GPU-based Steady-State Solution of the Chemical Master Equation

Marco Maggioni, Tanya Berger-Wolf, Jie Liang
Department of Computer Science, Department of Bioengineering
University of Illinois at Chicago
Email: {mmaggi3,tanyabw,jliang}@uic.edu

***Abstract*—The Chemical Master Equation (CME) is a stochastic and discrete-state continuous-time model for macromolecular reaction networks inside the cell. Under this theoretical framework, the solution of a sparse linear system provides the steady-state probability landscape over the molecular microstates. The CME framework can in fact reveal important insights into basic principles on how biological networks function, having critical applications in stem cell study and cancer development. However, the exploratory nature of system biology research involves the solution of the same reaction network under different conditions. As a result, the application of the CME framework is feasible only if we are able to solve several large linear systems in a reasonable amount of time.**

In recent years, GPU has emerged as a cost-effective high performance architecture easily available to bioscientists around the world. In this paper, we propose an efficient GPU-based Jacobi iteration for steady-state probability calculation. We provide several optimization strategies based on the problem structure with the aim of outperforming the conventional multicore implementation while minimizing the GPU memory footprint. We combine an ELL+DIAG sparse matrix format with DFS ordering to leverage the diagonal density. Moreover, we devise an improved sliced ELL sparse matrix representation based on warp granularity and local rearrangement.

Experimental results demonstrate an average double-precision performance of 14.212 GFLOPS in solving the CME (a speedup of 15.67x compared to the optimized Intel MKL library). Our implementation of the warp-grained sliced ELL format outperforms the state-of-the-art in terms of SpMV performance (a speedup of 1.24x over clSpMV). Moreover, it shows consistent improvements for a wider set of application domains and a good memory footprint reduction. The results achieved in this work provide the foundation for applying the CME framework to realistic biochemical systems. In addition, our GPU-based steady-state computation can be generalized to operation on stochastic matrices (Markov models), achieving good performance with matrix structures similar to biological reaction networks.

***Keywords*-GPU; sparse linear algebra; chemical master equation; system biology; Jacobi iteration; computational biology;**

I. Introduction

Networks of interacting biomolecules are at the heart of the regulation of cellular processes, and stochasticity plays important roles. The Chemical Master Equation (CME) [1] is a stochastic and discrete-state continuos-time formulation that provides a fundamental framework to model biochemical reaction networks inside the cells. In general, an accurate solution to the CME can reveal important insights into basic principles on how biological networks function and how they respond to various environmental perturbations, having critical applications in stem cell study and cancer development.

Under the CME framework, a stochastic characterization of the biochemical system is provided by a time-varying probability landscape over microstates representing the detailed chemical amount of each and every molecular species. The stationary behavior at the limit can be analyzed to identify biologically meaningful macrostates. This step involves the solution of a sparse linear system of equations representing stochastic microstate transitions. Despite the assumption of small copy numbers in a cell, the CME framework poses a computational challenge when applied to realistic systems due to an exponential growth in the number of microstates [2]. Moreover, the exploratory nature of system biology research involves the study of the same reaction network under different conditions (e.g. varying the intrinsic rate of one of the involved reactions). The computational demands may become overwhelming and, hence, practically infeasible for computer architectures based on a limited number of parallel processing elements such as conventional multicore CPUs. This observation motivates the use of high performance computer architectures to solve the steady-state probability landscape problem.

Over the past years, GPUs have evolved from an application-specific processors dedicated to 3D visualization into a more general purpose parallel platform available for scientific computing. Bioscientists around the world can easily have access to this high performance architecture due its cost-effectiveness. As a consequence, the solution of increasingly complex scientific problems can be computed in practice, advancing science in novel directions. The stationary analysis of biochemical reaction network can certainly benefit of the high floating-point throughput and memory bandwidth peaks offered by modern GPU architecture like Fermi [3]. Specifically, the availability of a large memory bandwidth permits to mitigate the bandwidth-bound nature of sparse linear algebra operations necessary to calculate the CME solution. Due to the centrality of linear algebra in scientific and engineering computations, a large research effort has been dedicated to improve efficiency and memory representation compactness of GPU-based Sparse Matrix-Vector multiplication (SpMV) [4, 5, 6, 7, 8]. This body of work can be taken as foundation for implementing the Jacobi iteration [9], a well-known and easy-to-parallelize iterative method for solving linear systems.

In this paper, we propose an efficient GPU-based Jacobi iteration for steady-state probability landscape calculation. Similarly to what done in [10] for graph mining, we analyze the structure of problems arising from the CME and we provide several optimization strategies with the aim of outperforming the conventional multicore implementation while minimizing the GPU memory footprint. First, the transition matrix can be efficiently represented using the ELL format [4] due to a limited number of possible reactions for each microstate. Second, it is possible to take advantage of reversible reactions in order to improve the diagonal density of the transition matrix. Specifically, the DFS ordering of the microstates exposes a densely populated band composed by the main diagonal and its two immediate neighbors. We can leverage this structure by separately storing the dense diagonals using the DIA format [4]. In this paper, we also propose a novel sparse matrix representation that optimizes the sliced ELL format presented in [5]. The basic idea is to reduce as much as possible the overhead associated with the data structure. This goal is achieved by following two strategies. First, the slice size is chosen in order to match the execution block in hardware (warp). Second, we apply a local rearrangement that uniforms the number of nonzeros within each slice and improves the data structure efficiency without deteriorating the cache locality. This warped-grained sliced ELL can be also combined with the DIA optimization, at least when the local rearrangement does not decrease too much the diagonal density.

The use of GPUs to solve large Markov models, a topic close to the research presented in this paper, has been discussed in previous literature. The authors of [11] implements a GPU-based Jacobi iteration. Besides being limited by an out-of-date GPU architecture, their work is based on the general CSR format [12] and does not propose any optimization strategy. Analogously, the authors of [13] do not introduce any significant contribution other than testing well-known GPU-based SpMV kernels on some matrices derived from Markov models. The novelty of this paper lies in specific GPU optimization strategies to achieve better performance and memory footprint. Moreover, we propose the first practical implementation of the CME stochastic framework. The ability to study realistic cellular reaction networks opens up a new direction of biological computing that will be, without exaggeration, every bit as important as molecular dynamics simulation. The specific contributions of this work are the followings:

- An efficient GPU-based Jacobi iteration for steady-state probability calculation of biological reaction networks, which can be generalized to operation on stochastic matrices (Markov models). On a dataset of transition matrices derived from biological models, we were able to reach an average double-precision performance of 14.212 GFLOP/s and a speedup of 15.67x compared to the optimized Intel MKL library [14].
- An improved sliced ELL sparse matrix representation based on warp granularity and local rearrangement. We are able to show that this optimized format outperforms the state-of-the-art in terms of SpMV performance on matrices arising from the CME framework (a speedup of 1.24x over clSpMV [15], a framework capable of selecting the best representation of any sparse matrix). Considering a wider set of application domains, the warp-grained ELL format has consistent improvements compared to the original sliced ELL format. Moreover, it minimizes the footprint of the underlying data structure (a relevant factor due to the limited memory available on current GPU devices).

The structure of the paper is as follows. Section II introduces the theoretical background of the CME. Section III presents a brief description of the GPU architecture. Section IV introduces the Jacobi iterative method. Section V provides a detailed analysis of the problem structure and devises some domain-specific GPU optimizations whereas Section VI describes the novel warp-grained ELL format. Performance results of the various optimization strategies are then given in Section VII. Finally, Section VIII is devoted to conclusions.

II. The Chemical Master Equation

In the theory of the Chemical Master Equation, the dynamics of a biochemical reaction system, in a small volume, is represented by a discrete-state continuous-time Markov process and is characterized by a probability distribution as function of time. The CME basically describes the change of probability of different microstates connected by reactions.

A. Stochastic framework

The state space of the CME is represented by a detailed amount of each and every of the m molecular species in the biochemical reaction network. The microstate of the system at time t is then defined as $\mathbf{x}(t) = \{x_1(t), x_2(t), ..., x_m(t)\} \in \mathbb{N}^m$ and the overall state space is the set $\mathcal{X}$ of all possible combination numbers $\mathcal{X} = \{\mathbf{x}(t) | t \in (0, \infty)\}$. A fundamental characterization of the system is given by the collection of probabilities $\mathbf{P}(t) \in [0,1]^{|\mathcal{X}|}$ for each of the microstate at time t. $\mathbf{P}(t)$ is defined as the probability landscape.

The transition rates between microstates ($\mathbf{x}_j \xrightarrow{k} \mathbf{x}_i$) connected by the k-th reaction are determined by the intrinsic reaction rate r_k and by reactants involved

$$A_k(\mathbf{x}_i, \mathbf{x}_j) = A_k(\cdot, \mathbf{x}_j) = A_k(\mathbf{x}_j) = r_k \prod_{i=1}^{m} \binom{x_i}{c_i}$$

where c_i is the copy number of species i needed to perform the reaction ($A_k(\mathbf{x}_i, \mathbf{x}_j) > 0$ when two microstates are connected by the k-th reaction, $A_k(\mathbf{x}_i, \mathbf{x}_j) = 0$ otherwise). In principle, the transition $\mathbf{x}_j \longrightarrow \mathbf{x}_i$ between two microstates can correspond to different reactions, so the overall reaction rate is

$$A(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k \in \mathcal{R}} A_k(\mathbf{x}_i, \mathbf{x}_j)$$

The discrete Chemical Master Equation can then be written as

$$\frac{d\mathbf{P}(\mathbf{x}, t)}{dt} = \sum_{\mathbf{x}' \neq \mathbf{x}} \Big[A(\mathbf{x}, \mathbf{x}')\mathbf{P}(\mathbf{x}', t) - A(\mathbf{x}', \mathbf{x})\mathbf{P}(\mathbf{x}, t) \Big]$$

where $\mathbf{P}(\mathbf{x}, t)$ is the probability in continuous time of a discrete microstate. The gain and loss of probability associated with each microstate is a balance between the incoming and the outcoming reaction rates. The CME can also be written in a more compact form by defining the rate constant for leaving the current state $A(\mathbf{x}, \mathbf{x})$ as

$$A(\mathbf{x}, \mathbf{x}) = -\sum_{\mathbf{x}' \neq \mathbf{x}} A(\mathbf{x}', \mathbf{x})$$

Consequently, the probability landscape variation can be described in the following matrix-vector form

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{A}\mathbf{P}(t)$$

where $\mathbf{A} \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$ is a sparse reaction rate matrix formed by the collection of all $A(\mathbf{x_i}, \mathbf{x_j})$.

B. Steady-state probability landscape

The stationary behavior of this stochastic model can provide biological insight about the underlying biochemical reaction network. Given the reaction rate matrix $\mathbf{A}$, the steady-state probability landscape $\mathbf{P}$ over the microstates is directly derived from the CME as

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{A}\mathbf{P} = \mathbf{0}$$

Intuitively, the steady-state corresponds to the condition $\frac{d\mathbf{P}(t)}{dt} = \mathbf{0}$. Therefore, we can derive $\mathbf{P}$ by solving the correspondent system of linear equations $\mathbf{A}\mathbf{P} = \mathbf{0}$.

We can show an example of the CME stochastic framework use. We consider the well-studied genetic toggle system [16], a network composed by two genes A and B arranged in mutual inhibition. This model is depicted in Figure 1. The intuitive bistable behavior would see one gene synthesizing its protein (gene "on") while the other is inhibited (gene "off"). The analysis of the steady-state probability landscape should provide an insight of such bistability. Hence, we can solve the linear system $\mathbf{A}\mathbf{P} = \mathbf{0}$ in order to derive the landscape shown in Figure 2. In this example, the CME stochastic framework is able to provide a connection between the microstate probability distribution and the macroscopic behavior of the biochemical reaction network. Specifically, the probability is concentrated around the states with mutual inhibition ("on/off" and "off/on") where only copies of one protein (nA or nB) are present.

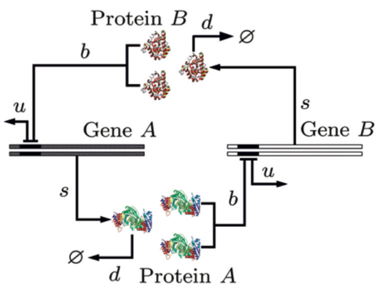


Figure 1. Model of a toggle switch [17]

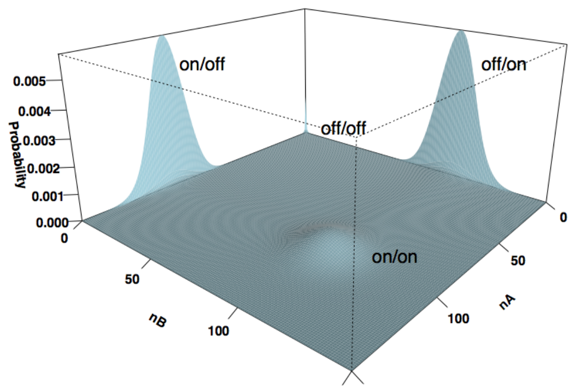


Figure 2. Steady state probability landscape of a toggle switch [17]

A reliable stochastic model of the reaction network requires a complete identification of the microstates. The corresponding state space $\mathcal{X}$ considers all the possible combination numbers and grows exponentially with the number m of molecular species. A rough bound for this state space size is given by $|\mathcal{X}| \leq k^m$, where k is the maximum copy number for each species. However, it might be argued that the structure of the reaction network limits the microstates effectively reachable from an initial condition. This intuition has been used in [17] to produce a minimal and comprehensive state space $\mathcal{X}$. Specifically, it is possible to identify a graph structure where microstates are nodes and reactions are edges. A DFS visit starts from an initial microstate and produces the reachable subspace $\mathcal{X}$, along with the associated reaction rate matrix $\mathbf{A}$.

The concrete application of the CME stochastic framework is not a trivial task. First, the arising linear systems are intrinsically ill-conditioned posing numerical problems during the solution. Second, the size $n = |\mathcal{X}|$ quickly poses a practical feasibility issue for computer architectures with limited parallelism. Last, bioscientists usually study a reaction network under different conditions. Considering that each combination of the parameters generates a different linear system, the total amount of computation may become excruciatingly large.

III. GPU ARCHITECTURE

The architecture of modern GPUs has evolved into a general purpose parallel platform optimized for computing-intensive data processing. The idea of hardware multithreading is central for this architecture. In brief, the availability of a large pool of threads ready to execute (coupled with fast context switching) can keep functional units busy and hide memory accesses. Hence, the use of large and complex cache memories becomes less crucial and more silicon area can be dedicated to implement computational cores. In this subsection we describe the key aspects of NVIDIA Fermi architecture [3]. We refer to this particular architecture because our experimental results are based on the CUDA programming model [18] and on the specific NVIDIA GTX580 device .

A GPU is composed by a number of processing units called Streaming Multiprocessors (SMs), each one containing a set of simple computing cores known as Streaming Processors (SPs) or CUDA cores. Referring to GTX580, we have 16 SMs with 32 SPs each for a total of 512 cores (a potential parallelism of 512 operations for clock cycle). The execution of instructions within SMs follows the single-instruction multiple-thread (SIMT) model with warp granularity (32 threads). Whenever threads within the same warp needs to execute a different branch of instructions, we have a divergence and the execution is serialized with intrinsic performance decreasing.

SMs are connected to a random access memory through a cache hierarchy. This global memory has high bandwidth (192 GB/s for GTX580) but high latency (up to 800 clock cycles). The cache hierarchy is organized on two levels. A coherent L2 cache (768 KB) is shared among all the SMs and provides a mean to reduce the global memory bandwidth usage (whereas latency is not improved). Each SM also has a local on-chip memory (64KB) with low latency and very high bandwidth ($\approx$ 3.15 TB/s). This fast memory can be split (16KB + 48KB) to work as L1 cache (hardware managed) or as shared memory (software managed). In addition, each SM has a large register file composed by 2^{15} 32-bit registers (used in pairs in case of double-precision arithmetic) with very low latency and very high bandwidth ($\approx$ 9.24 TB/s). Fast context switching between warps is possible by statically assigning different registers to different threads. A crucial factor for GPU efficiency is the memory access pattern. The Fermi architecture is indeed optimized for regular accesses. In more detail, the memory requests within a warp are converted into L1 cache line requests (128 bytes). Hence, memory performance is maximized only when memory addresses can be coalesced into a single 128-byte aligned transaction (opposed to inefficient scattered accesses).

Despite the availability of a large memory bandwidth, algorithms with low arithmetic intensity are able to achieve only a fraction of the theoretical performance peak. The Fermi architecture implements a fused multiple-add (FMA) instruction for which two floating point operations are executed in a single cycle. Considering a bandwidth of 192 GB/s and the need of loading double-precision operands from global memory, we can perform at most 12 GFLOPS (each FMA needs 4 doubles, or 32 bytes). This performance is way below to the Fermi architecture theoretical peak ($\approx$ 789 GB/s). It might be also argued that gaming-oriented GPUs (such a GTX580) have a performance edge over computing-oriented GPUs (such as Tesla) for double-precision sparse linear algebra. In more detail, GTX580 offers a larger bandwidth despite having a double-precision performance peak locked at one-quarter of the chip's peak potential ($\approx$ 197 GFLOPS). Hence, it can potentially offer a better performance for bandwidth-limited algorithms such as sparse linear algebra.

CUDA is an abstract parallel computing architecture and programming model based on few concepts. First, there exists a hierarchy of concurrent lightweight threads to model the computation according to a data-parallel model. In brief, the algorithm should be expressed in such a way each data element is processed by an independent lightweight thread. All threads execute the same program on different data. Threads are logically grouped into equally sized blocks. Cooperation and synchronization within blocks are allowed by the means of shared memory and primitives. Finally, blocks are grouped into a grid for covering all the data to process. This hierarchy represents an abstraction of the underlying GPU architecture. Specifically, blocks represent abstract SMs capable to run all the threads simultaneously. However, a block is permanently assigned to one of the available SMs and executed as warps in an arbitrary order. This approach assures scalability since CUDA code can be compiled and executed on devices with different number of multiprocessors. Referring to GTX580, each SM can manage up to 1536 threads (forming up to 48 warps). Considering all the SMs, it is possible to reach a massive parallelism with up to 24576 simultaneous threads. We define occupancy as the number of active threads compared to the maximum capacity. This metric is important in order to provide an effective hardware multithreading. Intuitively, a large number of warps ready to execute is useful to hide memory latency. However, active threads depend by a combination of blocks assigned to a SM and size of such blocks. Given an hardware limit of 8 blocks per SM, the choice of block size becomes critical. First, the block size should be a multiple of the warp size in order to avoid partially unused warps. Second, the block size should be big enough in order to cover the maximum SM occupancy with exactly or less than 8 blocks. Third, a very large block may not be always a good choice.

For example, a block size of 1024 threads cannot achieve full occupancy since only one block fits the SM. Moreover, a block size of 512 threads provides full occupancy but the hardware should wait the completion of 16 warps before allocating a new block. Intuitively, a block size of 256 may provide slightly better performance because full occupancy is associated with a better block turnover.

IV. Jacobi iteration

The Jacobi iteration is a simple but easy-to-parallelize stationary iterative method to solve a system of linear equations. The main idea behind this approach is to construct an iteration matrix $\mathbf{M}$ that is applied, step after step, to improve an approximate solution. Assuming $\mathbf{b} = \mathbf{0}$, the Jacobi iteration is defined as

$$\mathbf{x^{(k+1)}} = \mathbf{M}\mathbf{x^{(k)}} = -\mathbf{D^{-1}}(\mathbf{L}+\mathbf{U})\mathbf{x^{(k)}}$$

where $\mathbf{L}$, $\mathbf{D}$ and $\mathbf{U}$ are respectively the strictly lower triangular, the diagonal and the strictly upper triangular part of the original matrix $\mathbf{A}$. Moreover $\mathbf{x^{(k+1)}}$ and $\mathbf{x^{(k)}}$ represent the iterative solution at step $k+1$ and k.

The convergence of the Jacobi method is not guaranteed in general and depends by the spectral radius $\rho(\mathbf{M})$. Specifically, the condition necessary and sufficient for convergence is

$$\rho(\mathbf{M}) < 1$$

where the converge rate is fast only when $\rho(\mathbf{M})$ is close to zero. Despite this drawback, the Jacobi iteration is a very popular technique due to the intrinsic level of parallelism available to speed up the convergence to the solution. This useful aspect can be highlighted by the following component-wise formulation

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[-\sum_{j\neq i} a_{ij}x_j^{(k)}\right]$$

where it is clear that any component $x_i^{(k+1)}$ can be calculated independently from the others. Moreover, it might be argued that the Jacobi iteration is computationally similar to SpMV except for a division involving the diagonal nonzero.

The Jacobi iteration starts from an initial solution $\mathbf{x^0}$ and continues until convergence is detected. Since the right-hand side vector is $\mathbf{b} = \mathbf{0}$, we normalize the infinity norm $\|\mathbf{r^{(k)}}\|_\infty$ of the residual vector respect to the matrix norm $\|\mathbf{A}\|_\infty$ and the solution norm $\|\mathbf{x^{(k)}}\|_\infty$. Then, we check if this normalized value is less than some predefined error ϵ

$$\frac{\|\mathbf{r^{(k)}}\|_\infty}{\|\mathbf{A}\|_\infty \cdot \|\mathbf{p^{(k)}}\|_\infty} \leq \epsilon$$

A practical stopping criterion should also limit the number of iterations and consider when the error is no longer decreasing or decreasing too slowly (stagnation). Applying the following formula

$$\frac{\|\mathbf{r^{(k+1)}}\|_\infty - \|\mathbf{r^{(k)}}\|_\infty}{\|\mathbf{r^{(k)}}\|_\infty} \leq \epsilon$$

we can monitor the error variation between successive iterations. It might be argued that the calculation of the residual vector $\mathbf{r^{(k+1)}}$ is approximatively as expensive as the Jacobi iteration. Therefore, it is reasonable to check the stopping criterion only once every several iterations.

The calculation of the steady-state probability landscape requires that the dense vector $\mathbf{x}$ is a probability vector. In other words, $\forall i, x_i \geq 0$ and $\|\mathbf{x}\|_\mathbf{1} = 1$. The Jacobi iteration should be able to keep this property during the solution process. Given an initial probability vector $\mathbf{x^0}$, the first condition always holds since the reaction rate matrix is composed by all positive nonzeros except for the diagonal. However, the second condition for being a probability vector may be violated during the iterations. Therefore, we need to periodically normalize $\mathbf{x^{(k)}}$ in order to produce a probability vector.

We should briefly mention alternative iterative methods such as those based on Krylov-subspace. Despite these linear algebra methods usually guarantee a faster convergence, we preferred to use the Jacobi iteration due to numerical stability reasons. Specifically, the linear system arising from the CME stochastic framework are ill-conditioned and singular. In fact, we performed some preliminary studies on using GMRES (Generalized Minimal RESidual) [19] for solving the steady-state problem but we observed no convergence. Hence, we primarily focused on the Jacobi iteration.

V. Domain-specific GPU optimizations

The goal of this section is to provide an efficient matrix format representation for the CME domain and to propose some optimizations to improve the performance of the associated Jacobi iteration. As mentioned, we build upon the foundation knowledge about SpMV on GPU, due to its computational similarity with the Jacobi iteration.

We can observe that reaction rate matrices arising from biochemical networks have a bounded number of nonzeros per row. In fact, there is a limited set of reactions that can trigger the transition from the current microstate to another one. Moreover, it is reasonable to assume that most of the microstates have enough copy numbers to allow all the possible reactions. Therefore, the number of nonzeros per row is reasonably regular and always close to the maximum. For such case, the ELL sparse matrix representation has been shown to be efficient [4]. This format is particularly well-suited to vector architectures. The basic idea of these formats is to compress a sparse $n \times m$ matrix using a dense $n \times k$ data structure, where k is the maximum number of nonzeros per row. In more detail, the sparse matrix is stored in memory by the means of two $n \times k$ dense arrays, one for nonzero values and one for column indices (row indices remain implicit as in a dense matrix). Hence, zero-padding is necessary for rows with less than k elements. The structure of the SpMV computation, using ELL-derived formats, follows the data-parallel model. In fact, a thread is assigned to each row in order to compute one element y_i of result vector $\mathbf{y}$. Moreover, simple strategies are used to optimize the memory access pattern. The $n \times k$ dense arrays are stored in column-major order for coalescing and padded to $n' = \lceil \frac{n}{warp} \rceil \cdot warp$ for 128-byte aligning.

The efficiency of the ELL data structure can be measured as $e = \frac{nnz}{n' \times k}$ where nnz is the amount of nonzeros in the original matrix. This metric evaluates the amount of zero padding necessary to fill the dense ELL matrices. Assume that each thread will iterate k times (no a priori knowledge about of the effective number of nonzeros in its row). When efficiency is $e \approx 1$, there is practically no wasted bandwidth. On the other hand, low efficiency will imply a lot of wasted bandwidth loading padding values. However, it is possible to mitigate this latter inefficiency by introducing a conditional statement. Let take the following code snippet as an example

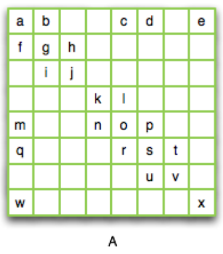
```
double temp = 0;
for (int i=0; i<k; ++i){
        double value = value[i*n+t];
        if (value!=0)
                temp += value * x[column[i*n+t]];
}
```

Listing 1. ELL SpMV code snippet

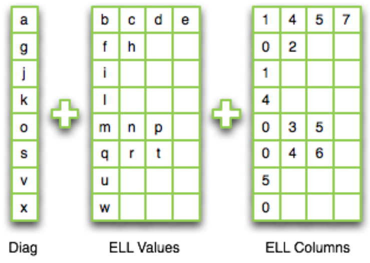
As we can see, we avoid two memory accesses (column indices and dense vector $\mathbf{x}$) when we load a padding value. Moreover, the possible divergence between adjacent threads does not create serialization since the **else** branch does not have any instruction. Therefore, there is a uniform execution among all the warps (except the last one when $n' \neq n$). The structure and the ordering of the sparse matrix determine the data locality of vector $\mathbf{x}$ accesses. The recent Fermi GPU architecture provides a memory hierarchy available to mitigate the sparse memory access inefficiency. A straight-forward way to fully take advantage of such feature is to configure the local on-chip memory in order to implement 48 KB of L1 cache (instead of the standard 16 KB).

The ELL format defines a precise arithmetic intensity for double precision. Given a FMA instruction, we need to load from memory the corresponding nonzero (8 bytes for the value and 4 bytes for the column) and to perform a sparse access to vector $\mathbf{x}$. Assuming no cache, this latter access will correspond to additional 8 bytes. The estimated performance peak for sparse linear algebra on GPU is then obtained by multiplying the arithmetic intensity $\frac{2}{8+4+8} = \frac{2}{20}$ with the available memory bandwidth (192 GB/s for GTX580). Hence, we obtain a performance peak of 20.6 GFLOPS with assumption of no cache. On the other hand, the additional 8 bytes contribution is not counted when we assume a perfect caching mechanism. In such case, the theoretical performance peak increases to 34.4 GFLOPS. The comparison with these two peaks gives an idea about the efficiency of the underlying memory system as well as about the data locality of the particular matrix structure. Moreover, we can conjecture that gaming boards such as GTX580 can be successfully used for sparse linear algebra despite an inferior double precision peak potential ($\approx$ 197 GFLOPS) compared to computing-oriented GPUs.

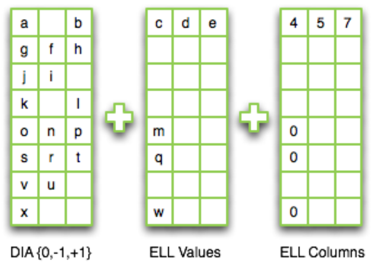
The analysis of reaction rate matrices arising from bio-chemical networks suggests further optimization strategies specific to this domain. By definition, the diagonal elements are $A(\mathbf{x}, \mathbf{x}) = -\sum_{\mathbf{x}' \neq \mathbf{x}} A(\mathbf{x}', \mathbf{x})$. Therefore, the diagonal $\mathbf{D}$ is densely populated by nonzeros and can be stored as a separate dense vector as shown in Figure 3(b).



(a) Sparse matrix



(b) ELL with separate diagonal



(c) DIA+ELL

Figure 3. Optimizing ELL format with diagonals

This separate representation does not require to store column indices for the diagonal elements. Therefore, the data structure size decrease by $4n$ bytes. Moreover, the use of a separate diagonal is useful for the Jacobi iteration. Specifically, the coefficients a_{ii} are readily available to perform division instead of being in an arbitrary position within the ELL structure.

In general, diagonal structure can be leveraged using a sparse matrix representation known as DIA [4]. We can observe that biochemical reaction networks include reversible reactions for which it is possible to jump back and forth between two adjacent microstates. If we enumerate these with two adjacent indices, we will expose a densely populated band composed by the main diagonal and its two immediate neighbors (respectively $\{-1\}$ and $\{+1\}$). It might be also argued that a DFS visit of the state space $\mathcal{X}$ creates chains of such reversible microstates and, hence, intrinsically arranges densely populated subregion around the diagonal band. Considering that the enumeration algorithm [17] already uses a DFS visit to create the reaction rate matrix, we can directly take advantage of this diagonal structure without any additional reordering as shown in Figure 3(c). .

The DIA sparse format is combined with the ELL format to store d diagonals of the sparse matrices as d contiguous dense vectors (adding alignment padding if necessary). A list of offset is used to identify the current position from the main diagonal ($\{0\}$, $\{-1\}$ and $\{+1\}$). The DIA format produces contiguous memory accesses to vector $\mathbf{x}$, although alignment only happens for offset multiple of 16. The combined format ELL+DIA is convenient in terms of memory efficiency only if the nonzero density within the diagonal band is greater than 0.66. This threshold derives from the memory required for storing a nonzero using the DIA format (8 bytes) and using the ELL format (12 bytes). The ELL+DIA format is well suited for the Jacobi iteration. The idea is to store the diagonal $\{0\}$ using the first column of the DIA matrix and use the remaining part (as well as the ELL structure) to calculate $-\sum_{j\neq i} a_{ij}x_j^{(k)}$.

VI. Warp-grained ELL format

The sliced ELL sparse matrix format [5] has been designed to improve the efficiency of the basic ELL data structure. This format is based on the idea of partitioning the matrix into slices representable with local ELL data structures. This approach has the practical advantage of reducing the zero-padding of each slice (which now depends on a local k dimension). The sliced ELL format needs additional data structures. First, we need an array K of size $\lceil \frac{n}{s} \rceil$ (where s is the slice size) in order to keep track of local k_i values. Second, we need another array of same size in order to identify the starting location in memory of each local ELL structure.

It is easy to see that a finer granularity decreases the amount of zero-padding values. On the other hand, this also decreases the SM occupancy. In more detail, the original sliced ELL formulation does not make a distinction between slice size s and block size b (in the CUDA programming model). Suppose, $s =$ warp. This slice size will give the best solution in terms of data structure efficiency. However, the hardware SMs will be seriously underutilized. Being limited by a maximum of 8 blocks for SM, we will be able to run only 256 threads (8 warps) simultaneously which represents only $\frac{1}{6}$ of the SM capability. Here we propose the use of warp-grained slices by decoupling slice size (set to $s = warp$) from CUDA block (set to $b = 256$). Typically, each thread (corresponding to a row in ELL representation) can explicitly calculate its warp index (which also translates to the slice number). This allows us to decouple the slice size and the block size, achieving both data structure efficiency and full SM occupancy. This fine-tuned sliced ELL format has a warp-level lockstep execution determined by the longest row. Moreover, the format can drastically reduce the memory footprint compared to the original ELL format, as clearly illustrated in Figure 4.

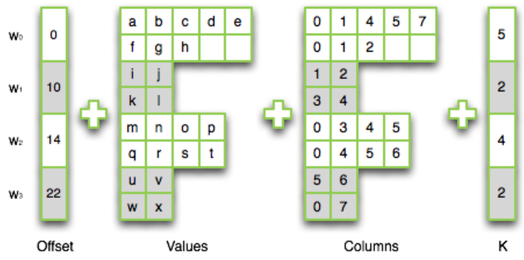


Figure 4. Sliced ELL with warp granularity

The efficiency of the warp-grained ELL can be further improved by row reordering. This technique is potentially able to reduce the variability of nonzeros per row within warps. A global row reordering (equivalent to pJDS [20]) can be performed in linear time $O(n)$ using bucket sort. However, this approach may shuffle data-unrelated rows close together worsening the overall data locality. Hence, we propose a reordering strategy based on local rearrangement. The idea is to decrease the variability without moving related rows too far apart. Assuming that the block size is larger than warp granularity, we order rows within the block to minimize variability of the corresponding warp-grained slices. Last, we can still combine the warp-grained ELL format with the DIA format by separately storing the main diagonal, obtaining a data structure well-suited for the Jacobi iteration.

VII. Experimental results

In this section we tested the proposed structure-aware optimizations, achieving substantial and consistent improvements for the CME steady-state calculation, outperforming a multicore implementation based on the state-of-the-art. We also evaluated the warp-grained ELL format more in general.

A. Hardware and software setup

All the experiments were performed on a NVIDIA GTX580 GPU equipped with 3GB of GDDR5 and a total of 512 CUDA cores. The computing platform was a quad-socket system equipped with four 16-cores AMD Opteron

	Linear system size			Nonzeros per row						Diagonal	
			[MB]								
Biological network	n	nnz	Disk	min	μ	max	σ	σ/μ	$\frac{max-\mu}{\mu}$	d{0}	d{-1,0,+1}
toggle-switch-1	319204	1908834	34.46	3	5.98	7	0.72	0.12	0.17	1.00	0.86
brusselator	501500	2501500	47.69	2	4.99	5	0.13	0.03	0.00	1.00	1.00
phage-lambda-1	1067713	10058061	202.60	2	9.42	15	2.78	0.30	0.59	1.00	0.70
schnakenberg	2003001	14001003	289.36	2	6.99	7	0.15	0.02	0.00	1.00	1.00
phage-lambda-2	2437455	14001003	529.15	3	10.65	15	1.63	0.15	0.41	1.00	0.98
toggle-switch-2	4425151	42202701	788.40	3	9.54	11	1.06	0.11	0.15	1.00	1.00
phage-lambda-3	9980913	94469061	2088.07	2	9.47	15	2.77	0.29	0.59	1.00	0.97

Table I
SPARSE LINEAR SYSTEMS FROM SAMPLE BIOLOGICAL NETWORKS

6274 and with 128 GB of DDR3. The operating system was 64-bit CentOS 6.3 with kernel 2.6.32. The compilers used in the implementation were gcc 4.4.6 and CUDA 4.2 (GPU device driver 295.41).

B. Benchmarks description

For our tests, we generated a set of reaction rate matrices from four different biological models by the means of the optimal enumeration algorithm [17]. We chose the following biochemical reaction networks: toggle switch [16], Brusselator [21], phage lambda switch [22] and Schnakenberg [23].

Table I presents some basic information about the generated reaction rate matrices. The first part of the table describes the linear system size in terms of microstates n, number of nonzeros nnz and disk memory necessary to store the sparse matrix using the Matrix Market [24] coordinate format. The second part of the table describes the matrices in terms of nonzeros per row (minimum, average μ, maximum and standard deviation σ). We also calculate two derived metrics. σ/μ represents a variability factor whereas $\frac{max-\mu}{\mu}$ represents a skew factor. 4 benchmarks out of 7 have low variation and skew, meaning that the ELL format is well suited. On the other hand, the remaining benchmarks provide an opportunity to achieve a better performance by using the warp-grained ELL format. The last part of the table analyzes the diagonal structure, showing the density of the main diagonal without and with its neighbors (where a density greater than 0.66 indicates a structure to leverage).

C. Sparse matrix-vector multiplication

The operations performed by the Jacobi iteration differ from SpMV by only a division. For convenience, we have chosen to collect some preliminary data about the proposed optimization techniques by using the ELL multiplication kernel. For all the tests, we measured the double-precision floating-point performance considering the average over 100 repetitions. We neglected the time needed to transfer the sparse matrix to the GPU global memory, considering the reasonable assumption of amortizing this one-time cost over several iterations during steady-state calculation. First, we identified the best block size b by exhaustive testing on the benchmarks. As pointed out in Section III, the best performance is achieved for $b = 256$ (a tradeoff between occupancy and block turnover). Second, we tested the actual effect of different L1-cache sizes obtaining a 6% improvement on the average performance (15.132 GFLOPS with 16KB versus 16.032 with 48KB). Hence, we fixed these optimal parameter values for the following experiments.

Table II evaluates the performance of the ELL+DIA format. The high diagonal density (≈ 0.97 on average) can be in fact exploited by this optimization technique. This provides an average performance improvement of about 1 GFLOPS (or 5%) and justifies the idea of leveraging the diagonal structure available in DFS-ordered reaction rate matrices. In general, any matrix with such structure can gain performance using the ELL+DIA format.

	ELL	ELL+DIA	
	[GFLOPS]	[GFLOPS]	
Biological network	Performance	Performance	Speedup
toggle-switch-1	17.652	17.844	1.01
brusselator	19.308	22.218	1.15
phage-lambda-1	11.602	11.956	1.03
schnakenberg	21.694	24.213	1.12
phage-lambda-2	11.375	11.463	1.01
toggle-switch-2	19.539	19.760	1.01
phage-lambda-3	11.056	11.352	1.03
Average	16.032	16.972	1.05

Table II
ELL VERSUS ELL+DIA

Before testing the warp-grained ELL format, we evaluate how the idea of local rearrangement affects data locality. We measured the average SpMV performance with different reordering. Specifically, we considered a random reordering (2.783 GFLOPS), a global nonzero reordering (15.137 GFLOPS) and a local nonzero rearrangement (16.278 GFLOPS). Not surprisingly, the global reordering decreases the performance (about -6%) since it probably shuffles unrelated rows close together. On the other hand, the local rearrangement has a slightly positive effect. Table III evaluates the performance of the warp-grained ELL format. As we can see, we are able to obtain a consistent average improvement of about 1.3 GFLOPS (or 8%) over the ELL format. We can also observe a 6% improvement over the original sliced ELL format due to the ideas of warp granularity and local rearrangement. We can conjecture that the improvement would have been even greater in case of benchmarks with more nonzero variability (in fact, 4 benchmarks out of 7 have a pretty regular structure and, hence, no margin for improvement). Table III also proposes a comparison with clSpMV [15], which can be considered the state-of-the-art for SpMV. This framework basically represents an ensemble of many available GPU sparse formats (precisely DIA, BDIA, ELL, SELL, CSR, COO, BELL, SBELL and BCSR) and is capable of selecting the best representation (or a combination of them) of any sparse matrix. Due to the intrinsic regularity of reaction rate matrices we avoided an unfair comparison with formats like SCOO [25], which is explicitly designed for unstructured matrices. Moreover, the available clSpMV implementation does not provide double-precision. However, we tried to draw a fair comparison by normalizing the results (e.g. if clSpMV selects single-precision ELL format, we normalize by $\frac{8}{12} = 0.66$). We can observe a substantial 24.41% improvement. Without going into the details, this result can be in part explained by the fact that clSpMV selects a non-intuitive mix of sparse formats (although the diagonal band is correctly identified in most of the cases).

Biological network	ELL [GFLOPS] Performance	Sliced ELL [GFLOPS] Performance	Warped ELL [GFLOPS] Performance	clSpMV [GFLOPS] Performance
toggle-switch-1	17.652	17.711	18.731	17.853
brusselator	19.308	19.156	18.859	16.399
phage-lambda-1	11.602	12.355	15.103	9.434
schnakenberg	21.694	21.694	24.213	20.203
phage-lambda-2	11.375	11.485	11.973	8.861
toggle-switch-2	19.539	20.294	20.627	17.717
phage-lambda-3	11.056	11.805	14.511	—
Average	16.032	16.346	17.320	15.078

Table III
ELL VS SLICED ELL VS WARP-GRAINED ELL VS CLSPMV

We also evaluated the memory footprint of the warp-grained ELL format. As mentioned, this aspect is very important for the practical feasibility since current GPU devices have limited global memory (if compared with conventional CPU systems). We observed that the warp-grained ELL format has an average footprint of 322.45 MBytes, much less than the 440.98 MBytes needed by the ELL format and slightly better than CSR format (which needs 323.71 MBytes). Finally, we performed a more general test on the warp-grained ELL format using several matrices taken from the University of Florida sparse matrix collection [26]. The aim was to capture the improvement over the original sliced ELL format. Figure 5 shows the comparison between these two formats. Each column represents the average performance over a set of matrices arising from a specific domain (we will make available a detailed list of the benchmarks in a future version of this manuscript). Similarly to what observed for molecular networks, the warp-grained ELL format has an edge over all the matrix domains. The average performance improvement is 12.62% and reaches a maximum of 48.09% for the quantum chemistry domain. This result suggests that the warp-grained ELL format can successfully replace the original sliced ELL format.

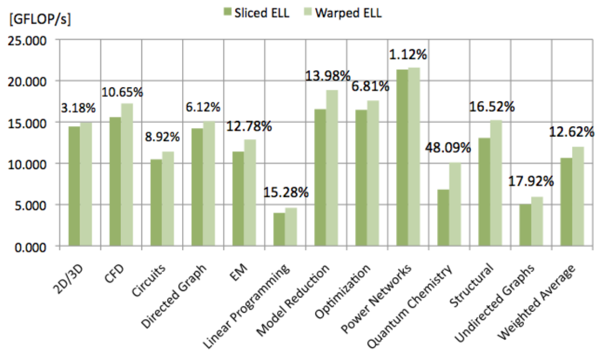


Figure 5. Sliced ELL versus Warp-grained sliced ELL

D. Jacobi iteration

The Jacobi iteration was implemented by adding the DIA component to each format previously presented. This approach has two advantages. First, it allows to directly read the diagonal element $a_{i,i}$. Second, it permits to exploit the dense diagonal structure $\{-1, +1\}$. In order to have a fair comparison, we took as baseline a multicore implementation derived from the Intel MKL library [14] (in practice CSR+DIA). We then used the Jacobi iteration to build a sparse linear solver and calculate the steady-state probability landscape of the given benchmarks. We set the error value to $\epsilon = 10^{-8}$ and the maximum number of iterations to 10^6. With these parameters, we were able to reach the stopping criterion for all the problems but one (phage-lambda-2). The obtained results are shown in Table IV. As we can see, the most sophisticate implementation achieves a performance 14.212 GFLOPS, outperforming the optimized multicore implementation by a 15.67x factor. This is definitely a good and concrete result that supports the practical use of the CME framework in system biology research.

Biological network	Iterations	Residual	CRS+DIA [GFLOPS] Performance	Warp ELL+DIA [GFLOPS] Performance
toggle-switch-1	36800	2.625e-06	1.399	15.479
brusselator	125800	1.331e-06	1.170	17.218
phage-lambda-1	453200	9.713e-06	0.730	10.323
schnakenberg	18300	2.536e-07	0.757	20.119
phage-lambda-2	1000000	9.025e-07	0.865	8.133
toggle-switch-2	21400	1.313e-05	0.783	17.772
phage-lambda-3	210600	1.288e-06	0.646	10.438
Average			0.907	14.212
Speedup				15.67x

Table IV
LINEAR SOLVER BASED ON JACOBI ITERATION

The new GPU architecture Kepler [27] introduces new architectural features. Most notably, the available computational resources (now known as SMXs) can be kept busy using the ability of launching simultaneous kernels from multiple CPU cores with no dependencies (Hyper-Q) or from the GPU itself (Dynamic Parallelism). However, a large matrix size combined with the iterative nature of the Jacobi solver already provides an efficient use of the SMs. In terms of double precision performance, Kepler assures a increased peak of 1.31 TFLOPS (one third of single precision) but this improvement is not fundamental for sparse linear algebra. In fact, we can expect more benefits from an improved memory hierarchy (more bandwidth at each level) and from a dedicated read-only 48KB data cache (well-suited for caching the dense vector x).

VIII. Conclusions

The Chemical Master Equation is a stochastic and discrete-state continuos-time formulation that provides a fundamental framework to model biochemical reaction networks inside the cells. In this paper, we proposed an efficient GPU-based Jacobi iteration for steady-state probability calculation. We provided several optimization strategies based on the problem structure with the aim of making this theoretical framework feasible, outperforming the conventional multicore implementation by a 15.67x factor.

We also devised a novel sparse format based on warp granularity and local rearrangement that achieves a substantial 24.41% improvement over the state-of-the-art for the specific domain (and a more general 12.62% improvement over the original sliced ELL format). We should point out that a large body of literature has been dedicated to SpMV optimization on GPU. As a results, it is not trivial to propose original ideas and it is very unlikely to achieve an impressive speedup over the state-of-the-art. However, we believe that our work has achieved, as a secondary contribution, a very reasonable result in terms general SpMV optimization.

This work provides the foundation and an approach for applying the CME framework to realistic biochemical systems. In addition, our GPU-based steady-state computation can be generalized to operation on stochastic matrices (Markov models), achieving good performance with matrix structures similar to biological reaction networks. We plan to extend our approach in order to overcome the current limitation in terms of GPU memory by moving to GPU clusters. Moreover, we plan to further develop our GPU-based CME stochastic framework by including transient dynamic calculation.

Acknowledgment

This work was supported by NSF grants IIS-106468 and DBI-1062328, and NIH grant GM-079804.

References

[1] D. A. Beard and H. Qian, *Chemical Biophysics: Quantitative Analysis of Cellular Systems*. Cambrige University Press, 2008.

[2] J. Liang and H. Qian, "Computational cellular dynamics based on the Chemical Master Equation: A challenge for understanding complexity," *Journal of Computer Science and Technology*, vol. 25, no. 1, pp. 154–168, 2010.

[3] Nvidia, "Nvidia's next generation cuda compute architecture: Fermi," *http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf*.

[4] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," *Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[5] A. Monakov, A. Lokhmotov, and A. Avetisyan, "Automatically tuning sparse matrix-vector multiplication for GPU architectures," *High Performance Embedded Architectures and Compilers*, vol. 5952, pp. 111–125, 2010.

[6] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," *Symposium on Principles and Practice of Parallel Programming*, vol. 45, no. 5, pp. 115–126, May 2010.

[7] F. Vázquez, J. J. Fernández, and E. M. Garzón, "Automatic tuning of the sparse matrix vector product on GPUs based on the ellr-t approach," *Parallel Computing*, August 2011.

[8] J. C. Pichel, F. F. Rivera, M. Fernández, and A. Rodríguez, "Optimization of sparse matrix–vector multiplication using reordering techniques on GPUs," *Microprocessors and Microsystems*, vol. 36, no. 2, pp. 65–77, March 2011.

[9] Y. Saad, *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[10] X. Yang, S. Parthasarathy, and P. Sadayappan, "Fast sparse matrix-vector multiplication on GPUs: Implications for graph mining," *International Conference on Very Large Data Bases*, vol. 4, no. 4, pp. 231–242, January 2011.

[11] B. R. C. Magalhaes, N. J. Dingle, and W. J. Knottenbelt, "GPU-enabled steady-state solution of large markov models," *International Workshop on the Numerical Solution of Markov Chains*, pp. 63–66, September 2010.

[12] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," *High performance computing, networking, and storage conference*, pp. 10–16, 2007.

[13] B. Bylina, J. Bylina, and M. Karwacki, "Computational aspects of gpu-accelerated sparse matrix-vector multiplication for solving markov models," *Theoretical and Applied Informatics*, vol. 23, no. 2, pp. 127–145, 2011.

[14] Intel, "Math kernel library," *http://software.intel.com/en-us/articles/intel-mkl/*.

[15] B.-Y. Su and K. Keutzer, "clSpMV: A cross-platform OpenCL SpMV framework on GPUs," in *Proceedings of the international conference on Supercomputing*, ser. ICS '12, 2012.

[16] T. S. Gardner, C. R. Cantor, and J. J. Collins, "Construction of a genetic toggle switch in escherichia coli," *Nature*, vol. 403, no. 6767, pp. 339–342, January 2000.

[17] Y. Cao and J. Liang, "Optimal enumeration of state space of finitely buffered stochastic molecular networks and exact computation of steady state landscape probability," *BMC System Biology*, vol. 2, no. 30, March 2008.

[18] Nvidia, "Cuda, parallel programming made easy," *http://www.nvidia.com/object/cuda _home_new.html*.

[19] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal of Scientific Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[20] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, A. Basermann, and A. R. Bishop, "Sparse matrix-vector multiplication on GPGPU clusters : A new storage format and a scalable implementation," *CoRR*, 2011.

[21] G. Nicolis and I. Prigogine, *Self-Organization in Nonequilibrium Systems*. John Wiley & Sons, 1977.

[22] Y. Cao, H.-M. Lu, and J. Liang, "Probability landscape of heritable and robust epigenetic state of lysogeny in phage lambda," *PNAS*, vol. 107, no. 43, pp. 18 445–18 450, October 2010.

[23] Y. Cao and J. Liang, "Nonlinear langevin model with product stochasticity for biological networks : the case of the schnakenberg model," *Journal of Systems Science and Complexity*, vol. 23, no. 5, pp. 896–905, October 2010.

[24] NIST, "Matrix market format," *http://math.nist.gov/MatrixMarket/*.

[25] H.-V. Dang and B. Schmidt, "The sliced coo format for sparse matrix-vector multiplication on cuda-enabled gpus," in *International Conference on Computational Science*, 2012.

[26] T. Davis, "University of florida sparse matrix collection," *http://www.cise.ufl.edu/research/sparse/matrices/*.

[27] Nvidia, "Nvidia's next generation cuda compute architecture: Kepler gk110," *http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf*.