UC Santa Barbara

UC Santa Barbara Previously Published Works

Title

EFFICIENT PARALLELIZATION OF STOCHASTIC SIMULATION ALGORITHM FOR CHEMICALLY REACTING SYSTEMS ON THE GRAPHICS PROCESSING UNIT

Permalink

<https://escholarship.org/uc/item/85q0201g>

Journal

International Journal of High Performance Computing, 1

Authors

Li, Hong
Petzold, Linda

Publication Date

2009

DOI

10.1177/1094342009106066

Peer reviewed

**EFFICIENT PARALLELIZATION OF
STOCHASTIC SIMULATION ALGORITHM
FOR CHEMICALLY REACTING SYSTEMS ON
THE GRAPHICS PROCESSING UNIT**

H. Li * L. Petzold *

** Department of Computer Science, University of
California, Santa Barbara, CA 93106, U.S.A.*

Proposed running head: Parallelization of SSA on GPU
 Author 1: Hong Li Address: Department of Computer
 Science, University of California, Santa Barbara, CA
 93106, U.S.A. Phone number: 805-893-5728 Fax Number:
 805-893-5435 Email address: hongli@cs.ucsb.edu Author 2:
 Linda Petzold (Corresponding author) Address:
 Department of Computer Science, University of California,
 Santa Barbara, CA 93106, U.S.A. Phone number:
 805-893-5362 Fax number: 805-893-5435 Email address:
 petzold@engr.ucsb.edu

The special symbols:

```
\newcommand{\vctr}[1]{\mbox{\boldmath $\#1$}}
\newcommand{\timevec}[2]{\vctr{\#1}_{\#2}}
\newtheorem{example}{\scshape Example}[section]
```

Abstract: In biological systems formed by living cells, the small populations of some reactant species can result in inherent randomness which cannot be captured by traditional deterministic approaches. In that case, a more accurate simulation can be obtained by using the Stochastic Simulation Algorithm (SSA). Many stochastic realizations are required to capture accurate statistical information of the solution. This carries a very high computational cost. The current generation of graphics processing units (GPU) is well-suited to this task. We describe our implementation, and present some computational experiments illustrating the power of this technology for this important and challenging class of problems.

Keywords: Stochastic simulation; Chemically reacting systems; System biology; Parallel; GPU.

* This work was supported in part by the U.S. Department of Energy under DOE award No. DE-FG02-04ER25621, by the National Science Foundation under NSF awards CCF-0428912, CTS-0205584, CCF-326576, and by the Institute for Collaborative Biotechnologies through grant DAAD19-03-D004 from the U.S. Army Research Office.

1. INTRODUCTION

Chemically reacting systems have traditionally been simulated by solving a set of coupled ordinary differential equations (ODEs). Although the traditional deterministic approaches are sufficient for most systems, they fail to capture the natural stochasticity in some biochemical systems formed by living cells (Gillespie, 1976; Gillespie, 1977; McAdams and Arkin, 1997; A. Arkin, J. Ross and H.H. McAdams, Aug 1998), in which the small population of a few critical reactant species can cause the behavior of the system to be discrete and stochastic. The dynamics of those systems can be simulated accurately using the machinery of Markov process theory, specifically the stochastic simulation algorithm (SSA) of Gillespie (Gillespie, 1976; Gillespie, 1977). For many realistic biochemical systems the computational cost of simulation by the SSA can be very high. The original form of the SSA is called the Direct Method (DM). Much recent work has focused on speeding up the SSA by reformulating the algorithm (Gibson and Bruck, 2000; Y. Cao, H. Li and L. Petzold, 2004; J. M. McColluma, G. D. Peterson, C. D. Cox, M. L. Simpson and N. F. Samatova, Feb. 2005; J. Blue and Sullivan, 1995; Schulze, 2002; Li and Petzold, 2006).

Often, the SSA is used to generate large (typically ten thousand to a million) ensembles of stochastic realizations to approximate probability density functions of species populations or other output variables. In this case, even the most efficient implementation of the SSA will be very time consuming. Parallel computation on clusters has been used to speed up the simulation of such ensembles (H. Li, Y. Cao, L. Petzold and D. Gillespie, 2007). In (M. Yoshimi, Y. Osana, Y. Iwaoka, A. Funahashi, N-Hiroi, Y. Shibata, N. Iwanaga, H. Kitano and H. Amano, 2005), the use of Field Programmable Gate Arrays (FPGAs) is investigated to speed up the SSA simulation. However, clusters are still relatively expensive to buy and maintain, and specialized devices such as FPGAs are difficult to program. In this paper we introduce an efficient parallelization of ensembles of SSA simulations for chemically reacting systems on the low cost graphics processing unit (GPU) NVIDIA GeForce 8800GTX ¹.

This paper is organized as follows. In Section 2 we briefly review the Stochastic Simulation Algorithm and some basics of parallel computation with the graphics processing unit. In section 3 we introduce the efficient parallelization of SSA on the GPU. Simulation results are presented in Section 4, and in Section 5 we draw some conclusions.

2. BACKGROUND

2.1 Stochastic Simulation Algorithm

The Stochastic Simulation Algorithm applies to a spatially homogeneous chemically reacting system within a fixed volume at a constant temperature. The system involves N molecular species $\{S_1, \dots, S_N\}$ represented by the dynamical state vector $X(t) = (X_1(t), \dots, X_N(t))$ where $X_i(t)$ is the population of species S_i in the system at time t , and M chemical reaction channels $\{R_1, \dots, R_M\}$. Each reaction channel R_j is characterized by a propensity function a_j and state change vector $\nu_j = \{\nu_{1j}, \dots, \nu_{Nj}\}$, where $a_j(x)dt$ is the probability, given $X(t) = x$, that one R_j reaction will occur in the next infinitesimal time interval $[t, t + dt)$, and ν_{ij} is the change in the number of species S_i due to one R_j reaction.

The *Next Reaction Density Function* (Gillespie, 2001), which is the basis of SSA, gives the joint probability that reaction R_j will be the next reaction and will occur in the infinitesimal time interval $[t, t + dt)$, given $X(t) = x$. By applying the laws of probability, the joint density function is formulated as follows:

$$P(\tau, j | \mathbf{x}_t, t) = a_j(\mathbf{x}_t)e^{-a_0(\mathbf{x}_t)\tau}, \quad (1)$$

where $a_0(\mathbf{x}_t) = \sum_{j=1}^M a_j(\mathbf{x}_t)$.

Starting from (1), the time τ , given $X(t) = x$, that the next reaction will fire at $t + \tau$, is the exponentially distributed random variable with mean $\frac{1}{a_0(x)}$

$$P(\tau | x, t) = a_0(\mathbf{x}_t)e^{-a_0(\mathbf{x}_t)\tau} \quad (\tau \geq 0), \quad (2)$$

The index j of that firing reaction is the integer random variable with probability

$$P(j | \tau, x, t) = \frac{a_j(\mathbf{x}_t)}{a_0(\mathbf{x}_t)} \quad (j = 1, \dots, M). \quad (3)$$

¹ At the time of this writing, the NVIDIA GeForce 8800GTX costs about \$400.

Thus on each step of the simulation, the random pairs (τ, j) are obtained based on the standard Monte Carlo inversion generating rules: first we produce two uniform random numbers r_1 and r_2 from $U(0, 1)$, the uniform distribution on $[0, 1]$. Then τ is given by

$$\tau = \frac{1}{a_0(\mathbf{x}_t)} \ln \left(\frac{1}{r_1} \right). \quad (4)$$

The index j of the selected reaction is the smallest integer in $[1, M]$ such that

$$\sum_{j'=1}^j a_{j'}(\mathbf{x}_t) > r_2 a_0(\mathbf{x}_t). \quad (5)$$

Finally, the population vector X is updated by the state change vector ν , and the simulation is advanced to the next reacting time.

Because SSA must simulate every reaction event, simulation with SSA can be quite computationally demanding. When an ensemble (ten thousand to a million realizations or more) must be generated, the computation can become intractable. A number of different formulations of SSA have been proposed, in an effort to speed up the simulation (Gibson and Bruck, 2000; Y. Cao, H. Li and L. Petzold, 2004; J. M. McColluma, G. D. Peterson, C. D. Cox, M. L. Simpson and N. F. Samatova, Feb. 2005; J. Blue and Sullivan, 1995; Schulze, 2002; Li and Petzold, 2006). The most time-consuming step of the SSA is the selection of the next reaction to fire. The complexity of this step for the Direct Method is $O(M)$, where M is the number of reactions. To the best of our knowledge, the fastest known SSA formulation is something we call the Logarithmic Direct Method (LDM) because its complexity for the critical step is $O(\log M)$. The LDM algorithm comes from the literature on Kinetic Monte Carlo (KMC) algorithms. The SSA is a type of KMC algorithm that is applied to chemical kinetics. Because of the special structure of the chemical kinetics problems, it has been possible to put SSA on a solid theoretical foundation. Further efficiency of the LDM can be achieved by using sparse matrix techniques in the system state update stage (Li and Petzold, 2006). It is this method (LDM with sparse matrix update) that we will use in our timing comparisons. The algorithm is summarized as follows:

- (1) *Initialization*: Initialize the system.
- (2) *Propensity calculation*: Calculate the propensity functions a_i ($i = 1, \dots, M$), and save the intermediate data as an ordered sequence of the propensities subtotaled from 1 to M , while summing all the propensity functions to obtain a_0 .
- (3) *Reaction time generation*: Generate the firing time of the next reaction.
- (4) *Reaction selection*: Select the reaction to fire next with binary search on the ordered subtotal sequence.
- (5) *System state update*: Update the state vector x by ν_j with sparse matrix techniques, where j is the index of the current firing reaction. Update the simulation time.
- (6) *Termination*: Go back to stage 2 if the simulation has not reached the desired final time.

2.2 Using the Graphics Processor Unit as a Data Parallel Computing Device

2.2.1. Modern Graphics Processor Unit The Graphics Processing Unit (GPU) is a dedicated graphics card for personal computers, workstations or video game consoles. Recently, GPUs with parallel programming capacities have become available. The GPU has a highly parallel structure with high memory bandwidth and more transistors devoted to data processing than to data caching and flow control (compared with a CPU architecture), as shown in Figure 1 (Corporation, n.d.a). The GPU architecture is most effective for problems that can be implemented with stream processing and using limited memory. Single Instruction Multiple Data (SIMD), which involves a large number of totally independent records being processed by the same sequence of operations simultaneously, is an ideal general purpose graphics processing unit (GPGPU) application.

2.2.2. NVIDIA 8 Series GeForce-based GPU Architecture NVIDIA corporation claims its graphics processing unit (GPU) as a “second processor in personal computers” (Corporation, n.d.b), which means that the data parallel computation intensive part of applications can be off-loaded to the GPU (Corporation, n.d.a). The Compute Unified Device Architecture (CUDA) Software Development Kit

Fig 1.

(SDK), supported by the NVIDIA GeForce 8 Series, supplies general purpose functionality for non-graphics applications to use the processors inside the GPU.

We chose the NVIDIA 8800 GTX chip, which was released at the end of 2006 with 768MB RAM and 681 million transistors on a $480mm^2$ surface area. There are 128 stream processors on a GeForce 8800 GTX chip, divided into 16 clusters of multiprocessors as shown in Figure 2 (Corporation, n.d.a). Each multiprocessor has 16 KB shared memory which brings data closer to the ALU. The processors are clocked at 1.35 GHz with dual processing of scalar operations supported, thus the peak computation rate accessible from the CUDA is $(16 \text{ multiprocessors} * 8 \text{ processors / multiprocessor}) * (2 \text{ flops / MAD})^2 * (1 \text{ MAD / processor-cycle}) * 1.35 \text{ GHz} = 345.6 \text{ GFLOP/s}$. The maximum observed bandwidth between system and device memory is about 2GB/second. To put this in perspective, the GPU chip provides up to 197 times the performance of an Intel Core 2 Duo E6700 2.67GHz dual-core Conroe processor (people, n.d.).

Fig 2.

Unfortunately, current GPU chips support only 32-bit technology while today's CPUs support 64-bit technology. Thus, only single-precision floating point is supported. Additionally, each multiprocessor has only 16K shared memory available. This restricts the range of applications that can make use of this architecture.

2.2.3. CUDA: A GPU Software Development Environment The CUDA provides an essential high-level development environment with standard C language, resulting in a minimal learning curve for beginners to access the low-level hardware. The CUDA provides both scatter and gather memory operations for

² A MAD is a multiply-add.

development flexibility. It also supports a shared memory with fast read and write ability to reduce the dependence of application performance on the DRAM bandwidth (Corporation, n.d.a).

The structure of CUDA computation broadly follows the data-parallel model: each of the processors executes the same sequence of instructions on different sets of the data in parallel. The data can be broken into a 1D or 2D grid of blocks, and each block can be 1D, 2D or 3D and can allow up to 512 threads which can collaborate through shared memory. Threads within a block can collaborate via the shared memory.

Unfortunately, currently the CPU and GPU cannot run in parallel. Also it is not possible to download data and run a kernel in parallel, or to execute multiple kernels at the same time through the CUDA. Users can use a single kernel to perform multiple tasks by branching in the kernel based on the thread id, but the branches will slow down the simulation.

3. IMPLEMENTATION DETAILS

3.1 *Parallelism across the simulations*

Our focus is on computation of ensembles of SSA realizations. Ensembles of SSA runs for chemically reacting systems are very well-suited for implementation on the GPU through the CUDA. The simulation code can be put into a single kernel running in parallel on a large set of system stat vectors $X(t)$. The large set of final stat vectors $X(t_{final})$ will contain the desired results.

The initial conditions $X(0)$ and the stoichiometric matrix ν originally will be in the host memory. We must copy them to the device memory by CUDAMemcpy in the driver running on the the CPU. We minimize the transfer between the host and device by using an intermediate data structure on the device and batch a few small transfers into a big transfer to reduce the overhead for each transfer. Next, we need to consider the relatively large global memory vs. the limited-size shared memory. The global memory adjacent to the GPU chip has higher latency and lower bandwidth than the on-chip shared memory. It takes about 400-600 clock cycle latency to access the global memory vs. 4 clock cycles to read or write the shared memory. To effectively use the GPU, our simulation makes as much use of on-chip shared memory as possible. We load $X(0)$ and the stoichiometric matrix ν from the device memory to the shared memory at the very beginning of the kernel, process the data (propensity calculation, state vector update, etc.) in shared memory, and write the result back to the device memory at the end. Because the same instruction sequence is executed for each data set, there is a low requirement for flow control. This matches the GPU’s architecture. The instruction sequence is performed on a large number of data sets which do not need to swap out, hence the memory access latency is negligible compared with the arithmetic calculation.

The CUDA allows each block to contain at most 512 threads, but blocks with the same dimension and size that run the same kernel can be put into a grid of blocks. Thus the total number of threads for one kernel can be very large. Given the total number of realizations of SSA to be simulated, the number of threads per block and the number of blocks must be carefully balanced to maximize the utilization of computation resources. For stochastic simulation, we can’t use too many threads per block since there is only a limited shared memory and all system state vectors and propensities have been put in shared memory for efficient frequent access. Thus the number P of threads per block should satisfy $(N + M) * 4 * P + \alpha < 16K$, where N is the number of chemical species, M is the number of reactions, 4 is the size (in bytes) of an integer/float variable, 16K is the maximum shared memory we can use within one block, and α is the shared memory used by the random number generator (this is relatively small).

3.1.1. Random Number Generation Statistical results can only be relied on if the independence of the random number samples can be guaranteed. Thus generating independent sequence of random numbers is one of the important issues of implementing simulation for ensembles of stochastic simulation algorithms in parallel.

Originally we considered pre-generating a large number of random numbers by the CPU. Since the CPU and GPU can’t run in parallel, we can pre-generate a huge number of random numbers and store them in the shared memory and swap back to the CPU to generate more when they are used up. Alternatively, we could pre-generate a huge number of random numbers and put them in the global memory. Both methods will waste too much time for data access. Furthermore, the Scalable Parallel Random Number Generators Library (SPRNG) (Mascagni, 1999; Mascagni and Srinivasan., 2000), which we use in our

StochKit (H. Li, Y. Cao, L. Petzold and D. Gillespie, 2007) package for discrete stochastic simulation because of its excellent statistical properties, cannot be implemented on the GPU due to its complicated data structure. The only solution appears to implement a simple random number generator on the GPU. Experts suggest using a mature random number generator instead of inventing a new one, since it requires great care and extensive testing to evaluate a random number generator (Richard P. Brent, 1992). Thus we chose the Mersenne Twister from the literature in our application (members, n.d.).

The Mersenne Twister (MT) was developed by Makoto Matsumoto and Takuji Nishimura (Makoto Matsumoto and Takuji Nishimura, 1998) in 1997. This method has passed many statistical randomness tests including the stringent Diehard tests (members, n.d.). The fully tested MT random number generator can efficiently generate high quality, long period random sequences with high order of dimensional equidistribution. Another good property of the MT is its efficient use of memory. Thus it is very suitable for our application. The original single threaded C version was developed by Takuji Nishimura and Makoto Matsumoto, with initialization improved in 2002 (Podlozhnyuk, n.d.). In our implementation we modified Eric Mills's multithreaded C implementation (members, n.d.). Since our application requires a huge number of random numbers even for one realization of a simple model, we use the shared memory for random number generation to minimize the data launching and accessing time. This random number generation implementation allows 227 threads for each block, which is more than the number of threads needed by the stochastic simulation.

4. PARALLEL SIMULATION PERFORMANCE

The performance of the parallel simulation is limited by the number of processors available for the computation, the workload of the available processors, and the communication and synchronization costs. It is important to note that more processors does not necessarily mean better performance. Our simulations were run on the NVIDIA GeForce 8800GTX installed on a personal computer with Intel Pentium 3.00Ghz CPU and 3.50GB of RAM with physical Address Extension, and Windows XP Professional Version 2002 with Service Pack 2.

EXAMPLE 4.1. Decay Dimerization Model

The decay dimerization model (Gillespie, 2001) involves three reacting species S_1, S_2, S_3 and four reaction channels R_1, R_2, R_3, R_4

We used the reaction rate constants from (Gillespie, 2001),

$$c_1 = 1, \quad c_2 = 0.002, \quad c_3 = 0.5, \quad c_4 = 0.04, \tag{7}$$

and the initial conditions

$$X_1 = 10^4, \quad X_2 = X_3 = 0. \tag{8}$$

The simulation performance has been extraordinary, as shown in Figure 3. For 30000 realizations, the parallel (GPU) simulation is almost 215 times faster than the sequential simulation on the host computer.

In general, as the size of the system increases, the speed-up of the GPU decreases because of the limited shared memory. However, most biochemical systems are loosely coupled. Very large biochemical systems arise when the model takes into account spatial inhomogeneity. The SSA is based on the assumption of a spatially homogenous system. However, by discretizing the space into cells and introducing variables associated to the population of the species in each cell, the SSA can also be applied to spatially inhomogeneous systems. Here we construct a simple example to illustrate the power of the GPU for this type of problem.

EXAMPLE 4.2. Spatially Inhomogeneous Model

This model was introduced by Shnerb et al. (N. Shnerb, Y. Louzoun, E. Bettelheim, S. Solomon, 2000) to illustrate the difference between the continuous deterministic approach and the discrete stochastic approach. For convenience, we simplified the model slightly by fixing the position of one species. The model is defined on a 2-dimensional grid. Species A is initially located at a single grid point and moves

Fig 3.

randomly with a given diffusion coefficient. Species B is initially located in a randomly-chosen area of adjacent grid points, away from the border regions. The model is simulated over a fixed time period, to find the spatial distribution of A . Two types of reactions are involved. Species A decays with a constant rate μ , and divides with rate λ when it meets the catalyst B . We simulated the model with $n = 8, 10, 16, 20$. To each grid cell (labeled (i, j)), we assign variables $A_{i,j}$ for species A and $B_{i,j}$ for species B . The reactions are listed as follows

The diffusion rate for A is $\mu = 0.5$. The decay rate for A is 0.1. The division rate (when A reaches the region occupied by B) is $\lambda = 0.0025$. The initial states are set so that one cell contains a population of 100 of species A , 4 cells contain a population of 100 of species B , and the remainder of the cells contain no A or B respectively.

$$\begin{aligned}
 A_{i,j} &= \begin{cases} 100, & i = 10, j = 10, \\ 0, & \text{else.} \end{cases} \\
 B_{i,j} &= \begin{cases} 100, & \text{selected } i, j, \\ 0, & \text{else.} \end{cases}
 \end{aligned} \tag{10}$$

The simulation was carried out from $t_0 = 0$ to $t_{final} = 0.5$. To use the shared memory efficiently, we group the M reactions according to the type of the reaction. We first determine which group will fire next, and then determine at which grid point that type of reaction will fire. By doing this, we can avoid saving the propensities at each grid point, which is what is normally done in SSA (Y. Cao, H. Li and L. Petzold, 2004; Li and Petzold, 2006). Instead, we keep track only of the number of species A at each grid point. By doing this, we can dramatically reduce the number of operations consumed in the calculation of the propensities, as well as the use of the shared memory and global memory. The disadvantage is that we must update the propensities for each group very frequently. We measured the CPU time for 40000 realizations. The timing results for different grids are shown in Table 5. The parallel (GPU) simulation is about 200 times faster than the sequential simulation on the host computer.

The corresponding CPU time vs. GPU time for the 16 by 16 grid is shown in Figure 4.2.

For these computations, we have been able to store all frequently-used reaction rates in shared memory. Because the shared memory is limited, it is not possible to store all of the data for a large grid like 100×100 in shared memory. Additionally, we can't run too many realizations for a large grid at the same time, because the device memory of the GeForce 8800 GTX is also limited (768M). For 5000 realizations of the 100×100 grid, the parallel simulation is about 50 times faster than the sequential one.

5. CONCLUSIONS

The SSA is the workhorse algorithm for discrete stochastic simulation in systems biology. Even the most efficient implementations of the SSA can be very time-consuming. Often the SSA is used to generate ensembles (typically ten thousand to a million) of stochastic simulations. The current generation of GPUs appears to be very well-suited for this purpose. On the two model problems we tested, we observed speedups about 200 times for the GPU, over the time to compute on the host workstation. With this

Fig 4.

impressive performance improvement, in one day we can generate data which would require more than six months of computation with the sequential code.

This technology is not quite ready for the novice user. Programs must be written to be memory efficient, with the GPU architecture in mind. The computation is limited to single precision, the API does not yet have all the features to take full advantage of the architecture, transferring data to the GPU is time consuming, the CPU and GPU cannot compute simultaneously, and there are a number of problems and limitations that NVIDIA is still working on.

List of Captions:

Figure 1: CPU vs. GPU architecture.

Figure 2: Hardware Model.

Figure 3: Simulation time for dimer decay model on GPU vs. host computer.

Figure 4: Simulation time for Spatially Inhomogeneous model on GPU vs. host computer, 16 by 16 grid.

REFERENCES

- Makoto Matsumoto and Takuji Nishimura (1998). Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* **8**, 3–30.
- A. Arkin, J. Ross and H.H. McAdams (Aug 1998). Stochastic kinetic analysis of developmental pathway bifurcation in phage λ -infected E. Coli cells. *Genetics* **149**, 1633–1648.
- Corporation, NVIDIA (n.d.a). NVIDIA CUDA Compute Unified Device Architecture Programming Guide. <http://developer.download.nvidia.com>.
- Corporation, NVIDIA (n.d.b). NVIDIA homepage. <http://www.nvidia.com>.
- Gibson, M. and J. Bruck (2000). Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem.* **105**, 1876–1889.
- Gillespie, D.T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.* **22**, 403–434.
- Gillespie, D.T. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* **81**, 2340–2361.
- Gillespie, D.T. (2001). Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.* **115**(4), 1716–1733.
- H. Li, Y. Cao, L. Petzold and D. Gillespie (2007). Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnology Progress*. Submitted.
- J. Blue, I. Beichl and F. Sullivan (1995). Faster Monte Carlo simulations. *Physical Rev. E* **51**, 867–868.
- J. M. McColluma, G. D. Peterson, C. D. Cox, M. L. Simpson and N. F. Samatova (Feb. 2005). The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *J. Comput. Biol. Chem.* **30**, 39–49.
- Li, H. and L. Petzold (2006). Logarithmic Direct Method for discrete stochastic simulation of chemically reacting systems. Technical report. Department of Computer Science, University of California, Santa Barbara. <http://www.engr.ucsb.edu/~cse>.
- M. Yoshimi, Y. Osana, Y. Iwaoka, A. Funahashi, N-Hiroi, Y. Shibata, N. Iwanaga, H. Kitano and H. Amano (2005). The design of scalable stochastic biochemical simulator on FPGA. *Proc. of I. C. on Field Programmable Technologies (FPT2005)* pp. 139–140.
- Mascagni, M. (1999). SPRNG: A scalable library for pseudorandom number generation. In: *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*. San Antonio, Texas.
- Mascagni, M. and A. Srinivasan. (2000). SPRNG: A scalable library for pseudorandom number generation. In: *ACM Transactions on Mathematical Software*. Vol. 26. pp. 436–461.
- McAdams, H.H. and A. Arkin (1997). Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci. USA* **94**, 814–819.
- members, NVIDIA Forums (n.d.). NVIDIA forums. <http://forums.nvidia.com>.
- N. Shnerb, Y. Louzoun, E. Bettelheim, S. Solomon (2000). The Importance of Being Discrete - Life Always Wins on the Surface Proc. *Proc. Natl. Acad. Sci. USA* **97**, 10332.
- people, PCperspective (n.d.). PCperspective. <http://www.pcper.com>.
- Podlozhnyuk, Victor (n.d.). Mersenne Twister. <http://developer.download.nvidia.com>.
- Richard P. Brent (1992). Uniform Random Number Generators for Vector and Parallel Computers. *Report TR-CS-92-02*.
- Schulze, T. P. (2002). Kinetic Monte Carlo simulations with minimal searching. *Physical Review E* **65**(3), 036704.
- Y. Cao, H. Li and L. Petzold (2004). Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Phys. Chem.* **121**(9), 4059–4067.

List of Tables:

S	N	M	ST	PT	SU
8*8	68	388	127 296.8	506.8	251.2
10*10	104	604	204 006.2	856.3	238.2
16*16	260	1 536	511 498.2	2338.6	218.7
20*20	404	2 404	785 646.3	3815.3	205.9

Table 1. small computation time for the Spatially Inhomogeneous model, where S is the system size, N is the number of species, M is the number of reactions, ST is the sequential simulation time, PT is the parallel simulation time, SU is speed up.