Vol. 27 no. 2 2011, pages 288–290
doi:10.1093/bioinformatics/btq622

Systems biology

Advance Access publication November 8, 2010

Accelerating reaction–diffusion simulations with general-purpose graphics processing units

Matthias Vigelius*, Aidan Lane and Bernd Meyer
FIT Centre for Research in Intelligent Systems, Monash University, Clayton Victoria 3800, Australia

Associate Editor: Jonathan Wren

ABSTRACT

Summary We present a massively parallel stochastic simulation algorithm (SSA) for reaction-diffusion systems implemented on Graphics Processing Units (GPUs). These are designated chips optimized to process a high number of floating point operations in parallel, rendering them well-suited for a range of scientific high-performance computations. Newer GPU generations provide a high-level programming interface which turns them into General-Purpose Graphics Processing Units (*GP*GPUs). Our SSA exploits GPGPU architecture to achieve a performance gain of two orders of magnitude over the fastest existing implementations on conventional hardware.

Availability: The software is freely available at http://www.csse.monash.edu.au/~berndm/inchman/.

Contact: matthias.vigelius@monash.edu

Supplementary Information: Supplementary data are available at *Bioinformatics* online.

Received on July 21, 2010; revised on October 21, 2010; accepted on November 2, 2010

1 INTRODUCTION

Treating chemical reaction systems as spatially homogeneous is insufficient for many applications in a biological context. Often, spatial distributions play an important role in the dynamic evolution of biomolecular systems. The analysis of such systems requires accurate yet highly performant simulation algorithms that can handle spatially inhomogeneous reaction–diffusion (Broderick and Rubin, 2006). Unfortunately, stochastic simulation methods for this problem are computationally extremely expensive and it thus becomes necessary to build parallel versions of existing algorithms (Ballarini *et al.*, 2009). GPGPUs can potentially provide high-performance computing resources to a broad audience and are consequently becoming increasingly popular for scientific computing. In the present article, we present a data-parallel GPGPU implementation of an SSA, which achieves significant performance gains over the fastest conventional implementations. To the best of our knowledge, this is the first time that a data-parallel GPGPU implementation of a quantitative SSA for spatially heterogeneous reaction-diffusion networks is reported.

Several approaches to compute the stochastic time evolution of reaction–diffusion networks can be found in the literature, such as agent-based models (Barrett *et al.*, 2008), first-passage kinetic Monte Carlo algorithms (Donev *et al.*, 2010; Oppelstrup *et al.*, 2009) or, on a mesoscopic level, compartment-based models, such as the Next-Subvolume Method (NSM) (Elf *et al.*, 2003).

Not all these methods lend themselves equally well to a data-parallel implementation on GPGPUs. The standard algorithms, based on Gillespie's next reaction method (Gillespie, 1976), perform an event-based simulation in which global communication is required to compute the next event time as well as to determine the corresponding reaction. Due to the high cost of global synchronization and inter-node communication, attempts to implement the Gillespie SSA directly on GPGPUs could only yield moderate performance gains (Dittamo and Cangelosi, 2009). Petzold and Li (2009) pursue a different approach by running many instances of the same model in parallel on a GPGPU. This technique allows immediate parallelization of sequential algorithms but cannot speed up individual runs and can thus only exploit the full hardware potential if a large number of simulations are required.

For a full parallelization, methods that treat diffusion seperately from reactions appear to be more promising. Such methods are termed hybrid. One can distinguish between deterministic–stochastic algorithms, where diffusion is handled in a deterministic manner (Rossinelli *et al.*, 2008), and stochastic–stochastic methods, which are preferable for cases where the diffusive species is not necessarily present in high densities. Two prominent examples of the latter type of hybrid algorithms are the Gillespie Multiparticle Method (GMP), first presented by Rodríguez *et al.* (2006), and the Multinomial Simulation Algorithm (MSA) (Lampoudi *et al.*, 2009). In this article, we report a GPGPU implementation of GMP.

Hybrid stochastic reaction–diffusion algorithms are an active and relatively recent field of research and no clear champion has emerged yet. Cellular automata methods are widely used to simulate reaction–diffusion systems [see Takahashi *et al.* (2005) for a comprehensive review]. In particular, the multiparticle lattice gas algorithm underlying GMP (Chopard *et al.*, 1994) has been successfully applied, for example, to problems in electrochemistry (Li *et al.*, 2009). Its applicability to biochemical pathways has been shown in a number of studies, e.g. for the phosphoenolpyruvate-dependent phosphotransferase (PTS) pathway in *Escherichia coli* (Rodríguez *et al.*, 2006). We believe that MSA should in principle be just as well suited for a data-parallel implementation. However, the free availability of the source code made GMP our first choice. A detailed comparison of computational methods for reaction–diffusion networks is given by Dobrzyski *et al.* (2007).

For completeness, we point out that the deterministic treatment of reaction–diffusion equations with GPGPUs has a long history in the context of computer graphics. The driving motivation behind

*To whom correspondence should be addressed.

these efforts is to realistically reproduce naturally occuring textures, for example of animals[1] or skin desease effects.[2] Likewise, agent-based reaction–diffusion simulations on GPUs have been performed before (Barrett *et al.*, 2008), but here the focus is, in contrast to our work, generally not on exact quantitative studies.

2 IMPLEMENTATION AND PERFORMANCE

A detailed description of the algorithm and our parallelization approach can be found in the Supplementary Material. Here we can only give a brief overview. The computational domain of dimensionality d is divided into equally spaced, cubical subvolumes with sidelength λ. Each subvolume contains a number of particles of each species, which are assumed to be distributed homogeneously within the subvolume. In each time step, the diffusion part is treated separately from the reaction algorithm. This approach is termed operator-splitting (Rodríguez *et al.*, 2006). During each diffusion time step Δt, every subvolume independently handles all reactions employing the standard direct method (Gillespie, 1977) until the simulation time for that particular subvolume exceeds Δt. One diffusion step is then simulated by redistributing particles among neighbouring subvolumes according to a probabilistic evolution rule which Chopard *et al.* (1994) proved to asymptotically reproduce the macroscopic diffusion equation. Rodríguez *et al.* (2006) demonstrated that this method correctly reproduces the literature results for a generic gene expression model and a phosphoenolpyruvate:glucose phosphotransferase system.

We parallelize this algorithm by assigning one thread to each subvolume of the computational domain. Each GPU thread executes the same C function, termed a kernel, at the same time. Threads are arranged into independent blocks, which cannot be synchronized globally. Blocks of threads implicitly divide the computational domain into rectangular spatial areas. Threads corresponding to subvolumes at the boundaries of such areas must communicate across blocks to exchange particles in a diffusion event. We need to return from the kernel to the host function to synchronize globally across block boundaries.

The main loop is thus executed on the CPU once per diffusion time step Δt. It calls three different kernels on the GPU. After choosing the species to be diffused next, we first perform the reactions for each subvolume independently in the GILLESPIE kernel before diffusing the corresponding species in the DIFFUSION kernel. Finally, we need to update the block boundaries in the UPDATE kernel. Implementation details of the kernels can be found in the Supplementary Material.

We evaluate the performance gain of our implementation by comparing it to MesoRD (Hattne *et al.*, 2005), the stochastic simulation compiler (SSC) (Lis *et al.*, 2009), which is a very efficient implementation of an SSA, and a serial implementation of GMP (Rodríguez *et al.*, 2006). We use two different setups to perform the scaling tests: a diffusion problem without reactions and an $A+B$ annihilation problem. We initialize both on a quadratic domain and vary the number of subvolumes per dimension, while keeping the subvolume size constant (for a detailed explanation of the problems and parameters see the Supplementary Material). The serial code (MesoRD and GMP) is compiled with GCC 4.4.3 and run on an

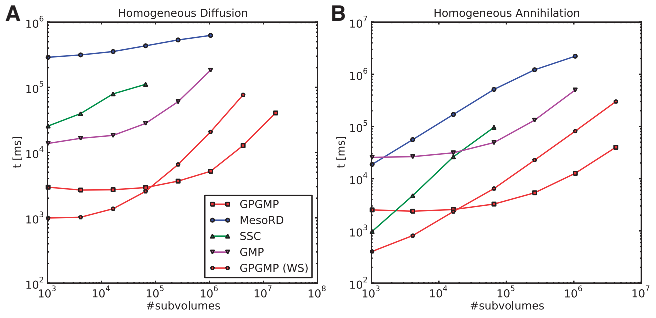

Fig. 1. Runtime (in ms) versus number of subvolumes for the homogeneous diffusion problem (**A**) and the homogeneous $A+B$ annihilation problem (**B**). The setup is detailed in Sections 2 and 3.4 of the Supplementary Material. Results are shown for MesoRD (blue circles), SSC (green, upward triangles), serial GMP (magenta, downward triangles), GPGMP run on a workstation (red pentagons) and GPGMP run on the GPU cluster (red squares). GPGMP outperforms all other implementations by two orders of magnitude.

INTEL E6550 dual core CPU at 2.33 GHz using the Linux operating system UBUNTU (Version 10.04 LTS). We run GPGMP on our GPU cluster (equipped with NVIDIA Tesla S1070 GPUs) and an ordinary workstation (an Apple MacBookPro with an Intel Core2 Duo CPU running at 2.5 GHz and a GeForce 8600M GT graphics card).

Figure 1 shows the run time (in ms) for SSC (green, upward triangles), MesoRD (blue circles), serial GMP (magenta, downward triangles), GPGMP run on our GPU cluster (red squares), and GPGMP run on a workstation (red pentagons). GPGMP scales very well with the number of subvolumes. The Tesla T10 processor consists of 240 cores but, due to the thread-scheduling algorithm, no significant speed loss occurs until the total number of subvolumes exceeds $\sim 10^6$. For the diffusion problem (left panel), with an integration domain consisting of 1024×1024 subvolumes, GPGMP outperforms the serial GMP implementation by a factor of 35. A test run with 4096×4096 subvolumes only takes about eight times longer than the same setup with 1024×1024 subvolumes. No comparison data can be given for this case, because the other implementations are unable to deal with such a large system size. We report similar speed gains for the homogeneous $A+B$ annihilation problem (Fig. 1B). GPGMP outperforms serial GMP by a factor of $\sim$39 and MesoRD by $\sim$175. However, these speed gains can be lower if the problem is highly inhomogeneous and the workload is unevenly shared between all GPU threads. On a workstation, GPGMP performs slightly better than on the GPU cluster for low number of subvolumes ($N \lesssim 10^5$) but saturates earlier. For high $N \gtrsim 10^5$, the Tesla T10 processor on the cluster outperforms the workstation graphics card. The initial performance advantage of the standard workstation appears to be due to the fact that its CPU is superior to the cluster node CPU and that the proportion of the workload performed by the CPU becomes negligible only for large subvolume numbers.

GPGMP scales with the number of subvolumes N as expected. For low $N \lesssim 10^4$, the runtime decreases with increasing N since the total workload (combined reaction and diffusion events) is shared among more threads. For $10^4 \lesssim N \lesssim 10^6$, the total runtime plateaus until the number of concurrently running threads saturates and the runtime scales linearly with N. Note again that the saturation point is higher than the number of GPU cores (240 for a Tesla T10 GPU)

[1]http://www.sci.utah.edu/~allen/reaction-diffusion.html
[2]http://developer.download.nvidia.com/SDK/9.5/Samples/samples.html

since the the thread scheduler is very effective in hiding memory-access latency. After saturation, GPGMP scales linearly with N. On the other hand, MesoRD behaves as $\mathcal{O}(\log N)$ and the runtime benefit of GPGMP decreases with increasing N. The saturation point can of course be shifted to higher N by using hardware with a larger number of cores, for example the next-generation NVIDIA Fermi architecture.

3 DISCUSSION

We have described an implementation of the Gillespie Multiparticle Method (GMP) on GPGPUs. We report performance gains of two orders of magnitude compared with standard implementations of the (exact) inhomogeneous stochastic simulation algorithm and the (hybrid) serial implementation of GMP.

Like any other hybrid method, GMP sacrifices some numerical accuracy for performance gains. This trade-off can in principle be arbitrarily adjusted through the choice of the diffusion time step. For a more detailed discussion, we refer the reader to Section 2.2.7 of the Supplementary Material.

We provide a full simulation system (Inchman) that allows the user to run their models without any coding (on the Monash Sun Grid GPU cluster). Access to this system is via an easy-to-use web interface[3] that understands systems biology markup language (SBML) specifications, the lingua franca of systems biology. In addition, we provide a full implementation of the algorithm on our website. Researchers may use the C++ interface to construct their own reaction-diffusion model from scratch. The application programming interface (API) is designed to mimic the structure of SBML models, allowing the user to easily convert their models into GPGMP without having to deal with the internal details of the simulation algorithm. A variety of test problems that can be used as templates are part of the package. The full source code is included so the user can easily add the relevant GPU implementation into their own projects.

Most scientific applications require a reasonable sample size to extract statistic information from the simulations. It is therefore necessary to perform multiple runs of the same problem, possibly with varying input parameters. We pursue a 2-fold approach to tackle this requirement. First, the standard implementation of GPGMP distributes the total number of runs over all available GPGPU cards. This works best if the host machine provides a one-to-one ratio of CPU cores to GPGPU cards. Second, we are integrating Inchman with Nimrod,[4] a toolkit to allow users to run parameter sweeps and parameter optimization and distribute runs over GPGPU clusters. This will become an integral part of the next release of Inchman. Our implementation is currently limited to a constant diffusivity and vanishing drift field. We are working on extending the algorithm to incorporate spatially inhomogeneous diffusivity and drift.

Funding: Australian Research Council (ARC DP0879239).

Conflict of Interest: none declared.

REFERENCES

Ballarini,P. *et al*. (2009) Taming the complexity of biological pathways through parallel computing. *Brief Bioinform.*, **10**, 278–288.

Barrett,C.L. *et al*. (2008) Episimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, IEEE Press, Piscataway, NJ, USA, pp. 1–12.

Broderick,G. and Rubin,E. (2006) The realistic modeling of biological systems: a workshop synopsis. *Complexus*, **3**, 217–230.

Chopard,B. *et al*. (1994) Multiparticle lattice gas automata for reaction diffusion systems. *Int. J. Mod. Phy. C*, **5**, 47–63.

Dittamo,C. and Cangelosi,D. (2009) Optimized parallel implementation of gillespie's first reaction method on graphics processing units. *Int. Conf. Comput. Model. Simul.*, 156–161.

Dobrzyski,M. *et al*. (2007) Computational methods for diffusion-influenced biochemical reactions. *Bioinformatics*, **23**, 1969–1977.

Donev,A. *et al*. (2010) A first-passage kinetic monte carlo algorithm for complex diffusion-reaction systems. *J. Comput. Phys.*, **229**, 3214–3236.

Elf,J. *et al*. (2003) *Mesoscopic Reaction-Diffusion in Intracellular Signaling*. Vol. 5110, SPIE, Bellingham, Washington, USA, pp. 114–124.

Gillespie,D.T. (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Computat. Phys.*, **22**, 403–434.

Gillespie,D.T. (1977) Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, **81**, 2340–2361.

Hattne,J. *et al*. (2005) Stochastic reaction-diffusion simulation with MesoRD. *Bioinformatics*, **21**, 2923–2924.

Lampoudi,S. *et al*. (2009) The multinomial simulation algorithm for discrete stochastic simulation of reaction-diffusion systems. *J. Chem. Phys.*, **130**, 094104.

Li,L. *et al*. (2009) Computational simulation of metastable pitting of stainless steel. *Electrochimica Acta*, **54**, 6389–6395.

Lis,M. *et al*. (2009) Efficient stochastic simulation of reaction-diffusion processes via direct compilation. *Bioinformatics*, **25**, 2289–2291.

Oppelstrup,T. *et al*. (2009) First-passage kinetic monte carlo method. *Phys. Rev. E*, **80**, 066701.

Petzold,L. and Li,H. (2009) Efficient parallelization of stochastic simulation algorithm for chemically reacting systems on the graphics processing unit. In *International Journal of High Performance Computing Applications*, pp. 107–116.

Rodríguez,J.V. *et al*. (2006) Spatial stochastic modelling of the phosphoenolpyruvate-dependent phosphotransferase (pts) pathway in escherichia coli. *Bioinformatics*, **22**, 1895–1901.

Rossinelli,D. *et al*. (2008) Accelerated stochastic and hybrid methods for spatial simulations of reaction-diffusion systems. *Chem. Phys. Lett.*, **451**, 136–140.

Takahashi,K. *et al*. (2005) Space in systems biology of signaling pathways – towards intracellular molecular crowding in silico. *FEBS Lett.*, **579**, 1783–1788.

[3]http://www.csse.monash.edu.au/~berndm/inchman/

[4]https://messagelab.monash.edu.au/Nimrod