Parallel stochastic simulation using graphics processing units for the Systems Biology Toolbox for MATLAB

Supplemental material

Guido Klingbeil, Radek Erban, Mike Giles, and Philip K. Maini

This document briefly introduces parallel computations on graphics processing units (GPUs) and the implemented exact stochastic simulation algorithms (i) the stochastic simulation algorithm (SSA) by Gillespie [5], (ii) the logarithmic direct method (LDM) by Li and Petzold [10] and (iii) the next reaction method of Gibson and Bruck [3]. The LDM and NRM are more efficient reformulations of the SSA.
Section 1 briefly introduces the application of GPUs to general purpose computation. Section 2 motivates the stochastic simulation of well-mixed chemical reaction systems and introduces the implemented simulation algorithms. Section 3 provides details on the threading approach taken and section 4 on the sampling regimes used. Section 5 lists the features and limitations of our stochastic simulation MATLAB plugin. Section 6 lists the hardware and software used.

1 Parallel computing on a GPU

Stochastic simulations of chemical reaction systems are *per se* not well parallelisable since the current state of the reaction system depends on the chain of previous reaction events. However NVIDIA's compute unified device architecture (CUDA) enables the efficient computation of ensembles of stochastic simulations in parallel using GPUs. While GPUs emerged as dedicated graphics boards to accelerate 3D graphics, these devices have recently been used for general purpose parallel computations. NVIDIA introduced CUDA in November 2006 in its GeForce 8800 GPU [11, 14]. GPUs are especially well suited for problems where the same set of instructions can be applied to several data sets simultaneously. This is called single instruction multiple data (SIMD). Even if the problem itself can not be well parallelised, several instances can be executed in parallel.

GPUs evolved over the last decade from special purpose devices supporting the central processing unit (CPU) by rendering 3D-graphics efficiently to multi-core data parallel general purpose processing units with hundreds of compute cores [1].
CPUs evolved towards very long pipelines to achieve instruction level parallelism increasing instruction throughput and multi-level caches to hide memory access latency. Pipelined processors are organised internally into stages which can work on separate jobs semi-independently. Fetching and decoding an instruction begins before the prior instruction finishes executing. While pipelining increases throughput, it comes at the cost of requiring branch prediction for optimal performance. Compared to GPUs, on a CPU considerably less space is allocated to the arithmetic logic units (ALUs) and more to caches and control logic like branch prediction [7]. Another major difference between CPUs and GPUs is the available memory bandwidth. The design of a generic dual-core CPU as well as a GPU is shown in Figure 1. While the CPU has a high bandwidth between its processing cores using the level 2 (L2) cache, the available bandwidth to main memory is limited. The GPU has a high memory bandwidth between the SMs and graphics card memory. CUDA enables one to apply GPUs to parallel general purpose computations. A NVIDIA CUDA enabled GPU consists of a set of streaming multiprocessors (SMs). Each SM currently aggregates 8 single precision and one double precision floating point processing cores called stream processors (SPs) accompanied by two special function units (SFUs). The SM is the smallest independent unit of scheduling with its own instruction cache, thread select logic, 64 kB of register file, 16 kB of shared memory, and 8 kB of constant cache [2]. Each SM is a threaded single-issue processor with single instruction multiple data (SIMD). Each SM can execute up to eight thread blocks with a total of 1024 threads, which is 128 threads per SP, concurrently. Each SM has a 64 kB register file comprising 16 k 32-bit wide entries. So each of the SM's eight SPs has a 2 k entry register file. The register's usage is assigned compile time. The graphics card's memory is called global memory. It is accessible by all threads of all SMs but it has a high access latency of 400 to 600 clock cycles [15].
Each SP can perform two floating point operations per cycle by its multiply-add unit. Each SFU can perform 4 instructions per cycle. This gives a total per SM of 16 operations per cycle for the eight SPs and eight for the two SFUs [12].
CUDA uses a variation of SIMD, as used in vector computers, called single instruction multiple thread (SIMT). The SIMT architecture applies one instruction to multiple independent threads in parallel achieving data level parallelism. In contrast to vector computers using SIMD, where a single instruction is applied to all data lanes, the threads are scheduled in groups, called a "warp", of 32 threads. This allows different warps to take different execution paths without performance penalty.
Within a warp, threads of a warp either execute the same instruction or stay idle. This allows threads of the warp to branch and take other execution paths. The execution of threads taking different branches is serialised decreasing the overall performance. An example is shown in Figure 2. Four threads are executing the same code fragment in parallel, but taking different branches of the if-statement and get serialised. The advantage of SIMT over SIMD is the

[1] The NVIDIA Tesla architecture has up to 240 cores, the Fermi architecture up to 512 cores.
[2] Based on NVIDIA Tesla architecture [15].

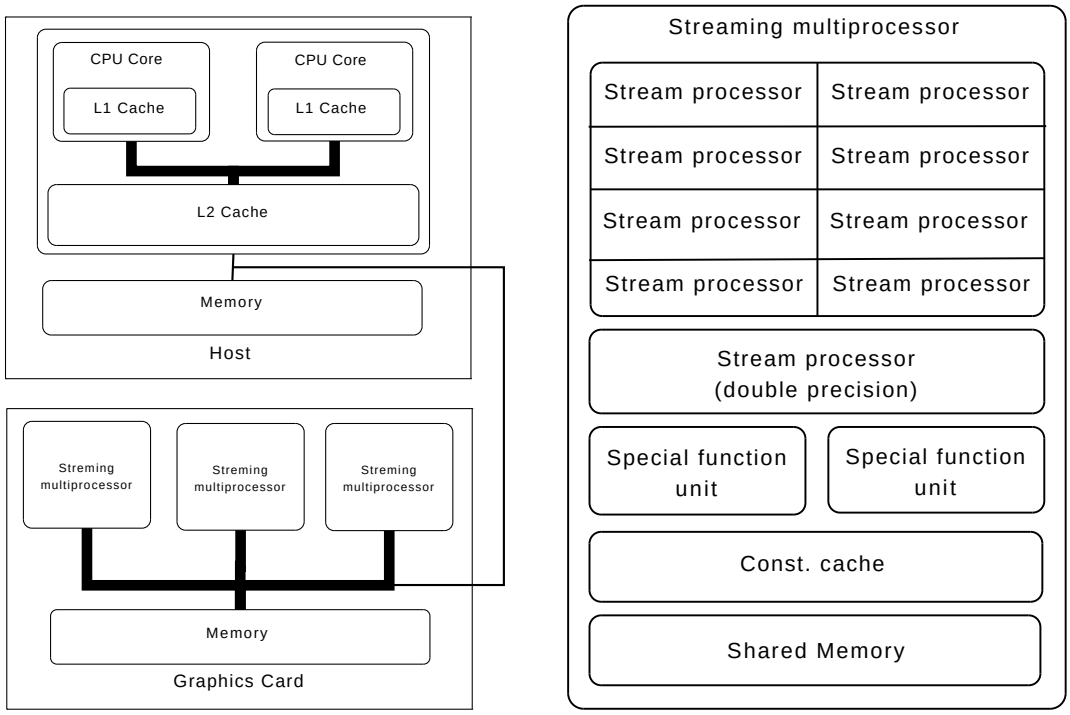

Figure 1 – *On the left the relative memory bandwidth between the processing cores of a multi-core CPU with its caches and the GPU. The line thickness of the links between the different components indicates the relative bandwidth available. In a multi-core CPU, the bandwidth between the cores and the shared 2-nd level cache is high, while the bandwidth to main memory is substantially lower. On the graphics board, the available bandwidth between the multiprocessors and the on-board memory is high compared to the link to the host main memory. On the right the organisation of a streaming multiprocessor is given. It aggregates eight single precision floating point stream processors and one double precision. The two special function units provide single precision functions such as trigonometric or logarithm.*

```
1  float a = unifrnd();
2  int b = 0;
3
4  if(a <= 0.5) {
5      // if-branch
6      b = b + 1;
7  } else {
8      // else-branch
9      b = b - 1;
10 }
```

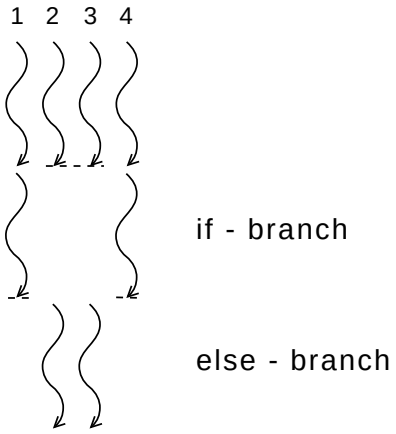

Figure 2 – *Thread execution of a branch. The code of the example is given on the left. Based on the uniform random number a, b is either incremented or decremented with a probability of 0.5. On the right the thread execution of a block of 4 threads executing the example is shown.*

independent scheduling of thread warps. This gives a higher flexibility and efficiency when branches occur, since only diverging threads within a warp need to be serialised. Obviously, full efficiency is gained when all threads of a warp take the same execution path [11].
Multiple warps of threads can be grouped into blocks which again form a computational grid. Blocks are assigned to the SMs and the threads of a block can cooperate and share cached memory areas and shared memory. Each thread is able to access its own set of local variables in local memory. Variables in local memory are stored in registers (fast access but read after write penalty of 2 cycles). If register space is exceeded, local variables spill into global memory.

2 Stochastic simulation of chemical reaction systems

The time evolution of a chemical reaction system is neither continuous since reactions are atomic, nor deterministic since a reaction may, but does not have to, occur whenever the appropriate molecules collide. Nevertheless, assuming a sufficiently large molecular population, a chemical reaction system can be treated as a deterministic continuous process which can be modelled using ordinary differential equations (ODEs). However in many biological systems small molecular abundances make deterministic ODE based simulations inaccurate [1]. As an example, the average copy numbers of some mRNA molecules relevant to the budding yeast cell cycle per cell may be less than one. Proteins, the actual work horses of the cell, are translated from mRNA. In this scenario all protein of a certain kind may be translated from a single mRNA molecule [8]. This means that depending on a single event occurring, the transcription of a mRNA molecule, the system may take different paths. The behaviour of the reaction system in a single cell, like switching between two states, may depend on stochastic fluctuations [6, 17].

2.1 Stochastic simulation algorithm

The Gillespie stochastic simulation (SSA) of chemical reaction systems addresses two questions [4, 5]. First, when does the next reaction event occur? Second, of which kind is this event? The SSA is an exact algorithm to simulate the time evolution of realisations consistent with the chemical master equation of a well mixed chemical reaction system. One time step of the Gillespie SSA is given in Listing 1, below. The SSA considers a spatially homogeneous (well mixed) molecular system of M reactions $R_i,\ i = 1, \ldots, M$, in thermal equilibrium of the N molecular species $S_j,\ j = 1, \ldots, N$. The current state vector is defined as

$$\mathbf{X(t)} = (x_1(t), \ldots, x_N(t))$$

where $x_i(t)$ is the number of molecules of chemical species $S_i(t), i = 1, 2, \ldots, N$.
The M reactions $R_i,\ i = 1, \ldots, M$, are accompanied by the vector of propensity functions

$$\mathbf{a(X(t))} = (a_i(\mathbf{X}(t)), \ldots, a_M(\mathbf{X}(t)))$$

and the state change vectors $\boldsymbol{\nu}_i$. The $\boldsymbol{\nu}_i$ compose the $N \times M$ stoichiometric reaction matrix ν. The N components

$$\nu_{ij},\ j = 1, \ldots, N$$

of the change vector $\boldsymbol{\nu}_i$ describe the change in the number of molecules of species S_i due to a R_j reaction event. The term $a_j(\mathbf{X}(t))dt$ is the probability, given the current state $\mathbf{X}(t)$, that a R_i reaction event will occur in the next infinitesimal time interval $[t, t + dt)$. The total propensity function $a_0(\mathbf{X}(t))$ is the sum of all propensity functions [4, 5]

$$a_0(\mathbf{X}(t)) = \sum_{i=1}^{M} a_i(\mathbf{X}(t)) \ .$$

The core of the SSA is to use random numbers to choose state transitions. The time τ at any given state of the system $\mathbf{X}(t)$ until the next reaction fires at $t + \tau$ is exponentially distributed with mean $a_0(\mathbf{X}(t))$. Then $a_j(\mathbf{X}(t))/a_0(\mathbf{X}(t))$ is the probability that the j-th reaction occurs.

```
1  while t < T do:
2    Calculate Propensity functions: a_i.
3    Calculate a_0(X(t)) = sum_{i=1}^{M} a_i(X(t)).
4
5    Sample two random numbers r_1, r_2 from a uniform
       distribution.
6
7    Select tau = -(1/a_0) ln(r_1).
8
9    Select the reaction j to fire such that
       sum_{k=1}^{j-1} a_k < r_2 a_0 < sum_{k=1}^{j} a_k .
10
```

```
11      Update X according to ν_j:
12      Let X(t+τ) = X(t) + ν_j .
13      Let t = t + τ .
14    end;
```

Listing 1 – *Main loop pseudocode of the SSA algorithm.*

The SSA requires the calculation of the propensity function vector $\mathbf{a}$ as well as the computation of the cumulative sum over it. This makes the SSA computationally expensive.

2.2 Logarithmic direct method (LDM)

The SSA spends most of its computational time in selecting the next firing reaction. The cumulated sums of propensity functions a_i are computed in line 9 of Listing 1. A linear search with the computational complexity of $O(M)$, where M is the number of reactions, is used to find the index μ of the next firing reaction. The idea of the logarithmic direct method (LDM) proposed by Li and Petzold [10] is to replace the linear search with a binary search of complexity $O(log\, M)$. The LDM is an exact and efficient re-formulation of the SSA.
The SSA computes the sum of all propensity functions almost twice. First in line 3 and second in line 9 of Listing 1. The linear search in line 9 terminates as soon as the next firing reaction is found. If one stores the partial sums $\sum_{i=1}^{j} a_i,\ j = 1, \ldots, M$ instead of the propensity functions $a_i,\ i = 1, \ldots, M$, the result is an ordered list. The binary search algorithm requires such an ordered list to select the next firing reaction. It divides the search interval in half repeatedly. Each step has to decide between only two alternatives, to which half the key belongs [9]. The key to be searched is $r_2 a_0$ as in the SSA. In contrast to the linear search, the average search depth of the binary search is independent of ordering the reactions [10].

2.3 Next reaction method (NRM)

The next reaction method (NRM) is an efficient and exact re-formulation of the SSA proposed by Gibson and Bruck [3]. Rather than using relative time until the next reaction event occurs, the NRM uses absolute time. It computes the absolute putative time of the next reaction event for all M reactions and stores them in an indexed priority queue. The next reaction is that with the smallest absolute putative time. The pseudocode of the NRM is given in listing 2 below.
Since not all propensities have to change with each reaction event, only those whose reactants are affected by the currently firing reaction are updated. The NRM uses a dependency graph to identify those reactions. Figure 3 exemplifies the dependency graph using the Oregonator reaction system [13]. The dependency graph $G = (V, E)$ is a set of nodes V and a set of edges E. There is a node $V = 1, \ldots, M$ for each reaction. G is a directed graph. A directed edge from node k to node l means if reaction k fires, the propensity function of reaction l changes and needs to be updated.

$$
\begin{aligned}
X_1 + Y_2 &\rightarrow Y_1 && (1)\\
Y_1 + Y_2 &\rightarrow Z_1 && (2)\\
X_2 + Y_1 &\rightarrow 2\ Y_1 + Y_3 && (3)\\
2\ Y_1 &\rightarrow Z_2 && (4)\\
X_3 + Y_3 &\rightarrow Y_2 && (5)
\end{aligned}
$$

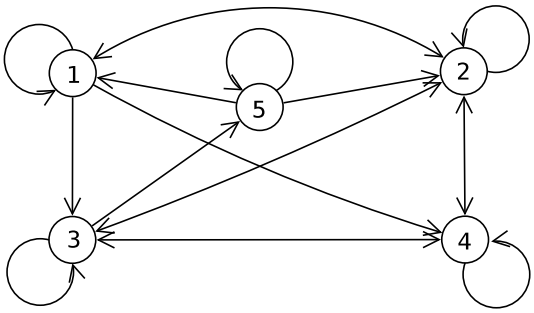

Figure 3 – *Dependency graph of the Oregonator reaction system [13]. The chemical reaction system of the Oregonator is given on the left. The numbers in brackets correspond to the node labels in the dependency graph on the right. A directed edge from node A to node B means that if reaction A fires, then the propensity function of reaction B changes.*

The indexed priority queue is implemented using a heap data structure [9]. To find the next firing reaction, the heap sort algorithm is used. The usage of the fairly complex heap data structure and heap sort algorithm has been criticised by Cao et al. [2].

```
1  Initialise outside the main loop dependency graph G,
     propensity functions a_i, initial putative reaction
     times τ = τ_1,...,τ_M stored indexed priority queue P;
2
3  while t < T do;
4    Let μ be the index of min(τ), let τ be τ_μ;
5
6    Update X according to ν_μ:
7    Let X(t+τ) = X(t) + ν_j .
8
9    For each edge (μ,α) in G:
10   (a) update a_α,
11   (b) if α ≠ μ, set τ_α = (a_{α,old}/a_α)(τ_α − t) + t,
12   (c) If α = μ, generate a random number ρ according
13   to an exponential distribution with parameter a_μ, and
14   set τ_α = ρ + t,
15   (d) update τ_α in P;
16 end;
```

Listing 2 – *Main loop pseudocode of the NRM algorithm.*

The putative absolute times of the next reaction for each reaction channel are in general an unsorted array of numbers of which we need to find the minimal element. The run-time of a naive linear search is $O(n)$ where n is the length of the input array. Instead, Gibson and Bruck use the more efficient heap-sort algorithm [3]. Its run-time is $O(n\ log\ n)$ [9].

The NRM exploits the fact that not all propensity functions change when a reaction event occurs. The less the dependency graph is populated, the fewer propensity functions need to be updated when a reaction event occurs. We expect the NRM to perform better for reaction systems for which the dependency graph is sparsely populated.

3 Threading approach and memory usage

The aim is to maximize the usage of shared memory and registers to keep the data required by the threads as "close" in terms of memory access time and latency as possible. This means to copy a working set of data into shared memory and process it by as many threads as possible in parallel [15]. Regarding the SSA and LDM implementation this is achieved by:

1. Loading the initial population vector $\mathbf{X}(0)$ from device memory to shared memory, initialising propensity function vector $\mathbf{a}$ and setting pointers to reaction rates and stoichiometric reaction matrix in constant memory,
2. Synchronising with all the other threads of the block so that each thread can safely execute the chosen algorithm in shared memory and performing the simulation in shared memory,
3. Each thread writing its current molecular population vector $\mathbf{X}(t)$ back to device memory as required.

Shared memory The number of threads per block is restricted by the available shared memory which again may hamper the overall performance. Both the propensity vector and the population vector depend on the size of the reaction system and with it the maximum number of possible threads per block. The pseudo random number generator (PRNG) state is stored in registers.
For each thread, the state vector $\mathbf{X}(t)$ of length N and the propensity function vector $\mathbf{a}$ of length M need to be stored in shared memory. Additionally, 128 bytes (32 32-bit wide entries) of shared memory are statically and automatically allocated by the compiler. This gives a per block shared memory requirement in bytes of:

$$\text{memory}_{\text{shared}} = ((M+N) \times \text{threads per block} + 32) \times 4 \ . \tag{1}$$

Re-arranging this for the number of threads gives the upper bound:

$$\text{threads per block} \leq \left\lfloor \frac{4096 - 32}{M+N} \right\rfloor \ . \tag{2}$$

The NRM requires additional shared memory. The dependency graph G needs to be traversed in each iteration but it is constant and can be pre-computed. It is stored in constant memory. The priority queue is updated in each iteration, so it needs to be stored in shared memory.

The storage requirement of the priority queue is $3 \times M$ (using 16-bit integers to conserve shared memory space). Considering the same reaction system, the maximum block size of the NRM is smaller than that of the SSA or LDM.
The shared memory requirement of the NRM in bytes is:

$$\text{memory}_{\text{shared}} = ((3 \times M + N) \times \text{threads per block} + 32) \times 4 \ . \tag{3}$$

Re-arranging this for the number of threads gives the upper bound:

$$\text{threads per block} \leq \left\lfloor \frac{4096 - 32}{3 \times M + N} \right\rfloor \ . \tag{4}$$

Given that the minimum efficient block size is 64 threads or 2 warps [15], the maximum size of the reaction system is given by $\lfloor N + M \rfloor \leq 63$. For the NRM $\lfloor 3 \times 3 \times N + M \rfloor \leq 63$.
Let us illustrate this with the Oregonator system given in Figure 3 [13]. The Oregonator system consists of $M = 8$ molecular species and $N = 5$ reactions. The dependency graph given in Figure 3 requires 18 32-bit entries of shared memory. The maximum number of threads per block is 213 for the SSA and 139 for the NRM, respectively.
The number of the histogram bins B is limited by the size of shared memory to

$$B = \frac{4096}{M} \tag{5}$$

where M is the number of moelcular species.

Constant memory The maximum size of the stoichiometric reaction matrix $\boldsymbol{\nu}$ can not be given in terms of the number of reactions and species, but in terms of non-zero entries. The GPU's constant memory is used to store the reaction matrix as sparse, the vector of reaction rates and for the NRM the dependency graph. The total size of constant memory is 64 kB or 16384 32-bit floating point entries. The constant memory is cached by a constant cache of 8 kB per SM [15].
The stoichiometric reaction matrix $\boldsymbol{\nu}$ is stored in the compressed row storage (CRS) sparse matrix format. The storage requirement of the sparse matrix using 32-bit float values of M rows and nnz non-zero entries is $M + 2 \times nnz$ 32-bit entries. The storage requirement of the dependency graph of M nodes and $|E|$ edges is $M + |E|$ 32-bit entries.
Regarding the SSA and LDM, the maximum number of non-zero entries in the reaction matrix is:

$$nnz_{max,\text{SSA}} = \left\lfloor \frac{16384 - 2 \times M}{2} \right\rfloor , \tag{6}$$

and regarding the NRM:

$$nnz_{max,\text{NRM}} = \left\lfloor \frac{16384 - 3 \times M - |E|}{2} \right\rfloor \ . \tag{7}$$

Device memory The number of realisations and the number of samples to be stored per realisation depends on the size of the available device memory [3]. Let S be the number of

[3] For the Tesla architecture the maximum size of the device memory is 4 GB, for the Fermi architecture it is 6 GB.

samples stored for each realisation. The the product realisations $\times S \times N \times 4$ in bytes has to fit into the device memory.

4 Sampling regimes

The implemented exact stochastic simulation algorithms, SSA, LDM and NRM, compute the time of the next occurring reaction by sampling from an exponential distribution. This means reaction events do not occur in equidistant time steps.
The available device memory of the graphics card is limited [4] and reaction events can be numerous, so we do not want to store every reaction event. Reaction events can be sampled non-equidistantly whenever a reaction event occurs resulting in a dynamic grid of reaction times which differs for each realisation of the simulation.
Another option is to sample at fixed time increments t_{ref}. This means the molecular population is sampled when a time t_{ref} passes regardless of the actual number of reaction events that occur. This makes it easier, for example, to calculate the average molecular population of multiple simulations. Both options are illustrated in Figure 4.

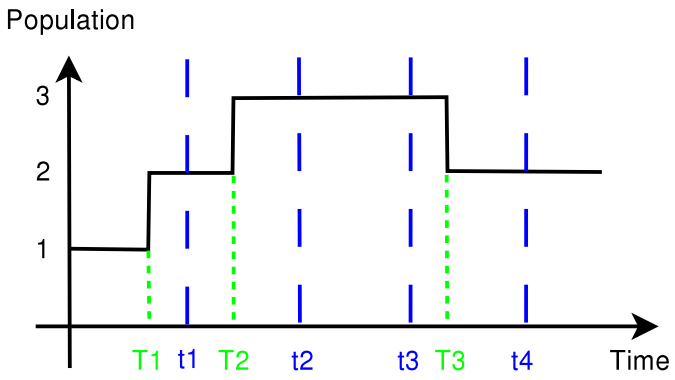

Figure 4 – Sampling the molecular population at fixed time increments or whenever a reaction event occurs.

5 Features and limitations of the stochastic simulation software

The main feature of our software is to compute ensembles of realisations of stochastic simulations in parallel on a GPU without assuming any knowledge about GPUs on behalf of the user. This is achieved by integrating it into MATLAB via a direct plugin to the Systems

[4] The Nvidia GTX 260 card used has 896 MB of device memory.

Biology Toolbox 2 for MATLAB (an opensource project by [16]) [5]. It replaces its MATLAB based implementation of stochastic simulations of well-mixed chemical reaction systems with the GPU-based implementation. This does not require any change to the user's code. The features of our software are:

- Parallel simulation of ensembles of well-mixed chemical reactions,
- Implementation of exact stochastic simulation algorithms:
  - Stochastic simulation algorithm (SSA) [5],
  - Next reaction method (NRM) [3],
  - Logarithmic direct method (LDM) [10],
- Integration into MATLAB and the Systems Biology Toolbox 2 for MATLAB [16],
- Histogram computation of the molecular populations,
- Computation of the average of the computed realisations,
- Fixed and dynamic sampling grid,
- The implementation uses sparse matrices to store the chemical reaction system to maximise the possible size of the reaction system to be simulated.

Beyond the limitations of the available device, constant and shared memory of the GPU used, the reaction kinetics of the chemical reaction system is limited to the law of mass action.

6 Hardware and software environment used

All simulations were computed using a NVIDIA GTX 260 GPU with 216 cores and 896 Mb DDR3 RAM. The Host CPU is an Intel Core 2 Duo at 2.16 GHz and 4 GB RAM. The Software environment is OpenSuse 11.1 with gcc 4.2.3, NVIDIA CUDA 3.0 and MATLAB 2010a. The stochastic simulation software requires a GPU with compute capability 1.2 or higher.

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, New York, 2007.

[2] Y. Cao, H. Li, and L. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *Journal of Chemical Physics*, 121(9):4059–4067, 2004.

[3] M. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104:1876–1889, 2000.

[5] The Systems Biology Toolbox 2 for Matlab is available at: `http://www.sbtoolbox2.org`

[4] D. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational physics*, 22:403–434, 1976.

[5] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[6] J. Hasty, D. McMillen, F. Isaacs, and J. Collins. Computational studies of gene regulatory networks: In numero molecul biology. *Nature Review Genetics*, 2(4):268–279, April 2001.

[7] John P. Hayes. *Computer architecture and organization*. McGraw-Hill, 3rd edition, 1998.

[8] F. Holstege, E. Jennings, J. Wyrick, T. Lee, C. Hengartner, M. Green, T. Golub, E. Lander, and R. Young. Dissecting the regulatory circuitry of a eukaryotic genome. *Cell*, 95(5):717–728, Nov 1998.

[9] Donald E. Knuth. *The Art of Computer Programming - Sorting and Searching*, volume 3. Addison Wesley, 2nd edition, 1997.

[10] H. Li and L. Petzold. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. Technical report, Department of Computer Science, University of California Santa Barbara, 2006. URL `http://www.cs.ucsb.edu/~cse/publications.php`.

[11] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Computer Society Hot Chips*, (19):39–45, March/April 2008.

[12] P. Maciol and K. Banas. Testing Tesla arcitecture for scientific computing: the performance of matrix-vector product. *Proceedings of the International Muticonference on Computer Science and Information Technology*, 3:285–291, 2008.

[13] J. Murray. *Mathematical Biology 1: An Introduction*, volume 1 of *Interdisciplinary applied sciences*. Springer Verlag, 3rd edition, 2002.

[14] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *ACM Queue*, 6(2):40–53, March/April 2008.

[15] NVIDIA. *NVIDIA CUDA Programming Guide, Version 2.1*. NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, California, 12 2008.

[16] Henning Schmidt and Mats Jirstrand. Systems Biology Toolbox for MATLAB: a computational platform for research in Systems Biology. *Bioinformatics*, 22(4):514–515, February 2006. http://www.sbtoolbox2.org.

[17] T. Tian and K. Burrage. Stochastic models for regulatory networks of the genetic toggle switch. *Proceedings of the National Academy of Sciences*, 103(22):8372–8377, 2006.