Solving Analytical Structured Formalisms in Parallel Using Slice Algorithm

Pedro Velho
Computer Science, UJF, LIG, INRIA
Grenoble, France
Email: pedro.velho@inrialpes.fr

Prof. Luiz Gustavo Leão Fernandes
Computer Science, PUCRS
Porto Alegre, Brazil
Email: luiz.fernandes@pucrs.br

***Abstract*—Analytical modeling can be used to predict performance, detect unexpected behavior and evaluate strategies in order to improve systems. In the context of modeling computational environments, a multitude of analytical structured modeling formalisms, such as Stochastic Automata Networks (SAN), is becoming popular since they provide high level abstractions and modularity. Indeed, in order to obtain performance estimations of a given SAN model, it is necessary to perform multiple matrix-vector multiplications. In structured formalisms, such as SAN, the matrix-vector multiplication is not presented in the usual format xA, since matrix A is replaced by an algebraic expression Q (called Markovian Descriptor or just descriptor, for short) due to modularity. The original multiplication is then replaced by a vector-descriptor multiplication (VDM), letting some space for algebraic improvements. Mainly, there are two algorithms that implement the VDM: shuffle and slice. In previous works, a parallel version of shuffle was proposed. It presented good results in terms of memory allocation but tasks are hard to split and hence compromises its scalability. The slice algorithm is based on an algebraic property that can, since its name says, split the VDM in many independent operations. The goal of this work is to underline a high performance version of the slice to check the scalability of this algorithm.**

***Keywords*-Parallel Applications, Kronecker/Markovian Descriptor, Structured Analytical Formalisms, Stochastic Automata Networks**

I. Introduction

Performance prediction has proven to be useful in a myriad of situations: to predict the performance of parallel applications [1], [2], establish constrains in algorithms design [1], evaluate the efficiency of fault-tolerant systems [3], among others.

Our focus throughout this paper is to propose a parallel version of the resolution of a set of formalisms employed in the stochastic analysis. Those formalisms are called **Analytical Structured Formalisms** (ASF) because they offer a structured and modular approach to represent **Markov Chains**. For the final user, the structure imposed by such formalism bring more understanding when modeling complex phenomena. In such formalism an equivalent but more efficient numerical treatment may be used in order to manage the time-memory trade-off.

All structured formalisms are based on the multiplication of an algebraic expression by a vector, Qx. This algebraic operation replaces the Markov Chain matrix-vector multiplication, Ax. Since the algebraic expression Q is called Markovian descriptor (throughout this paper we use just descriptor for the sake of brevity) it is called vector-descriptor multiplication (VDM). One of the advantages of using VDM is a more rational use of memory which limits most Markov Chain models. Although, the response time to achieve performance estimations is still unacceptable in many cases [1], [2], [3].

To fill that gap, one can find mainly two algorithms to compute the VDM. Firstly, the Shuffle [4], [5], [6] algorithm proposes an on-the-fly mapping of non-zero elements from the descriptor to the equivalent Markov Chain matrix. This approach significantly reduces the use of memory making the solution bounded by makespan. Secondly, the Slice [5] algorithm provides the opposite trade-off where mapping is performed in grouped data reducing the response time but increasing memory usage.

The use of high performance computing environments to improve the VDM has already been explored before in [4] for the Shuffle algorithm. To harness the computational and memory power of many machines is an alternative to CPU and memory bounded applications [7], [8], [9]. Although, the memory management of the algebraic structures in shuffle make it hard to exploit many processors, since the number of tasks depend on problem characteristics. In this context Slice can split the VDM operation in many small tasks. This implies more memory usage but we believe the scalability should be suitable to exploit a high performance computing environment. Although, a parallel Slice version is not yet existent. The objective of this paper is to underline a parallel version of the Slice algorithm hence extending the understanding CPU and memory limitations when solving such formalisms.

In Section II we state the VDM problem where the Slice algorithm is presented. Following, in Section III, we underline the parallel strategy. We present performance results in comparison with a sequential version in Section IV. Finally, in Section V, we state conclusions and directives for future work.

II. The Markovian Descriptor

Analytical structured formalisms (ASF) emerged in the last decade as an alternative to Markov Chain models. The

main motivation of using them is due to the lower storage cost. Avoiding the combination of elements which would result in zero (not affecting the calculation) is the key to reduce memory usage. Besides that, ASFs provide higher level abstraction because, in those formalisms, a system can be modeled in subsystems. In those cases the interaction among subsystem may be specified using synchronizing primitives [10], [11], [12].

To illustrate how the structured formalisms make life easier, the Markov Chain seen in Fig. 1 is a possible model to design a mutual exclusion problem of two process sharing a resource. Using an analytical structured formalism one can split the model in subsystems. As can be seen in Fig. 2 which shows a SAN approach modeling the same situation, each entity of the system (processor and resource) is modeled as a separate subsystem. The main lack of abstraction in Markov Chain when compared to structured formalisms is due to the fact that each state must be thought as a combination of two or more states in each entity. When the system being modeled is intrinsic divided in subsystems the use of an ASF is straight forward.

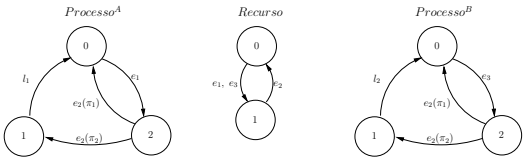

Figure 2. A SAN model that models the same problem of Fig. 1.

The automata are algebraic represented as a set of matrices. For instance, in Fig. 2, each automaton in that model has a matrix representing their state transitions. The equivalence of an ASF with Markov Chains is proven providing a way to map the specific matrices into a **Infinitesimal Generator** (IG). This Infinitesimal Generator is the unique algebraic structure in a Markov Chain model, which has the rates associated to all states transitions. The set of structures and the algebraic operation involved in the construction of the IG is called **Markovian Descriptor**, or descriptor for short. A descriptor is also called **Kronecker Descriptor** once that the algebraic operations and properties used to map then into a Markov Chain are known as Kronecker algebra [12].

There is a variety of formalisms that are based on the Markovian descriptor, such as: **Stochastic Automata Networks** [10] (SAN), **Stochastic Petri Nets** (SPN) [13], **Performance Evaluation Process Algebra** (PEPA) [14]. Those formalisms have in common the use of Kronecker operations to combine a set of matrices into a Markov Chain. Although, those formalisms are similar they differ in the rates associated and in the manner that transitions are tackled.

$$\sum_{k=1}^{T} \bigotimes_{i=1}^{N} Q_i^{(k)} \tag{1}$$

The format of a Markovian Descriptor is generalized as in (1). Although this expression summarize the core of structured analytical formalisms, they are rarely presented in that shape when the focus is to prove their equivalence to Markov Chains. For instance, in [15] a SPN descriptor is proposed as (2). Another example could be the SAN descriptor seen in (3), where E is the total amount of synchronizing events and N the number of automata. Both descriptors given by (2) and (3) are presented in terms of how to combine the matrices in order to prove their equivalence to classic Markov Chains. Once this paper aim is not to prove such equivalence, a more generic shape of the Markovian Descriptor, given by (1), is adopted.

$$R' = \bigoplus_{i=1}^{N} R^{(i)} + \sum_{\tau_j \in T} \bigotimes_{i=1}^{N} R^{(i,j)} \tag{2}$$

$$Q = \bigoplus_{i=1}^{N} Q_l^{(i)} + \sum_{e=1}^{E} \left(\bigotimes_{i=1}^{N} Q_{e^+}^{(i)} + \bigotimes_{i=1}^{N} Q_{e^-}^{(i)} \right) \tag{3}$$

The resolution of structured analytical models is made, analogue to a Markov Chain model, solving a linear equation system. Such as in Markov Chain models, this resolution is based on iterative processes which are based on a vector-matrix multiplication at each iteration (4).

$$xA \tag{4}$$

However, when using a structured analytical formalism the coefficient matrix A is replaced by the Markovian Descriptor, (5). Hence some complexity is added, once the user deals with the manipulation of a greater amount of matrices whose elements must be combined and mapped into a matrix equation. This process of combining and mapping elements is made using the Kronecker operations defined over matrices: Kronecker product (denoted by $\otimes$) and Kronecker sum (denoted by $\oplus$).

$$xQ = x \left(\sum_{k=1}^{T} \bigotimes_{i=1}^{N} Q_i^{(k)} \right) \tag{5}$$

Numerical iterative methods are suitable to solve structured analytical models [10]. The iterative method used may vary: Power method, Arnoldi method, and so on. Nevertheless, all iterative methods are based on the same basic operation [16]: the vector-descriptor multiplication.

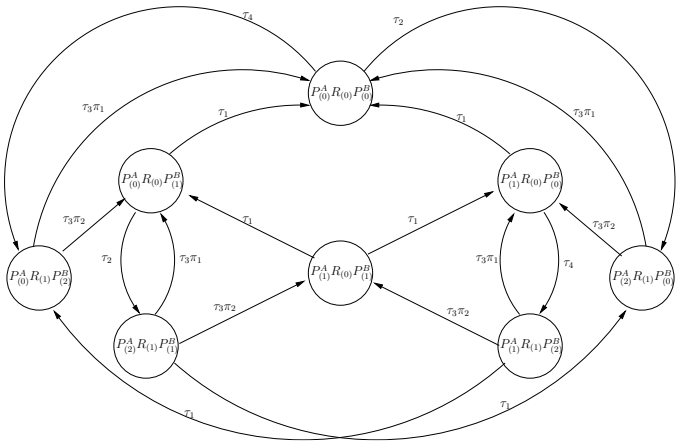

Figure 1. Markov Chain modeling the mutual exclusion problem with two process and one resource.

A. Slice Algorithm

The parallel Slice algorithm presented here is focused in one single multiplication. Slice algorithm is based on the additive unitary normal factor decomposition property. This property holds that given a series of Kronecker products, the left side (6), it can be factorized in a sum of scalar multiplications (7). To maintain the position, i and j, where each scalar will be placed in the resulting equation matrix we combine the i and j information of each scalar. The formalization of this property is presented below.

$$Q^{(1)} \otimes \cdots \otimes Q^{(N)} = \tag{6}$$

$$\sum_{i_1=1}^{n_1} \cdots \sum_{i_N=1}^{n_N} \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \left(\hat{q}^1_{(i_1 j_1)} \otimes \cdots \otimes \hat{q}^N_{(i_N j_N)} \right) \tag{7}$$

Applying this properties in the $N-1$ leftmost Kronecker product is the core idea of slice. We call the scalars resulting form the $N-1$ first matrices **Additive Unitary Normal Factors** (AUNF or factor for short), because they are actually not just scalars but a triple of one scalar and its i, j position in the resulting matrix. Since slice generates a significant number of AUNFS this implies in more memory usage when compared to Shuffle.

III. Parallel Multiplication

The approach used to execute an iteration in parallel is presented in Fig. 3. In this communication scheme the processor with lowest rank is responsible to send the vector of the current iteration, x^k, to all processors starting an iteration step. Once a processor gets x^k it starts the VDM multiplication concerning the AUNFs (tasks) previously assigned. Afterward, each processor returns a partial vector of the next iteration, x_p^{k+1} (where p denotes the processor id). Finally, the partial vector are summed resulting in the next iteration vector x^{k+1} finishing one step of the parallel VDM. This procedure is executed several times until a solution is reached.

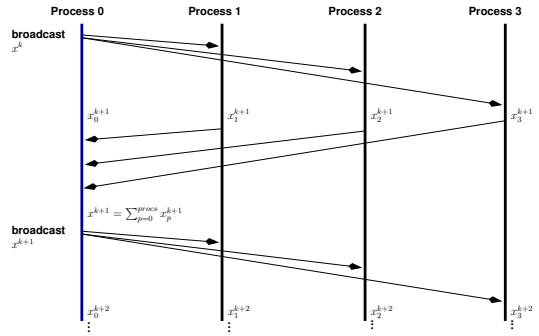

Figure 3. A high level overview of the parallel vector-product multiplication.

Our aim with this parallel approach is to verify its scalability. To do so, we propose two granularities: (i) coarse grain, map each model automaton in one task, this approach is the same as presented in [4] for parallel Shuffle; (ii) fine grain, where each AUNF is one task, this approach should improve scalability.

A. Coarse Grain

The Markovian Descriptor adopted in (1) is a sum. A coarse granularity approach is then obtained using the distributive property over the sum as given by (8). In this strategy the maximum amount of tasks is the total number of algebraic terms (T). This approach is similar to the one used in previous works for the parallel shuffle [12].

$$\sum_{k=1}^{T} \left(x \left(\bigotimes_{i=1}^{N} Q_i^{(k)} \right) \right) \tag{8}$$

Both algorithms, shuffle and slice, can benefit from this method of distributing computation. Once this approach

Table I
TEST CASES PARAMETERS IN DETAIL.

Test Case Label	# of Coarse Grains	# of Fine Grains	Density	Sequential Time (s)
MUTEX	32	9469952	46.25 %	13.68
MIXED	16	3280080	19.88 %	5.95
DENSE (A)	16	10649600	22.13 %	13.14
DENSE (B)	16	48771072	27.52 %	64.45

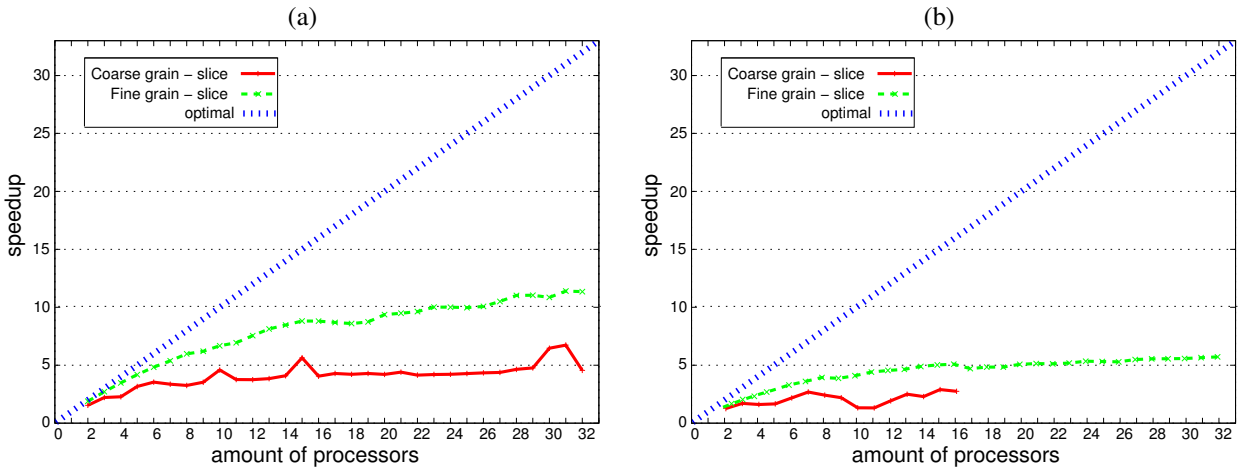

Figure 4. Speedup for the mutex (a) and mixed (b) test cases.

is based on distributing the multiplication directly in the Markovian Descriptor level, which is the same structure for both algorithms.

In order to balance the amount of computation given to each process a static scheduling approach based in the cost of each task is used. This strategy is based on computing each task computational cost before execution. Next, the smallest task will be assigned to the processor with less load. The number of steps taken depends on the total amount of tasks (or terms, T) and the number of processing units P.

One advantage of using coarse granularity is that the computation of AUNFs can be done in parallel, since there is no interdependence between each term. Although, the scalability problem can not be tackled without the use of some algebraic property, which leads us to the fine grain approach.

B. Fine Grain

Using each AUNF as an independent task is hence explored as an alternative. AUNFs are homogeneous in terms of computational cost. Moreover, AUNFs are dependent of problem complexity and do not rely on problem characteristics as before. For these reasons, the fine grain approach seems to be suitable to increase scalability.

One drawback of fine grain is that the AUNF generation step can not be distributed. This step can be distributed when using coarse grain because a term is always entirely assigned to a processor. However, using fine grain, tasks have to be computed in a preconditioning step.

The preconditioning step represent a significant computing effort. Even though, the time taken by this step is slightly greater than one iteration. Once it is needed a significant amount of iterations to solve a model, typically 1000 up to 10000, this cost is attenuated in the final execution time (less than 1 % of overall makespan).

IV. PERFORMANCE EVALUATION

Tests were carried out in Grid5000 using machines from the i-cluster2 located in Montbonnot, France inside the INRIA Rhône-Alpes facility. Each of the 104 nodes of i-cluster2 features two Itanium-2 64 bits processors at 900 MHz, 3 Gigabytes of memory and 72 gigabytes of local disk. The nodes are interconnected through Gigabit ethernet switch. In order to evaluate the algorithm using different test cases we choose four Markovian Descriptors detailed in Tab. I.

Aiming to test how this version would behave using classic models, the test cases mutex and mixed are two real algebraic representation of SAN models. The first, named mutex, was extracted from a SAN model that implements the problem of 16 processors sharing 4 resources in a race condition. The second, called mixed, is a hypothetical SAN model that stress different parameters of this formalism such

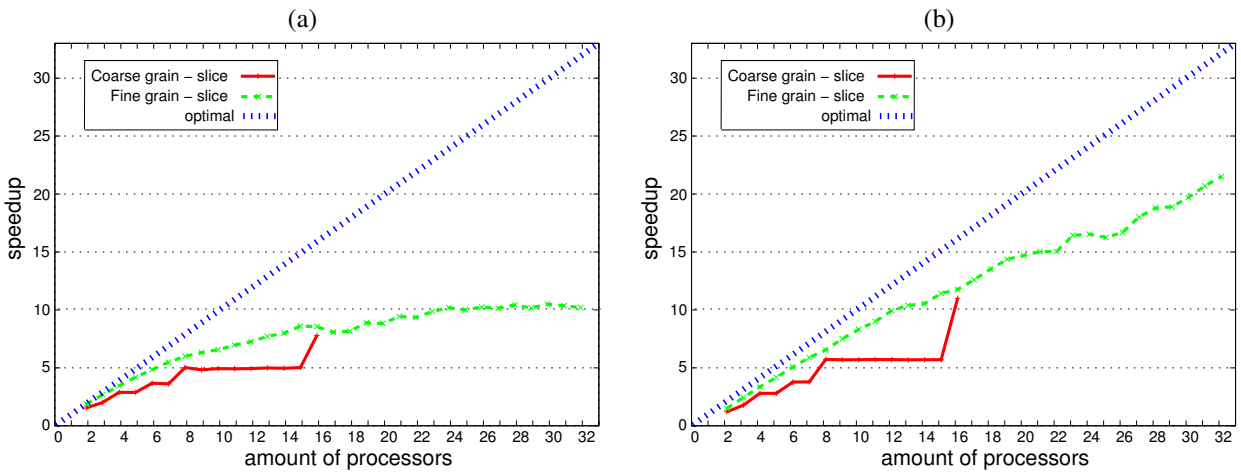

Figure 5. Speedup for the dense (a) and (b) test cases.

as: number of states per automata; number of synchronizing events; among others [10].

The other two tests, dense (a) and (b), were obtained inflating the number of non-null elements from the mixed test case. The overall mean of non-null elements is obtained using the equation (9). In (9), T means the total amount of terms and N means the total number of matrices that compound each term. The variable nz and n are respectively the number of non-null elements and the order of each matrix i from a given term k.

$$\left(\frac{\sum_{k=1}^{T} \sum_{i=1}^{N} \frac{nz_i^{(k)}}{n_i^{(k)} \times n_i^{(k)}}}{N \times T} \right) \times 100 \qquad (9)$$

The speedup achieved are presented in Figs. 4 and 5 for both grains, fine and coarse. Test cases mutex, mixed, dense (a) and dense (b) are presented respectively in Figs. 4-a, 4-b, 5-a and 5-b. The results were obtained through a real experiment running the application 10 times and in each one executing 100 iterations. The mean of one iteration execution time was computed discarding the higher and the lowest values obtained. Even though, the standard deviation was observed to avoid intrusiveness such as other applications running at the same time or cache effects.

In the two real test cases, mutex (Fig. 4-a) and mixed (Fig. 4-b), the behavior of both proposed grains are similar. One can observe that the speedup is distant from the ideal and that the finer grain approach is always better because it can scale further, for those test cases. The test case mutex has 32 maximum tasks with coarse grain approach due to the problem characteristics which is limited to 32 terms, the other tests are 16 terms each.

Observing, Figs. 5-a and 5-b, one can see that the solution can scale better maintaining speedup factor closer to optimal for more computational intensive test cases. Note that the coarse grain is limited to 16 tasks in Dense-A/B because of their amount of terms, as showed in Tab. I.

Figs. 5-a and 5-b highlight coarse grain scalability issue. The presence of some constant speedup is due to task heterogeneous characteristics, hence one task becomes the makespan bottleneck. The coarse grain approach, originally inspired by the successful parallel implementation of shuffle [4], is not suitable for slice algorithm.

V. Conclusions

The main goal of this work was the parallel implementation of the Slice algorithm to provide a effective scalable parallel vector-descriptor multiplication (VDM). This operation is used by a multitude of structured formalism based on Markovian Chains.

Table II
Best results achieved.

Test case	Best results		Sequential time
	Fine grain	Coarse grain	
MUTEX	1.199684	2.030804	13.688620
MIXED	0.963382	1.881302	5.953364
DENSE (A)	1.241858	1.668218	13.014060
DENSE (B)	2.941724	5.664326	80.211800

The results presented show a significant gain of performance summarized in Tab. II. In all test cases the fine grain has presented better results than the coarse grain approach. Graphs show that fine grain has also better scalability. However, it is still an open point when to use shuffle or slice. Since both have different characteristics. A scalability vs preconditioning and time-memory trade-off analysis will be done in future work.

REFERENCES

[1] L. Baldo, L. Brenner, L. G. Fernandes, and A. Sales, "Performance Models for Master/Slave Parallel Programs," *Electronic Notes in Theoretical Computer Science*, vol. 104, no. 1, pp. 65–84, 2005.

[2] L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales, "Performance Analysis Issues for Parallel Implementations of Propagation Algorithm," in *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, Publisher ACM Press, Ed., Sao Paulo, Brazil, 2003, pp. 183–190.

[3] C. Bertolini, L. Brenner, A. Sales, P. Fernandes, and A. Zorzo, "Structured Stochastic Modeling of Fault-Tolerant Systems," in *12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS'04*, New York: IEEE Press, Ed., Volendam, Netherlands, 2004, pp. 139–146.

[4] L. Baldo, L. G. Fernandes, P. Roisenberg, P. Velho, and T. Webber, "Parallel PEPS Tool Performance Analysis Using Stochastic Automata Networks," in *Euro-Par*, Piza, Italy, 2004, pp. 214–219.

[5] Pedro Velho, "Acelerando a Ferramenta PEPS através da Paralelização do Algoritmo Shuffle para a Multiplicação Vetor-Descritor," Trabalho Individual I, PUCRS-PPGCC, Porto Alegre, Brasil, 2005.

[6] T. Webber, "Alternativas para o Tratamento Numérico Otimizado da Multiplicação Vetor-Descritor," Dissertação de Mestrado, Porto Alegre, Brasil, 2003.

[7] L. G. Fernandes, E. Bezerra, F. Oliveira, M. Raeder, P. Velho, and L. Amaral, "Probe Effect Mitigation in the Software Testing of Parallel Systems," in *Latin-American Test Workshop LATW*, Buenos Aires, 2006, pp. 153–158.

[8] M. Kolberg, L. Baldo, P. Velho, L. G. Fernandes, and D. M. Claudio, "Optimizing a Parallel Self-verified Method for Solving Linear Systems," in *Workshop on State-of-the-Art in Scientific and Parallel Computing - PARA*, Umea, 2006, pp. 1–10.

[9] P. Velho, L. G. Fernandes, M. Raeder, M. Castro, and L. Baldo, "A Parallel Version for the Propagation Algorithm," in *8th International Conference on Parallel Computing Technologies - PaCT*, vol. (LNCS 3606), Kranoyarsk, 2005, pp. 403–412.

[10] P. Fernandes, "Methodes Numériques pour la Solution de Systèmes Markoviens a Grand Space d'états," Ph.D. dissertation, Institut National Polytechnique de Grenoble, Grenoble, 1998.

[11] P. Fernandes, B. Plateau, and W. J. Stewart, "Efficient descriptor-Vector Multiplication in Stochastic Automata Networks," *Journal of the ACM*, vol. 45, no. 3, pp. 381 – 414, 1998.

[12] P. Fernandes, R. Presotto, A. Sales, and T. Webber, "An Alternative Algorithm to Multiply a Vector by a Kronecker Represented Descriptor," in *21th Annual UK Performance Engineering Workshop, UKPEW*, Newcastle, England, 2005, p. 3.

[13] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, "Modelling with Generalized Stochastic Petri Nets," *Special issue on Stochastic Petri Nets*, vol. 26, no. 2, p. 2, 1998.

[14] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo, "PEPA Nets: A Structured Performance Modelling Formalism," *Performance Evaluation*, vol. 17, no. 10, pp. 79–104, 2003.

[15] G. Ciardo and M. Tilgner, "On the Use of Kronecker Operators for the Solution of Generalized Stochastic Petri Nets," ICASE, Tech. Rep. 96(35), 1996.

[16] K. Atif and B. Plateau, "Stochastic Automata Networks for Modelling Parallel Systems," *IEEE Transactions on Software Engineering*, vol. 17, no. 10, pp. 1093–1108, 1991.