Computational Methods for the Parallel 3D Simulation of Biochemical Kinetics at the Microscopic Scale

Laurent Crépin, Fabrice Harrouet, Sébastien Kerdelo, Jacques Tisseau, Pascal Redou

Computational Methods for the Parallel 3D Simulation of Biochemical Kinetics at the Microscopic Scale

Laurent Crépin[1,2], Fabrice Harrouet[1], Sébastien Kerdélo[1,2], Jacques Tisseau[1], and Pascal Redou[1]

[1] Lab-STICC, UMR 6285 CNRS, UEB/ENIB/CERV, France
[2] Diagnostica Stago, Gennevilliers, France
crepin@enib.fr

Abstract. This work takes place in the context of biochemical kinetics simulation for the understanding of complex biological systems such as hæmostasis. The classical approach, based on the numerical solving of differential systems, cannot satisfactorily handle local geometrical constraints, such as membrane binding events. To address this problem, we propose a particle-based system in which each molecular species is represented by a three-dimensional entity which diffuses and may undergo reactions. Such a system can be computationaly intensive, since a small time step and a very large number of entities are required to get significant results. Therefore, we propose a model that is suitable for parallel computing and that can especially take advantage of recent multicore and multiprocessor architectures. We present our particle-based system, detail the behaviour of our entities, and describe our parallel computing algorithms. Comparisons between simulations and theoretical results are exposed, as well as a performance evaluation of our algorithms.

Keywords: Particle-based system, parallel computing, biochemical kinetics, microscopic scale simulation.

1 Introduction

A common way to improve the understanding of complex biological systems is to run experiments on models in order to test hypotheses. The results are then extrapolated from the models to the real systems they represent. One usually distinguishes *in vivo* experimentation, taking place in a living organism, from *in vitro* experimentation, run in test tubes. One challenge for the upcoming years would be to save the various expenses of some of these experiments by running reliable computer simulations of predictive numerical models.

Complex biological systems necessarily involve the occurence of biochemical events such as chemical reactions; numerical simulations of these complex systems thus imply the modelisation of their kinetics. Biochemical kinetics can be simulated in many ways, but this is originally done by implementing empirical

laws, e.g. *mass-action* or *Henri-Michaelis-Menten* [1], into a set of ordinary differential equations that is solved using a numerical method [2]. The presence of biological material, such as membranes, can induce heterogeneity in the medium where the biochemical events occur. Partial differential equations are thereby required to take into account this spatial heterogeneity. The differential equation approach relies on the hypothesis that the medium is large enough to provide significant results. When it comes to small volumes, e.g. inside a biological cell, this assumption fails, which makes this approach only suitable for macroscopic scale simulations. Gillespie [3, 4] thus introduced *stochastic simulation algorithms* to mimic accurately the behaviour of the solution of the *chemical master equation* that describes biochemical kinetics at the mesoscopic scale. This approach takes into account the discrete and stochastic aspects of biochemical reactions [5] and makes the numerical simulation of biochemical kinetics possible, whatever the volume. The case of heterogeneous media has also been treated [6].

It is now widely accepted that, in addition to the discrete and stochastic aspects, spatial constraints must also be taken into account in order to simulate accurately biochemical kinetics [7]. As an example, the blood coagulation cascade [8] involves enzymatic reactions that take place both in solution and on a membrane surface. Moreover, as the clotting process goes on, a fibrin mesh is formed and the medium becomes insoluble: classical enzyme kinetics no longer applies. Simulations of such complex phenomena are typically achieved using *particle-based methods* that intend to describe biochemical kinetics at the microscopic scale [9]. These methods track individual molecules, named particles or entities, as they diffuse in three dimensions, collide and react. The main drawbacks with particle-based methods stand in the small time steps and the very large number of particles required to obtain significant results, which makes the simulations computationaly intensive. Although other particle-based simulations of biochemical kinetics at the microscopic scale already exist [10–13], none of them, to our knowledge, focused on a noteworthy gain of computational performances.

This work addresses this issue using a particle-based method for the simulation of biochemical kinetics at the microscopic scale that is suitable for parallel computing and that can especially take advantage of recent multicore and multiprocessor architectures. We present our model in two steps. First, we introduce our particle-based system and detail the behaviour of our entities, as well as their specific scheduling scheme. Then, we describe the cache-aware simulation engine and the parallel computing algorithms that we developed for performance purpose. This description is followed by a validation section in which we illustrate our approach on both a reversible and an enzymatic reactions and compare the results of our method with those obtained with the classical approach. As our work intends to improve the performances of the simulations, we then show the computational gain that our algorithms offer. Finally, we discuss our choices and give some perspectives for our work.

2 Model

2.1 Particle-Based System

Our method uses a particle-based approach, illustrated on Fig. 1, to simulate the kinetics of biochemical systems with spatial and stochastic details [14]. Two kinds of biochemical species can be represented: the `Species3D` entities which diffuse in solution (in a `Volume`) and the `Species2D` entities which diffuse along a physiological membrane (on a `Surface`). Each molecular species is represented by its geometrical shape, i.e. an ellipsoid for the 3D species in solution and a disc for the 2D species bound to a membrane.

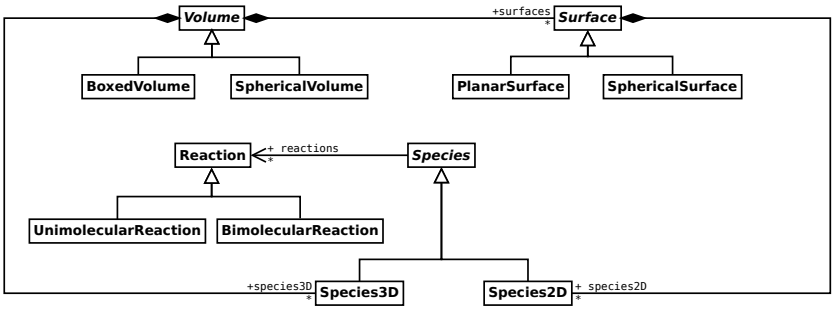

Fig. 1. UML class diagram of our model – The `Species` class represents an entity diffusing in a `Volume` or on a `Surface`, and which may undergo a `Reaction`.

Each entity diffuses in the reactional volume according to Brownian motion, with a diffusion coefficient computed from the entity's radii, the volume temperature and viscosity. It can also undergo two main biochemical reactions: unimolecular ones or bimolecular ones. These reactions are responsible for the creation or the destruction of other entities and, as a result, they govern the variations of the chemical concentrations in the system. The whole life cycle of an entity is detailed in Fig. 2 algorithm. The following sections describe each step of this algorithm.

Unimolecular Reactions. Unimolecular reactions are phenomena which can transform a biochemical species (the reagent) into one or more products. A reaction R converting a molecular species C in a couple A and B is represented by the scheme:

$$\mathrm{R} : \mathrm{C} \xrightarrow{k} \mathrm{A} + \mathrm{B} \tag{1}$$

where k is the reaction rate characterising the velocity of the phenomenon.

A molecular species can take part in one or many reactions. In order to simulate a system of n reactions, it is necessary to compute the probability of

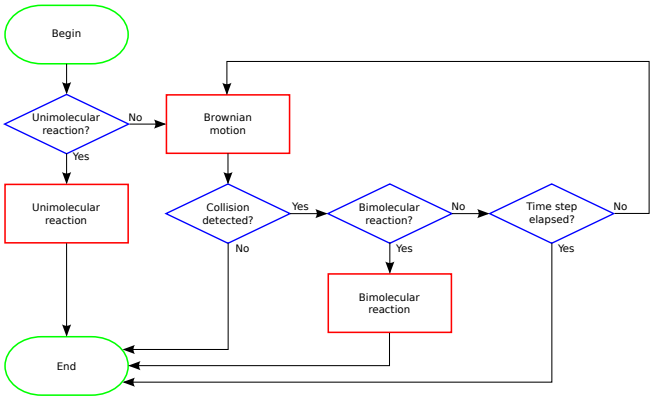

Fig. 2. Life cycle of an entity – This diagram summarises the global behaviour of an entity during one simulation step. First, the entity checks if an unimolecular reaction should happen. If not, the entity diffuses in the environment according to Brownian motion. Then, if a collision happens, the entity tries to react with the collided one and undergoes a bimolecular reaction if necessary. This process is repeated until the time step allocated to the entity is elapsed.

each one, i.e. the probability $P(R_i)$ of the event "the reaction R_i occurs on a given time step Δt" and the probability $P(\overline{R})$ of the event "no reaction occurs". These probabilities, also detailed in [15], are given by:

$$\begin{cases} P(R_i) = \dfrac{k_i}{\sum_{j=1}^{n} k_j} \times \left(1 - \exp(-\sum_{j=1}^{n} k_j \times \Delta t)\right) \\ P(\overline{R}) = \exp(-\sum_{j=1}^{n} k_j \times \Delta t) \ . \end{cases} \tag{2}$$

This system determines which reaction will occur during the current time step Δt thanks to uniform random sampling.

Brownian Motion and Collision Detection. Brownian motion characterises the erratic motion of an entity within a fluid. It arises from the collisions between the solvent particles and the entity itself which seems to move randomly under these impacts [16]. The theory associated with the brownian motion of ellipsoidal molecules is studied in [17, 18]. At the microscopic scale, this phenomenon is usually modeled using three-dimensional random walks, i.e. Markov processes defined by a succession of random elementary steps. The future position and orientation of the entity solely depend on its current location, which means the process is memoryless. This succession of elementary steps can be approximated by random Gaussian sampling. As a matter of fact, during a time step Δt, each three-dimensional translation and rotation are computed from six random

variables following a Gaussian law with a zero mean and a standard deviation of $\sqrt{2D\Delta t}$, where D is the diffusion coefficient of the entity.

During its diffusion step, an entity may encounter one of its close neighbours. To determine if a collision occured between two molecular species, we use the collision detection algorithms for ellipsoids presented in [19]. Then we use a dichotomy algorithm to manage the collision and avoid the overlap of the entities. Such collisions can lead to bimolecular reactions.

Bimolecular Reactions. As opposed to unimolecular ones, bimolecular reactions can only occur when collisions happen in the system. As a result, two reagents are necessary to create a product. A reaction R between two species A and B, producing C, can be represented as follows:

$$\mathrm{R} : \mathrm{A} + \mathrm{B} \xrightarrow{k} \mathrm{C} \tag{3}$$

where k is the reaction rate. To simulate such reactions, the probability $P(R)$ that a reaction occurs upon a collision between the species A and B has to be determined. This probability is maximal, i.e. $P(R) = 1$, when all the collisions between the reagents lead to the creation of a product. The corresponding maximum rate constant $k_{\max}$ can easily be computed by using our simulator and by fitting the concentrations with the solution of the mass action law equation. It is worth noting that the values we obtain for different simulations are rather consistent with Von Smoluchowski theory [1] which states that the maximum rate constant is $k_{\max} = 4\pi(D_{\mathrm{A}} + D_{\mathrm{B}})(r_{\mathrm{A}} + r_{\mathrm{B}})N_{\mathrm{A}}$, where D_{A} and r_{A} are respectively the diffusion coefficient and the radius of the species A (likewise for B), and N_{A} is the Avogadro number. Once $k_{\max}$ has been determined, the probability $P(R)$ is given by the ratio $\frac{k}{k_{\max}}$ where k is the rate constant that we want to simulate. A reaction R occurs only if the value of a random variable X, following a standard uniform distribution, is less than $P(R)$.

However, if two entities do not react the first time they collide, there is a very high probability that they will during the next simulation steps. This is due to the fact that the entities are still very close to each other after their first encounter. To avoid the bias introduced by this recollision problem, we decide to consider only the first collision in a sequence of encounters.

Scheduling. The interactions between our entities are not predetermined and the overall behaviour of the system is unknown. Because these entities are not just numerical equations which results could be added, as in a synchronous system, they introduce concurrency in the simulation. When two entities collide and react in one place, a third one cannot pretend having reacted with one of them (which could have just disappeared) in another place during the same time step; the state of the system has been irreversibly changed by the preceding reaction and all the following actions have to consider this new state. Therefore we chose to use an asynchronous and chaotic iteration scheme to schedule the

entities [20]. Although a common time step ensures the temporal consistency of the entities, the asynchronous scheduler executes every entity one after another inside this time step in order to take into account every single event. However, since every entity is affected by the previous ones actions, a fixed scheduling order would have implied an unwanted priority between them. As shown on Fig. 3, the chaotic scheduler gets rid of this artifact with a random reordering at each new step. The convergence and stability of such a scheduling scheme in the context of differential system solving were strictly validated in [21].

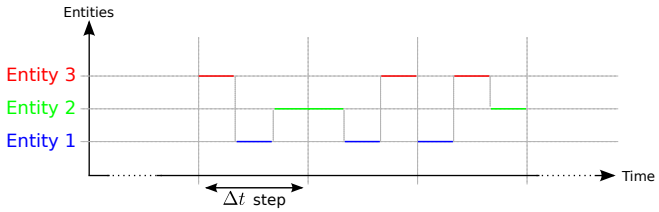

Fig. 3. Asynchronous and chaotic scheduling – This figure shows the execution order of three entities scheduled according to the asynchronous and chaotic iteration scheme. Three simulation steps are represented. The scheduler executes every entity exactly once in a time step, randomly reordering the sequence at each new step.

2.2 Parallel Asynchronous Scheduler

According to the law of large numbers, a particle-based biochemical kinetics simulation requires many entities (approximately 10^5) to be significant. Moreover, the microscopic scale implies the use of very short time steps (approximately 10 nanoseconds) and therefore, many iterations are required to compute the whole simulation. To speed up such intensive computations, we developed algorithms suitable for parallel computing on multicore and multiprocessor architectures.

Background: Cache-Aware Simulation Engine. As detailed in [22], we previously designed a simulation engine that can harness the full potential of all the Central Processing Units (CPUs) (would they be processors, physical cores, or logical cores) in a parallel computer.

To prevent cache-memory trashing, the whole set of entities to schedule is subdivided in as many subsets as there are CPUs. Since our simulations imply a common repetitive time step, some work-stealing [23] is used at the end of each step to dynamically balance the CPU workloads, keeping all of them busy until the end. This stealing relies on the knowledge of the cache-memory hierarchy to minimise trashing. The assignment of every entity to the CPU in charge of its execution is arbitrary at first but is dynamically adjusted: each executed entity keeps track of its neighbours' current CPU and moves to the most represented

one for the next simulation step. It is then very likely for an entity and its neighbours to be run on the same CPU, and thus to find their respective data already up to date in the same cache-memory. This solution is more generic than a spatial partitioning and offers a better load balancing when it comes to heterogeneous spatial distributions or gregarious behaviours.

This cache-aware simulation engine shows a very good scalability related to the number of CPUs used [22]. However, it was formerly dedicated to synchronous simulations and, as stated in section 2.1, our particle-based biochemical model relies on an asynchronous scheduling scheme.

From Synchronous to Asynchronous Parallel Scheduling. The first issue we have to deal with consists in keeping the consistency of any entity when accessed by many CPUs simultaneously. Each entity has its own reader-writer-lock [23] which is locked for writing (one at a time) when its behaviour is executed. When an entity collects informations from its neighbours, it locks them for reading (many reader-lock operations are allowed at the same time on a given lock). In case it needs to modify a neighbour, the reader-lock is promoted to writer-lock (one at a time). Even though the consistency is now guaranteed for concurrent accesses, the main drawback with overlapped lock operations stands in deadlock situations: several entities having to lock themselves and one another, thus waiting endlessly for these locks to become free.

To prevent this new issue from happening, we turn the locking operations into attempts that may immediately fail if the lock is not free. This requires that the behaviour of the entities has to be written so that all the decision making takes place in local variables; the entities are finally modified only if all the chain of locking operations succeeds. When a locking failure occurs, the currently executed behaviour is simply given up as if it has never been started; it will be rescheduled later in the same time step. Nevertheless, when approaching the end of the step, a live-lock situation may occur: the rescheduled entities which interact with one another on different CPUs are probably the only remaining ones and will forever miss their locking attempts.

To get around this situation, as soon as an entity fails twice to be scheduled in the same time step, we postpone its execution to the next step. As the scheduling follows a random order, it is very unlikely for these concurrent entities, amongst many other ones now, to be scheduled simultaneously one more time. Of course, a postponed entity has to be scheduled twice in this next step to ensure a long term temporal consistency between entities.

Although these locking attempts, rescheduling and postponing decisions may seem to raise the computation workload, they actually do not so much. Since our scheduler tends to assign the entities to the CPUs according to their neighbourhood, most of the time a whole set of interacting entities is scheduled by only one CPU. Consequently, the locking attempts are serialised and mostly succeed, thus it is scarcely ever necessary to reschedule or even postpone some entities.

3 Results

This section presents some results which validate our algorithms on simple simulations. We achieve this validation by comparing the simulation results obtained with our approach with the ones determined by solving the differential equations of the mass action law. Then, we study how simulation speed scales related to the number of CPUs involved.

3.1 Validation

We chose to illustrate our approach on a reversible reaction and an enzymatic reaction as they are the most frequent in biochemical kinetics. These reactions are modeled by one bimolecular reaction and one or two unimolecular reactions. As an example for the simulation of a reversible reaction, our approach is illustrated on the interaction of blood coagulation factor Xa with its tight-binding inhibitor, tissue factor pathway inhibitor (TFPI). The whole biochemical description of this interaction can especially be found in [24]. Then, we illustrate our method on the activation of prothrombin (II) into thrombin (IIa) by an enzymatic complex (PT_{Ptex}) that can be found in the venom of the Australian snake *Pseudonaja textilis*. This interaction is fully detailed in [25]. Table 1 gathers all the data we used to set up our validation simulations.

Table 1. Validation parameters – This table presents the validation conditions of our model. It gathers every data necessary to reproduce our results. It should be noted that the shape of factors Xa and II are assumed to be prolate ellipsoids whereas the other species have a spherical shape.

<table>
<tr><th colspan="3">Reversible reaction</th><th colspan="4">Enzymatic reaction</th></tr>
<tr><td colspan="3">$\mathrm{Xa} + \mathrm{TFPI} \underset{k_{\mathrm{off}}}{\overset{k_{\mathrm{on}}}{\leftrightarrows}} \mathrm{Xa}\cdot\mathrm{TFPI}$</td><td colspan="4">$\mathrm{PT_{Ptex}} + \mathrm{II} \underset{}{\overset{K_{\mathrm{M}}}{\leftrightarrows}} \mathrm{PT_{Ptex}}\cdot\mathrm{II} \overset{k_{\mathrm{cat}}}{\rightarrow} \mathrm{PT_{Ptex}} + \mathrm{IIa}$</td></tr>
<tr><td colspan="3">$k_{\mathrm{on}} = 0.9 \times 10^{6}\,\mathrm{M^{-1}\cdot s^{-1}}$
$k_{\mathrm{off}} = 3.6 \times 10^{-4}\,\mathrm{s^{-1}}$</td><td colspan="4">$K_{\mathrm{M}} = 1.83 \times 10^{-6}\,\mathrm{M}$
$k_{\mathrm{cat}} = 5.87\,\mathrm{s^{-1}}$</td></tr>
<tr><th colspan="7">Initial concentrations</th></tr>
<tr><td colspan="3">$[\mathrm{Xa}]_0 = 170\,\mathrm{nM}$
$[\mathrm{TFPI}]_0 = 2.5\,\mathrm{nM}$</td><td colspan="4">$[\mathrm{PT_{Ptex}}]_0 = 100\,\mathrm{nM}$
$[\mathrm{II}]_0 = 1.4\,\mu\mathrm{M}$</td></tr>
<tr><th colspan="7">Dimensions</th></tr>
<tr><td>Xa</td><td>TFPI</td><td>Xa·TFPI</td><td>$\mathrm{PT_{Ptex}}$</td><td>II</td><td>$\mathrm{PT_{Ptex}}\cdot\mathrm{II}$</td><td>IIa</td></tr>
<tr><td>$r_{\mathrm{x}} = 26.0\,\text{Å}$
$r_{\mathrm{y}} = 26.0\,\text{Å}$
$r_{\mathrm{z}} = 51.5\,\text{Å}$</td><td>$r = 22.5\,\text{Å}$</td><td>$r = 35.9\,\text{Å}$</td><td>$r = 39.3\,\text{Å}$</td><td>$r_{\mathrm{x}} = 22.5\,\text{Å}$
$r_{\mathrm{y}} = 22.5\,\text{Å}$
$r_{\mathrm{z}} = 60.0\,\text{Å}$</td><td>$r = 45.0\,\text{Å}$</td><td>$r = 25.0\,\text{Å}$</td></tr>
</table>

The results computed from our simulations are illustrated on Fig. 4, and are consistent with the ones coming from reference laws: the mean relative errors at steady state are about 1% for both reversible and enzymatic reactions. These

two validations are essential before addressing more complex simulations. One may notice that we do not deal with membrane binding events in these examples. This point will be developed in the perspective section.

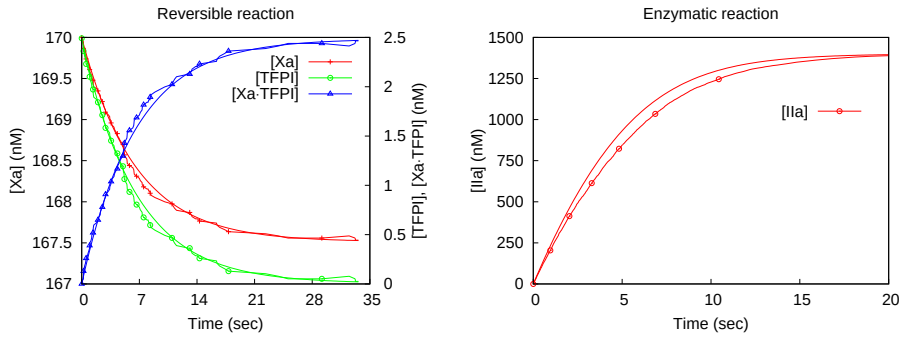

Fig. 4. Reversible and enzymatic reaction validation – These curves illustrate the validation of our model on two kinds of reaction. Smooth curves are computed from mass action law whereas noisy ones are the results of our simulations.

3.2 Performances

We ran many simulations, similar to the preceding ones, on a single computer with two Intel® Xeon® X5680 processors at 3.33 GHz clockspeed and a 12-Mbyte level-3 cache each. Thanks to the 2-way SMT technology, the twelve CPU cores provided here can be seen as twenty four logical CPUs running simultaneously. Figure 5 reports the computational frequency, i.e. the number of simulation steps per second, as well as the number of simulated entities running at a given rate, depending on the number of CPUs used. Both curves seem to scale linearly but with a slight change in the slopes around twelve CPUs. When using only the first dozen of CPUs, the physical cores fully exploit their respective hardware resources and cache hierarchy, but when it comes to the second dozen of CPUs, the 2-way SMT technology is involved and implies the sharing of the same resources for twice as much workload. This explains the slightly lower efficiency.

These results show that, as long as the CPUs come along with some cache memory, the simulation performances scale linearly with the number of CPUs involved. This lets us foresee that the cache-aware design of our simulator would enable even bigger simulations when using over twenty four CPUs.

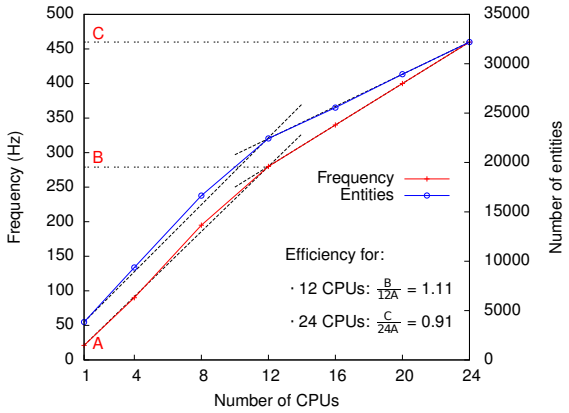

Fig. 5. Performance scaling – One curve illustrates the scaling of the computational frequency, depending on the number of CPUs used, for a simulation with 22438 entities. The other one shows the number of entities that our simulator can handle at a given computational frequency (280 Hz).

4 Discussions and Perspectives

4.1 Alternatives for Computing Power

Intensive computations can be handled by several technologies involving many computing units. Some of these parallel solutions are local, taking place in a single computer, such as multicore and multiprocessor computers or graphical processing units (GPUs), whereas others are distributed, implying communication through a network (dedicated or not), such as computer clusters, grid computing or cloud computing.

While distributed solutions are flexible and may provide a huge amount of computing units, the communication delays make them only suitable for computational problems which can be naturally subdivided into many independent subproblems requiring very little or even no synchronisation. Our simulations are made of multiple recurrent interactions which cannot be split in fixed and independent subsets beforehand; they require intensive synchronisations between the computing units. Because the hardware of general purpose parallel computers ensures very efficiently the cache-memory coherency, we decided to focus on this technology in the design of our simulation engine. Even in this favourable local context, our experiments showed that the slightest clumsiness in synchronisation and cache usage impacted significantly the overall computing performance; this definitely discouraged us from investigating distributed solutions.

GPUs are local to a computer and provide much more computing units than CPUs do. Our experiments highlighted that, not only the data transfer delay with the GPUs was far from being negligible compared to the computation duration, but also the programming model associated with this technology [26] was well-suited for a synchronous simulation scheme in which thousands of en-

tities can compute simultaneously an identical behaviour without taking care of any unexpected data change in their neighbourhood. Unfortunately, as stated in section 2.2, the asynchronous simulation scheme associated with our model implies a careful synchronisation strategy. Although some synchronisation primitives are available on GPUs, their usage totally contradicts the programming model of these devices and tends to ruin dramatically the raw computing performance they are capable of. Consequently, we prefer saving this technology for synchronous simulations of physical phenomena which could however interact with our particle-based asynchronous simulations.

4.2 Perspectives and Future Work

Due to the fact that we chose the microscopic scale, our diffusion algorithm requires a very short time step in order not to miss collisions (approximately 10 nanoseconds). This constraint somewhat limitates the performances of the simulations. As a matter of fact, two main parameters govern these performances: the number of entities executed during each simulation step and the length of this time step. Parallel computing enables us to increase the first parameter by using more computational units. However, it has no impact on the second one. We are currently working on a statistical method to address this problem. For the time being, we recommend the use of our simulator for short time applications, i.e. biochemical processes lasting no longer than ten seconds.

Beside this time step optimisation, a part of our model still needs to be thoroughly validated. As stated previously in section 2.1, the molecular species in solution may bind with membranes. Altough such bindings (and unbindings) are fully implemented, we still have to ensure the consistency of our results with experimental ones. In the present situation, our simulator provides nothing more than the classical approach. Nevertheless, this method is the only one capable of handling local geometrical constraints in biochemical systems. We will therefore focus our future work on modeling and validating these membrane binding events.

References

1. Purich, D., Allison, R.: Handbook of biochemical kinetics. Academic Press (2000)
2. Alves, R., Antunes, F., Salvador, A.: Tools for kinetic modeling of biochemical networks. Nature Biotechnology (6) (2006) 667–672
3. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics **22**(4) (1976) 403–434
4. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry **81**(25) (1977) 2340–2361
5. Van Kampen, N.: Stochastic processes in physics and chemistry. North Holland (March 2007)
6. Stundzia, A.B., Lumsden, C.J.: Stochastic simulation of coupled reaction-diffusion processes. Journal of Computational Physics **127**(1) (1996) 196–207

7. Resat, H., Costa, M.N., Shankaran, H.: Spatial aspects in biological system simulations. In Johnson, M.L., Brand, L., eds.: Computer Methods, Part C. Volume 487 of Methods in Enzymology. Academic Press (2011) 485–511
8. Marder, V.J., Aird, W.C., Bennett, J.S., Schulman, S., White, G.C.: Hemostasis and Thrombosis: Basic Principles and Clinical Practice. Lippincott Williams & Wilkins (November 2012)
9. Tolle, D.P., Le Novère, N.: Particle-based stochastic simulation in systems biology. Current Bioinformatics **1**(3) (August 2006) 315–320
10. Andrews, S.S., Addy, N.J., Brent, R., Arkin, A.P.: Detailed simulations of cell biology with smoldyn 2.1. PLoS Computational Biology **6**(3) (03 2010)
11. Stiles, J.R., Bartol, T.M.: 4. In: Monte Carlo Methods for Simulating Realistic Synaptic Microphysiology Using MCell. CRC Press (2001)
12. Plimpton, S.J.: Chemcell: a particle-based model of protein chemistry and diffusion in microbial cells. Technical report, Sandia National Laboratories (December 2003)
13. Van Zon, J., Ten Wolde, P.: Green's-function reaction dynamics: a particle-based approach for simulating biochemical networks in time and space. The Journal of Chemical. Physics. **123** (January 2005) 234910 1–16
14. Kerdélo, S.: Méthodes informatiques pour l'expérimentation in virtuo de la cinétique biochimique - Application à la coagulation du sang. PhD thesis, Université de Rennes 1 (January 2006)
15. Andrews, S.S., Bray, D.: Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. Physical Biology **1**(3) (2004) 137
16. Berg, H.C.: Random Walks in Biology. Princeton University Press (September 1993)
17. Perrin, F.: Mouvement brownien d'un ellipsoïde (i) : Dispersion diélectrique pour des molécules ellipsoidales. Journal de Physique et le Radium **5**(10) (1934) 497–511
18. Perrin, F.: Mouvement brownien d'un ellipsoïde (ii) : Rotation libre et dépolarisation des fluorescences. translation et diffusion des molécules ellipsoïdales. Journal de Physique et le Radium **7**(1) (1936) 1–11
19. Wang, W., Wang, J., Kim, M.S.: An algebraic condition for the separation of two ellipsoids. Computer Aided Geometric Design **18**(6) (2001) 531–539
20. Harrouet, F.: oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes. PhD thesis, Université de Bretagne Occidentale (December 2000)
21. Redou, P., Gaubert, L., Desmeulles, G., Béal, P.A., Le Gal, C., Rodin, V.: Absolute stability of chaotic asynchronous multi-interactions schemes for solving ode. CMES: Computer Modeling in Engineering & Sciences (December 2010)
22. Harrouet, F.: Designing a multicore and multiprocessor individual-based simulation engine. Micro, IEEE **32**(1) (January–February 2012) 54–65
23. Padua, D.A.: Encyclopedia of Parallel Computing. Springer (2011)
24. Baugh, R.J., Broze, G.J., Krishnaswamy, S.: Regulation of extrinsic pathway factor Xa formation by tissue factor pathway inhibitor. Journal of Biological Chemistry **273**(8) (1998) 4378–4386
25. Bos, M.H.A., Boltz, M., St. Pierre, L., Masci, P.P., de Jersey, J., Lavin, M.F., Camire, R.M.: Venom factor V from the common brown snake escapes hemostatic regulation through procoagulant adaptations. Blood **114**(3) (2009) 686–692
26. NVIDIA: NVIDIA CUDA C best practice. Technical report, NVIDIA (October 2012)