A Parallel Implementation of the Sτ-DPP Stochastic Simulator for the Modelling of Biological Systems

Ettore Mosca,Ivan Merelli, Luciano Milanesi
Institute for Biomedical Technologies
National Research Council of Italy
{*ettore.mosca,ivan.merelli, luciano.milanesi*}*@itb.cnr.it*

Andrea Clematis, Daniele D'Agostino
Institute for Applied Mathematics and Information Technologies
National Research Council of Italy
{*dago,clematis*}*@ge.imati.cnr.it*

***Abstract*—In the last decade, different computing paradigms and modelling frameworks for the description and simulation of biochemical systems based on stochastic modelling have been proposed. From a computational point of view, many simulations of the model are necessary to identify the behaviour of the system. The execution of thousands of simulation can require huge amount of time, therefore the parallelization of these algorithms is highly desirable. In this work we discuss the different strategies that can be implemented for the parallelization of a space aware τ-DPP variant, that is proving a C-MPI implementation of the system and discussing its performances according to the simulation of a particle diffusion in a crowded environment.**

I. Introduction

Membrane systems, also called P systems, are computing devices inspired to the structure and operation of living cells as well as from the way the cells are organized in tissues and higher order structures.

The properties of this class of systems make them suitable also for modelling biological systems [1], in which the different sets of objects represent molecular species and the rewriting rules represent chemical reactions that describe the evolution of the system in the time. However, some features of P systems as non-determinism and maximal parallelism have to be mitigated, while other properties, as physical-based procedure to describe the time evolution, have to be considered more carefully to ensure the accurateness of the results. Moreover, stochastic methods have gained a great attention since many biological processes are controlled by noisy mechanisms. This is particularly true when the molecular quantities involved are small, as in this case.

A membrane system variant which relies on these considerations is called τ-DPP [2], where Dynamical Probabilistic P systems have been coupled with a modified version of the τ leaping stochastic simulation method [3], in order to obtain a quantitative time streamline. A novel variant of τ-DPP, called Sτ-DPP [4], [5], has been introduced to consider the size of volumes and objects involved in a system, in order to better describe systems where the space plays an important role in the dynamics, such as crowded systems.

The algorithm can be used in the modelling and simulation of reaction-diffusion (RD) systems in crowded environments. RD systems are models used to describe those chemical systems for which the spatial distributions of chemicals influence the overall dynamics. The standard methods used to describe such systems are based on partial differential equations. However, when the intrinsic fluctuations of the chemical system play a major role in the dynamic, as in the case of many systems of interest for the biology, a stochastic approach is more suitable. The intracellular environment in RD systems is considered crowded since it is characterized by the presence of high concentrations of soluble and insoluble macromolecules: therefore, the classic approaches, which consider molecules as points without size, are no longer adequate. Under crowded conditions, in fact, the rate of some cellular processes can be increased or decreased, according to the free space in the system. By using Sτ-DPP, it is possible to increase the correctness of the description of a crowded system by computing the reactions probability according to the free space occurring in a volume.

In this paper we present the possible parallelization strategies for the Sτ-DPP Stochastic Simulator, in order to develop a tool able to efficiently process systems of arbitrary size. In particular we implemented a first C-MPI version of the algorithm, which shows the importance of a proper exploitation of the available computational capabilities.

II. The Sτ-DPP Algorithm

A Sτ-DPP system composed by n membranes is defined mainly by:

- $\Sigma = \{s_1, \ldots, s_m\}$, the set of molecular species;
- $R = \{R_1, \ldots, R_n\}$, where R_i is the set of internal and communication rules occurring inside the i^{th} membrane. An *internal* rule is of the form
$$\alpha_1 s_1 + \cdots + \alpha_m s_m \xrightarrow{c} \beta_1 s_1 + \ldots + \beta_m s_m$$
and a *communication* rule is of the form
$$\alpha_1 s_1 + \ldots + \alpha_m s_m \xrightarrow{c} (\beta_{1,1} s_1 + \ldots + \beta_{m,1} s_m, \text{in}_1) + \\ + \ldots + (\beta_{1,n} s_1 + \ldots + \beta_{m,n} s_m, \text{in}_n)$$
where the quantities α_i and β_j are natural numbers, c is a stochastic constant and $in_1, \ldots, in_n$ indicate the target membrane to which the resulting quantities of molecular species are sent.

- $W(t) = \{w_1(t), \ldots, w_n(t)\}$, where $w_i = \alpha_{1,i} \ldots \alpha_{m,i}$ is the multisets representing, at time t and for the i^{th} membrane, the amounts of each molecular specie;

The system evolves by applying the rules in a set of states $W(t_a) \ldots W(t_z)$. Adopting the notation used to represent chemical reactions, at the left side of the rule we have reactants, while at the right part we have products. Each object is preceded by a number (called stoichiometric coefficient), which indicates how many copies of that objects are involved in the rule. The rules are executed in a maximally parallel manner, non-deterministically choosing among those applicable. In other words, a rule is applicable if there is a sufficient amount of molecules, as specified in its left side, and if *the propensity function* value $a(x)dt$, that is the probability of one reaction to occur in the next infinitesimal time interval $[t, t + dt]$, given $x = W(t)$, is high.

The time evolution of such systems can be solved using approaches based on the use of ordinary differential equations (ODEs). However such approaches are both very computational intensive and, more important, do not take into account the noisy mechanism.

To overcome this limitation Gillespie proposed the *stochastic simulation algorithm* (SSA) [6] based on a Monte Carlo approach. The theoretical basis of the SSA is the function $p(\tau, j|x, t)$, that is defined so that $p(\tau, j|x, t)d\tau$ is the probability, given the state $W(t) = x$, that the next reaction in the system will occur in the infinitesimal time $[t + \tau, t + \tau + d\tau)$ and will be a reaction r_j.

The function $p(\tau, j|x, t)$ is therefore the joint probability density function of the two random variables τ (the time to the next reaction) and j (the next reaction), considering the system in the state x. By applying the laws of probability, and defining l the possible reactions, it is possible to derive the following analytical expression :

$$p(\tau, j|x, t) = a_j(x)e^{-a_0(x,\tau)}$$

where $a_0(x, \tau) = \sum_{k=1}^{l} a_k(x)$.

The main disadvantage of the SSA is its computational cost, which depends on the reaction and molecule numbers. As a consequence, the SSA is hardly useful to study complex biological pathways, which are characterized by many different types of molecules and reactions. The τ leaping method [3] represents a way to decrease this cost. The basic idea of this simulation technique is to fire more than one reaction events after a pre-selected time step τ. Moreover, this quantity must satisfy the *leap condition*: the change in the system's state due to the occurring of the selected reactions must be sufficiently small such that no propensity function will suffer an appreciable change in its value. The gain obtained at the computational level is paid with the introduction of an error on the systems dynamics that is no longer exact, as in the SSA, but approximated.

In case of many applicable rules, in fact, the rule to be applied is non-deterministically chosen among them. This may result in different behaviours due to the possible presence of the so-called "critical rules", that is reactions that involve at least one reactant, which is available in a little quantity and may prevent the application of some of the others. This aspect and the fact that τ is computed using random values results in the need to perform stochastic simulations in order to improve the quality of results.

The explicit consideration of reactants and membrane space occupation is a crucial feature for modelling real systems in which the crowding can have an important impact over the system's dynamics, i.e. the intracellular environment of living cells. The Sτ-DPP system introduces the concept of space occupation defining the free space F_i for the membrane $i - th$ at time t:

$$F_i(t) = v_i - \left(\sum_{j=1}^{m} (w_i(s_j, t) \cdot v_{s_j}) + \sum_{l=1}^{n} \alpha_l \cdot v_l \right)$$

where $\alpha_l \in \{0, 1\}$ takes the value 1 if the membrane labelled l is a son of membrane i in the membrane hierarchy, otherwise is 0, $w_i(s_j)$ denotes the number of occurrences of the symbol s_j in the multiset w_i, and v_x the volume of membranes and molecules. Hence, the free space of membrane i is simply defined as the difference between its volume and both (i) the sum of the volumes of the objects it contains and (ii) the sum of the volumes of all the membranes that are both nested in and connected to membrane i.

This add a further constraint in the evolution of the system: a rule is applicable if $F_i \geq 0$ for $1 \leq i \leq n$ calculated in the configuration potentially reached after its application considering both the internal and communication rules. The resulting algorithm is the following one.

```
1) load the description of the S$\tau$-DPP system;
2) for each membrane $i \in [1..n]$ calculate $F_i(t_0)$;
3) for each membrane $i \in [1..n]$
    a) $\forall$ rule $r_k$, $(k \in \{1, \ldots, l\})$: compute $a_k$;
    b) evaluate the sum of all the propensity functions
       $a_0$ in the compartment;
    c) IF $a_0 \neq 0$: $\tilde{\tau}_i = \infty$;
       ELSE generate the step size $\tilde{\tau}_i$ according to the
       internal state, and select the way to proceed in
       the current iteration (i.e. $\tau$ leaping evolution with
       or without critical reactions);
4) select $\tau_{min} = \tau_i = \min\{\tilde{\tau}_i, \ldots, \tilde{\tau}_n\}$;
5) for each membrane $i \in [1..n]$
    a) IF $\tilde{\tau}_i = \infty$ goto e) ;
    b) switch the evolution strategy type:
        • case "$\tau$ leaping with one critical reaction":
            if($\tilde{\tau}_i > \tau_i$): goto d) else: goto c);
        • else goto d);
    c) extract the critical and the non-critical rules that will be applied in the current iteration;
    d) IF the execution of the selected rules in all the volumes leads to an unfeasible state $\tau_{min} = \frac{\tau_i}{2}$;
6) IF a new value of $\tau_{min}$ was computed: $\tau_i = \tau_{min}$ and goto 4;
7) for each membrane $i \in [1..n]$
    a) update the internal state by applying the internal rules
    b) update the state of other membranes by applying the communication rules;
8) for each membrane $i \in [1..n]$ update the value of the free space $F_i$;
9) IF the termination criteria is satisfied, namely (i) the current time exceeds the end time OR (ii) there is not enough free space in any membrane: finish;
   ELSE: goto 3.
```

III. Related Works

The Stochastic Simulation Algorithm is a very important approach for the simulation of the behaviour of chemically reacting systems. However, in a large number of cases, its execution can be very time consuming [7].

It is worth noting that our approach presents a further synchronization step, due to the check on the free space, therefore in general it will present more accurate results but longer execution times with respect to other algorithms.

Considering the possible large number of runs to obtain high quality values for the probability density function of the molecular species required by the stochastic approach, parallel computation on multicore CPUs [8], clusters [9], GPUs [10] and also FPGA [11] were investigated. A common issue of these works is that they focuses on one architectural feature, while present systems are composed by a large number of heterogeneous cores, many of them specialized for different tasks, whose proper exploitation is still an issue.

IV. The Parallel Implementation

The Sτ-DPP Stochastic Simulator algorithm presents three possible levels of parallelization.

The simplest one is related to the stochastic nature of the simulation, that is many independent instances of Sτ-DPP system, from several dozens up to several thousands, can be executed in an embarrassingly parallel way, in order to achieve a statistically accurate result. This approach is typically adopted while dealing with distributed platforms, such as Grid computing [12], [13].

The second one is related to the execution of each single simulation, which can exploit several parallel processes by assigning each of them with one or more membranes. In this case the steps 2, 3, 5, 7.a and 8 of the algorithm are executed in parallel, with steps 4, 6, 7.b and 9 representing communications-synchronization points, to be performed in a collaborative way among the parallel processes.

The third parallelization paradigm is related to the evaluation in parallel of the propensity function, represented by the step 3.a which, for example can be performed by one thread for rule.

Considering that our aim is to develop a parallel implementation able to process systems of arbitrary size, in this version we focused on the first two levels of parallelization. We exploited a traditional clusters for the implementation of a test case, developing a C-MPI version of the algorithm to test the performance derived by the possible different usage of the resources. As regards the third level of parallelization, in fact, the possible improvements are proportional to the number of rules associated with each membrane, that in many cases are less than a dozen.

V. Experimental Results

We considered a model of reaction-diffusion system in a crowded environment. The system, composed by 81 membranes organised as a square lattice, contains four molecular species: s_1 and s_2 represent two generic proteins of volume 0.0001 which interact by means of a reversible process of association leading to the protein complex s_3. The specie s_4 represents a motionless crowding macromolecule with a volume equal to 0.1, a value which is three orders of magnitude bigger than the others. We carried out simulations to determine the behaviour of the system in 1 second. The system was initialised considering 100 molecules of s_1 and 100 molecules of s_2 in each membrane, while we choose the number of s_4 such that $\frac{1}{3}$ of the total volume was occupied by s_4 molecules, with a random distribution among the membranes.

We used a Linux cluster of 3 nodes, each of them equipped with two 6 cores Xeon processors and 32 GB of RAM, linked together with a Gigabit Ethernet network. We exploited also the Intel Cluster Studio XE 2013 suite, in particular the C compiler and the MPI Library implementation.

The execution times of both the sequential and parallel implementations of the algorithm are proportional to the number of iterations, that is the number of times the steps 3-9 are performed until the termination criteria is satisfied. This depends by both the initial conditions (the initial state and parameter values) and the chain of generated stochastic numbers. In general, a higher number of iterations is required whenever the system dynamics are fast, that is when many reactions occur and the system state (number of molecules) is significantly modified: in this scenario in fact small τ values will be required to satisfy the leap condition and any iteration will represent a short interval of time. It is worth noting that, due to the use of randomly-generated numbers, two runs with the same initial conditions are likely to have a different evolution, and thus a different number of iterations.

In our tests the fastest sequential execution using one core required 43.73 seconds for 6,244 iterations (wall clock

Table I
AVERAGE EXECUTION TIME, IN SECONDS, FOR A SINGLE ITERATION IN THE CASE OF A FAST EXECUTION (I.E. $\leq$ 10,000 ITERATIONS - 10K) AND OF A SLOW EXECUTION (I.E. $\geq$ 100,000 ITERATIONS - 100K). WE INDICATE WITH "N" THE EXECUTION OF ALL THE PARALLEL PROCESSES ON A SINGLE NODE, WHILE "C" THE EXECUTION USING ALL THE NODES

	$\leq$ 10K	Speedup	$\geq$ 100K	Speedup
Sequential	0.005		0.005	
6N	0.0015	3.33	0.0019	2.63
12N	0.00055	9.09	0.00075	6.66
24N	0.00077	6.49	0.00087	5.74
24C	0.0023	2.17	0.0024	2.08

time), the slowest one required 2,214.23 seconds for 427,156 iterations. The time for data acquisition depends on the size of the system description, that is of about 100 KB and therefore negligible, while the output time depends on a user-defined parameter that specifies the sampling frequency of the system status. By sampling every 100 iterations, in the slowest case the result correspond to more than 100 MB of output data, while the production of only the final state of the system results always in a size of 1.6 MB.

This is the reason why in the following we focus on the data processing time that, for the sequential algorithm, is respectively of 31.82 and 2,135.78 seconds. Moreover, due to the random behaviour of the system evolution, we analyze the performance of the parallel algorithm by comparing the average time for executing a single iteration that, in the sequential version, is of about 0.005 seconds.

Each node of the cluster has 12 cores and it is able to execute 24 threads at a time. Therefore we considered four execution patterns: 6, 12 and 24 parallel processes running on a single node, to test the scalability, and 24 parallel processes spread on the three nodes to test the impact of the communication overheads. We denoted them, respectively, 6N, 12N, 24N and 24C. Results are shown in Table I.

For each iteration the parallel algorithm requires the processes to communicate four times, as described in Section IV. They represent synchronization steps, implemented with the collective operation MPI_Allreduce. By analyzing the trace data of the cases 24N and 24C, shown respectively in Figure 1a and 1b, we can see that in 24C the time for the communication operations is about 8 times that to perform the computation. On the contrary if all the processes are on one node the ratio is about 2 times. It is worth noting that this last result is achievable only if an optimized implementation of the MPI library is used, as with the Intel one. General purpose implementations as MPICH2 1.2.5 in fact provide lower performance.

As regards the scalability, the aspect that affects the parallel execution times is the mapping between membranes and processors. The step size τ is selected in order to be the smallest among those independently computed within each membrane. In general only a subset of the membranes are able to execute reactions in each step, therefore some processes may not perform any update of their membranes. Moving from 6 to 12 processes the load remains, on average, balanced, while the unbalance is more relevant moving from 12 to 24 parallel processes.

On the basis of these results an important issue is which is the configuration that better exploits the computational capabilities considering the two levels of parallelism and the characteristics of the available resources. With our cluster, 12N is the case presenting the highest speedup value: it is therefore better the execution of a parallel computation with 12 processes for each Xeon processor (i.e the exploitation of both the first two levels of parallelism), or only sequential processes (i.e. only the first level)?

On the basis of the test we performed, on average, the first is the best solution. In fact the second case outperforms the first one only if all the sequential processes execute a comparable number of iterations and, therefore, have similar execution times. This case correspond to the ideal linear speedup and, as shown, our parallel implementation is sublinear. Otherwise the execution time corresponds to the time of the slowest sequential process. On a single processor, in the case of 11 sequential executions with 6,244 iterations (i.e. about 31 seconds) and one with 427,156 iterations (i.e. about 2,135 seconds), we will get the final result when this last ends. On the contrary, with the exploitation of the two levels of parallelism we will wait only about 357 seconds (357 = 11*3.4 seconds for the 11 fastest runs + 1*320.4 for the slowest one).

VI. CONCLUSIONS AND FUTURE PLANS

In this paper we presented the parallelization of the Sτ-DPP Stochastic Simulator, a tool for the analysis of the evolution of multi-volume biochemical systems in which the size of volumes and chemicals are taken into account to improve the accuracy of the results.

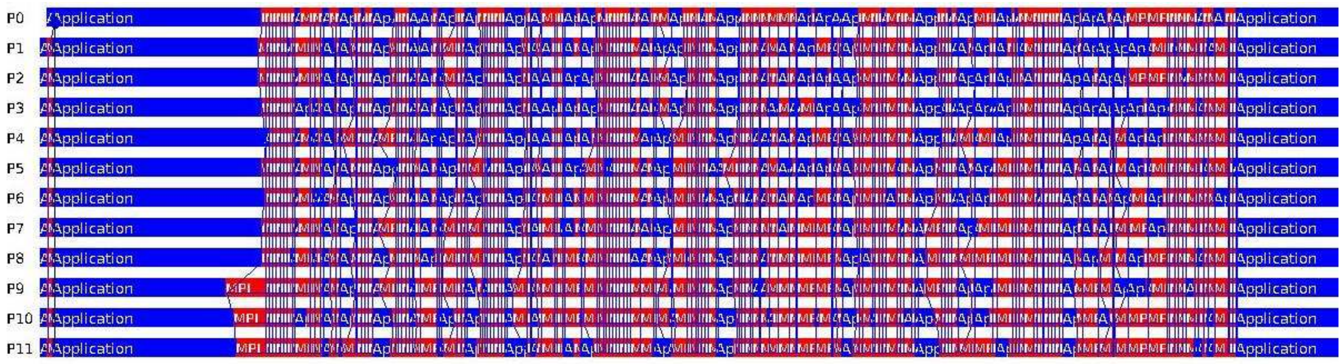
In particular we discussed our experience in the design of a parallel version able to exploit different levels of parallelism in order to get the best performance on the basis of the computational capabilities of the available resources.

As regards the future works, we plan to design a parallel implementation able to exploit the architectural characteristics of GPUs and coprocessors like the Xeon Phi, in order to reduce the impact of the load unbalancing and the communication overheads.

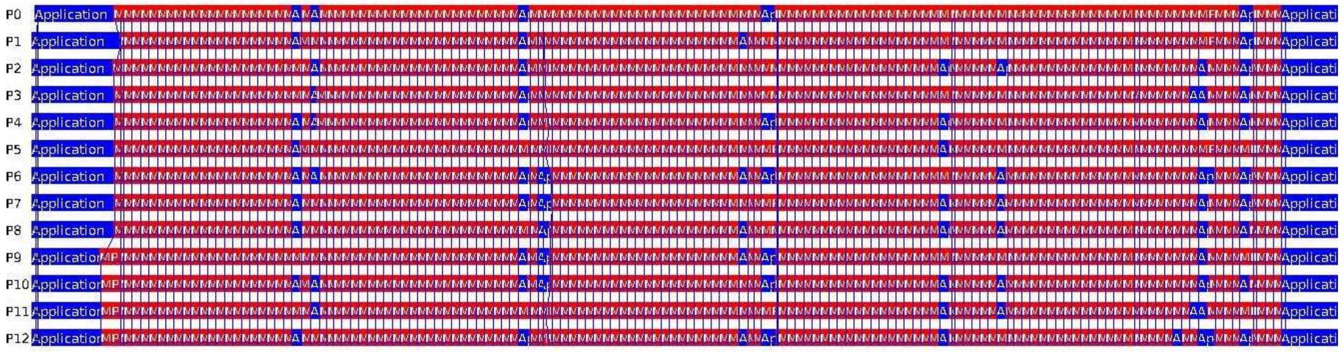
REFERENCES

[1] Păun, G., Pérez-Jiménez, M. J., Membrane computing: brief introduction, recent results and applications. Biosystems 85 (1):11-22, 2006.

[2] Cazzaniga, P., Pescini, D., Besozzi, D., and Mauri, G. Tau Leaping Stochastic Simulation Method in P Systems. Proceedings of the 7th International Workshop on Membrane Computing (WMC), LNCS 4361, 298-313, 2006.



(a) If all the 24 processes are allocated on the same node (case 24N) the communication time ("MPI" - light red) is about twice the time to perform the data process ("Application" - dark blue).



(b) If the processes are allocated using all the nodes of the cluster (case 24C) the execution is communication bounded.

Figure 1. These images show the trace analysis results of a slow execution in the cases 24N and 24C. The behaviour of the processes not shown due to lacks of space is similar.

[3] Cao, Y., Gillespie, D. T., and Petzold, L. R., Effcient step size selection for the tau-leaping simulation method. J Chem Phys 124 (4):044109, 2006.

[4] Cazzaniga, P., Mauri, G., Milanesi, L., Mosca, E., Pescini, D., A novel variant of tissue p systems for the modelling of biochemical systems. In: Paun, G., Perez-Jimenez, M., Riscos-Nunez, A., Rozenberg, G., Salomaa, A. (eds.) Proc. of the 10th International Workshop on Membrane Computing. LNCS 5957 : 210-226, 2009.

[5] Mosca, E., Cazzaniga, P., Pescini, D., Mauri, G., Milanesi, L., Modelling Spatial Heterogeneity and Macromolecular Crowding with Membrane Systems. Proc. of the Int. Conf. on Membrane Computing, LNCS 6501: 285-304, 2011.

[6] Gillespie, D. T., Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry 81: 2340-2361, 1977.

[7] Lok, L., The need for speed in stochastic simulation. Nature Biotechnology 22: 964 - 965, 2004.

[8] Aldinucci, M., Coppo, M., Damiani, F., Drocco, M., Torquati, M., Troina, A., On Designing Multicore-Aware Simulators for Biological Systems. International Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 318-325, 2011.

[9] Li, H., Cao, Y., Petzold L., Gillespie D., Algorithms and software for stochastic simulation of biochemical reacting systems. Biotechnology Progress, 24: 56-61, 2007.

[10] H. Li and L. Petzold. Efficient stochastic simulation of biochemical systems on the graphics processing unit. The International Journal of High Performance Computing Applications, 24(2): 107-116, 2010.

[11] Salwinski, L., Eisenberg, D., In silico simulation of biological network dynamics. Nat. Biotechnol. 22: 1017-1019, 2004.

[12] Mosca, E., Cazzaniga, P., Merelli, I, Pescini, D., Mauri, G., Milanesi, L., Stochastic Simulations on a Grid Framework for Parameter Sweep Applications in Biological Models, Proceedings of the International Conference on High Performance Computational Systems Biology 1: 33 - 42, 2009.

[13] Merelli, I., Pescini, D., Mosca, E., Cazzaniga, P., Maj, C., Mauri, G., Milanesi, L., Grid computing for sensitivity analysis of stochastic biological models. Proceedings of the 11th International Conference on Parallel Computing Technologies, 2011.