international conference on parallel computing technologies (PaCT 2013), vol LNCS 7979, pp 344–357
11. Nobile MS, Cazzaniga P, Besozzi D et al (2013) Reverse engineering of kinetic reaction networks by means of Cartesian genetic programming and particle swarm optimization. In: IEEE congress evolutionary computation (CEC 2013), pp 1594–1601
12. Petzold L (1983) Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. SIAM J Sci Stat Comp 4(1):136–148
13. Vigelius M, Lane A, Meyer B (2011) Accelerating reaction-diffusion simulations with general-purpose graphics processing units. Bioinformatics 27(2):288–290
14. Wang Y, Christley S, Mjolsness E et al (2010) Parameter inference for discretely observed stochastic kinetic models using stochastic gradient descent. BMC Syst Biol 4:99
15. Wilkinson D (2009) Stochastic modelling for quantitative description of heterogeneous biological systems. Nat Rev Genet 10:122–133
16. Wolkenhauer O, Ullah M, Kolch W et al (2004) Modeling and simulation of intracellular dynamics: choosing an appropriate framework. IEEE Trans Nanobiosci 3(3):200–207
17. Zhou Y, Liepe J, Sheng X et al (2011) GPU accelerated biochemical network simulation. Bioinformatics 27(6):874–876