FACULDADE DE INFORMTICA

PUCRS - Brazil

http://www.pucrs.br/inf/pos/

Parallel PEPS Tool Performance Analysis Using Stochastic Automata Networks

L. Baldo, L. Fernandes, P. Roisenberg, P. Velho, T. Webber

TECHNICAL REPORT SERIES

Number tr039
April, 2004

Contact:
lbaldo@inf.pucrs.br
gustavo@inf.pucrs.br
www.inf.pucrs.br/~gustavo
paulo.roisenberg@hp.com
pedro@inf.pucrs.br
twebber@inf.pucrs.br
www.inf.pucrs.br/~twebber

Published by PPGCC - FACIN - PUCRS
Av. Ipiranga, 6681
90619-900 Porto Algre - RS - Brasil

Parallel PEPS Tool Performance Analysis Using Stochastic Automata Networks ⋆

Lucas Baldo[1], Luiz Gustavo Fernandes[1], Paulo Roisenberg[2],
Pedro Velho[1] and Thaís Webber[1]

[1] Faculdade de Informática, PUCRS
Avenida Ipiranga, 6681 Prédio 16 - Porto Alegre, Brazil
{lucas_baldo, gustavo, pedro, twebber}@inf.pucrs.br

[2] HP do Brasil
Avenida Ipiranga, 6681 Prédio 91A - TecnoPuc - Porto Alegre, Brazil
paulo.roisenberg@hp.com

Abstract. This paper presents a theoretical performance analysis of a parallel implementation of a tool called Performance Evaluation for Parallel Systems (PEPS). PEPS is a software tool used to analyze Stochastic Automata Networks (SAN) models. In its sequential version, the cost of the execution time becomes impracticable when analyzing large SAN models. A parallel version of the PEPS tool using distributed memory is proposed and modelled with SAN formalism. After, the sequential version of PEPS tool itself is applied to predict the performance of this model.

1 Introduction

In recent years, the effort of many authors has confirmed the importance of performance prediction of parallel implementations. Some authors have presented generic studies offering options to the performance prediction of parallel implementations [1–3]. The research community classifies the different approaches in three quite distinct groups: monitoring, simulation and analytical modelling. This last approach (analytical modelling), compared to the two first ones, is more rarely employed to achieve parallel programs performance prediction. This happens due to a frequently misconception: the most known formalisms to analytical modelling, *e.g.*, Markov chains [4] and queueing networks [5], are not very suitable to represent parallelism and synchronization.

In this paper, we adopted the analytical modelling approach using a formalism called Stochastic Automata Networks (SAN). The reader interested in a formal description of the formalism can consult previous publications [6, 7]. The SAN formalism describes a complete system as a collection of subsystems that interact with each other. Each subsystem is described as a stochastic automaton, *i.e.*, an automaton in which the transitions are labelled with probabilistic and timing information. The analysis of the theoretical SAN models is performed by a software package called Performance Evaluation for Parallel Systems (PEPS) [8]. To analyze models consists of obtaining performance measures for parallel systems described in SAN. The performance measures

⋆ This work is partially supported by HP do Brasil - PUCRS agreement CAP (T.A. 30) project.

given by PEPS correspond to the probability to be on a specific state, which represents a slice of time that a model remains in a state. This application is a powerful tool to evaluate performance in a theoretical level of parallel implementations.

Although PEPS has proven its usability during the past years, it presents an important drawback: the computational cost of the execution time to analyze SAN models with too many states is very often impracticable. Thus, the objective of this paper is to verify the feasibility of a parallel implementation of PEPS tool using a SAN model which will be analyzed by the PEPS tool itself. We intend to get as close as possible of a fine-tuning performance prediction of the parallel PEPS implementation based on the communication and processing tasks description.

The next section describes the main aspects of PEPS tool implementation which are relevant to the parallel solution. Section 3 presents the proposed parallel version of PEPS. The representative SAN model and its parameters for the PEPS parallel version are discussed in section 4. Section 5 presents the results obtained so far. Finally, the conclusion draws the next steps to continue this work and summarizes the lessons learned until the present time.

2 PEPS Implementation Analysis

The input of PEPS is a SAN model described in a predefined grammar. The current application loads this grammar and builds an equation system using the events rates. The SAN models allow a better management of the needs for space memory than Markov Chains, because they are described in a more compact and efficient way. This optimization is possible due to the use of tensorial algebra [6, 7]. Thus, a model is no more described by a unique matrix, but instead, by a new structure called Markovian Descriptor. This structure is the kernel of the PEPS tool and it has a significant impact over its execution time. Considering a network with N automata and E synchronizing events, the Markovian Descriptor is given by:

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

In this equation, there are N matrices $Q_l^{(i)}$ representing the local events and $2E$ matrices $Q_{e_k}^{(i)}$ representing the synchronizing events, which result in a total of $N + 2E$ stored matrices.

The basic operation to solve a SAN model [3] is to multiply a probability vector π by a Q matrix stored in the Markovian Descriptor form. This probability vector assigns a probability π_i $(i \in \{1, 2, ..., n\})$ to each one of the n states of the Markov Chain equivalent to the SAN model. Each element of Q may represent a local event or a synchronizing event. The local events are given by a sum of normal factors. In the other hand, synchronizing events are given by a product of normal factors [7]. Hence, the most important point in this descriptor-vector multiplication is to know how one can

[3] In numeric iterative methods like the Power Method, GMRES, etc.

multiply normal factors.

According to Fernandes, Plateau and Stewart [7] the complete computational cost of the descriptor-vector multiplication is given by:

$$\sum_{k=1}^{1+2|E|} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i{}^{(k)}}{n_i} \right) \tag{2}$$

where n_i stands for the order of the i^{th} matrix and nz_i stands for number of non zero elements of the i^{th} matrix. Thus, an optimization can be added to the Markovian Descriptor according to equation 3, where the matrices $\bar{Q}_l^{(i)}$, which correspond to local events, do not contain the diagonal negative elements anymore. These elements are now stored in a separated vector D. Consequently the matrices that represent the negative synchronizing part of the model can also be stored in vector D.

$$Q = \bigoplus_{i=1}^{N} \bar{Q}_l^{(i)} + \sum_{e=1}^{E} \bigotimes_{i=1}^{N} Q_{e^+}^{(i)} \; + \; D \tag{3}$$

In spite of the Markovian Descriptor optimization, the PEPS application still suffers from a performance decline at the same time that the complexity[4] of the model grows. Table 1 shows some examples of the relation between a number of states and the memory and time used to execute 100 iterations [5].

Table 1. Memory and time used for 100 iterations.

Number of States	Memory(*Kb*)	Time(*sec*)
1,048,576	8,207	1,204.80
28,697,814	224,222	3,216.20
53,477,376	417,811	12,118.00
67,108,864	524,296	37,078.40

Note that the time to execute a model with 67,108,864 states triples comparing with the time spent to execute the 53,477,376 states model. This can be explained because the number of matrices increases and besides they become denser , resulting in a larger computational cost for the multiplications [8].

3 PEPS Parallel Version

In order to improve the PEPS software tool performance, this paper proposes a parallel version for this tool based on the features of a specific kind of high performance

[4] Complexity here is related to synchronizing events and states amount.

[5] The target architecture is a Pentium IV Xeon 2.4 Ghz clock

architecture. To reinforce the usability of this new version, the hardware environment should be based on an usual and not very expensive (compared to a supercomputer) one. Following this scenario, the new version is designed to run over a cluster architecture which has several processors connected by a dedicated fast network.

As seen in section 2, PEPS solves a SAN model using numeric iterative methods. In order to represent a SAN model, many matrices are created, describing local and synchronized events rates. Another feature of PEPS is that the number of iterations necessary to make the model converge is different from one input model to another. Due to this, it is not possible to deduce how many iterations are necessary to determinate the convergence of a model.

The solution proposed here is based on a synchronized master-slave model. Considering that a SAN model is described as a set of tensorial matrices (from the Markovian Descriptor, equation 3) and each iteration represents the multiplication of a probability vector by these matrices, the main idea is to distribute a set of matrices to each slave and execute the multiplications concurrently. In each iteration, the master node sends, in broadcast, the vector from the i^{th} iteration to the slaves. Each slave takes some time processing its own task, and returns to the master a piece of the next iteration vector. For each iteration, the master must wait for the results from all slaves (reduce operation) to combine them into the next iteration vector. Finally, the master sends this new probability vector in broadcast to the slaves, starting a new iteration. This procedure will continue until the convergence criteria of the numerical method is matched.

4 SAN Model for parallel PEPS

Taking on a parallel point of view, the main relevant states to be modelled using SAN are those who are focused on processes data exchange and computing time. That happens because the trade-off between this two features is the key to determine the success of the parallel implementation. The SAN model which describes the PEPS parallel implementation explained in section 3 is presented in Fig. 1. The model contains one automaton *Master* and *P* automata $Slave^{(i)}$ (*i* = 1..*P*).

The automaton *Master* has five states *ITx*, *Tx*, *Rx*, *Cx* and *FCx*. It is responsible over the distribution of iteration vectors to the slaves. The states *ITx*, *Tx* and *Rx* means, respectively, the initial transmission of the matrices and the first vector to the slaves, transmission of new iteration vectors to slaves, and the reception of the resulting vectors evaluated by the slaves. The states *Cx* and *FCx* represents, respectively, the time spent to sum all slaves evaluated vectors and write on file the asked results.

The occurrence of the synchronizing event *in* broadcasts the matrices and the first vector to the slaves. On the other hand, the occurrence of the event *end* finalizes the communication between master and slaves. The synchronizing event *s* broadcasts the vector of the i^{th} iteration to slaves. The synchronizing event r_i receives one resulting vector of the slave *i*. The occurrence of the events $r_{1..P}$ can change the state of the master automaton or not. Those events are related by the probability π_1, shown on fig. 1.

The *Master* automaton initializes new iterations or finalizes the descriptor-vector multiplication by the occurrence of the local event *c*. In 99% of the times (represented

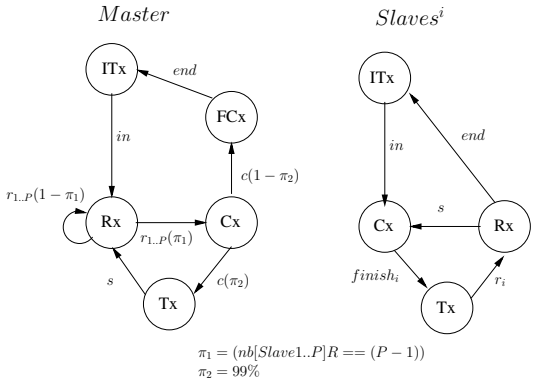

Fig. 1. The PEPS SAN model.

by the probability π_2) the event *c* will initialize new iterations. On the remaining time $(1 - \pi_2)$, the event *c* will finalize the descriptor-vector multiplications.

Finally, the automaton $Slave^{(i)}$ represents the slave of index i, where $i = 1..P$. It has four states: *I* (Idle), *Cx* (Computing), *Tx* (Transmitting) and *Rx* (Receiving). All slaves start their tasks when synchronizing event *in* occurs. Slave *i* stops processing a task by the occurrence of the local event $finish_i$. The same slave sends its resulting vector to the master with the synchronizing event r_i. When the event *s* occurs, all slaves start to compute a new step. The occurrence of the event *end* finalizes the execution of the slaves processes.

In order to complete the model representation it is necessary to assign occurrence rates to its events. The rate of one event is the inverse of the time spent in its associated state.

The occurrence rate of the event end is extremely high because the time spent to finish a model is insignificant. The occurrence rates of events s, in and c are inversely dependent by the probability vector size and also by the number of slaves nodes. On the other hand the rate of the events $r_{1..p}$ are only inversely dependent by the probability vector size.

The rate of the events $finish_{1..p}$ are inversely dependent by the number of multiplications each slave will perform. This number of multiplications is directly dependent by the vector probability size and inversely dependent by the number of slaves nodes.

Table 2 shows the relevant event rates for three SAN models with different size of the probability vector and four slave processes. Those values were obtained with some statistical measurements performed through some sample programs over the target architecture[6]. The procedure was to get five execution times of each program, eliminating the lowest and highest values. The mean value for the three remaining samples was used. Thus, the broadcast and unicast times were obtained directly. The time spent to

[6] The target architecture is a COW (Cluster of Workstations) heterogenous in which PE's may vary from two different processor architectures pentium III 500 Mhz clock to pentium IV 1.0 GHz.

compute a vector by one slave was obtained through a program that simulates the number of multiplications performed by one slave. Finally, the time spent to compute the resulting vectors in the master node was obtained through a program which simulates the sum of the resulting vectors sent by all slaves.

Table 2. Event rates for four different SAN models using four slave nodes.

Case Study	Size	s	r_i	$finish_i$
16384	256 KB	38.31	94.34	17.15
65536	1 MB	9.42	21.46	3.28
327680	5 MB	1.92	4.42	0.30

5 Results

The results given by PEPS are steady state probabilities. For instance, the time that the system stay in state *Tx* of the automaton *Master* is equal to the probability of the system be in this state. These probabilities are important information to indicate the amount of time the system will stay in a given state. For a parallel implementation this information will help to verify if the proposed parallel solution has bottlenecks which compromise its performance.

In order to evaluate the behavior of the proposed SAN model for the parallel version of the PEPS tool, three case studies will be presented next. Each one uses an hypothetical different input SAN model which differs in the number of states (directly related to the size of the probability vector). The behavior of the PEPS parallel version will be analyzed for each input model. Readers must pay attention that the sequential version of PEPS is used to analyze the SAN model that represents the proposed parallel version of PEPS, which needs some input SAN models to solve. Another important remark is that for each new slave added, the SAN model of the parallel version of PEPS changes because a new automaton Slave must be added to the model. Thus, the computational cost to solve the parallel PEPS model strongly increases and that is the reason why we have carried out experiments with only until seven slaves nodes for each case study.

Table 3, Tab. 4 and Tab. 5 show the state stay probabilities for the set of slaves nodes used. Analyzing the figures of theses three tables, one can find out that all case studies present similar behavior, *i.e.*the proposed solution does not present any irregular behavior. The most important remark is that the probabilities of the slaves nodes be computing (Cx_{slaves}) declines as the number of slaves grows, indicating that more work is done in parallel. Other interesting remark is that increasing the number of slaves does not means that the master will have a much bigger probability of be sending (Tx_{master}) or receiving (Rx_{master}) information to or from the slaves. These values do increase, but not in a dramatic way meaning that for until seven slaves the master is still not the bottleneck of the system. That explains why the probability of the master be computing (Cx_{master}) declines significantly.

Table 3. Results for the case study 16384.

slaves	Tx_{Master}	Rx_{Master}	Cx_{Master}	Cx_{Slave}
2	0.3690	0.1834	0.0025	0.0585
3	0.3790	0.1887	0.0011	0.0369
4	0.3810	0.1910	0.0005	0.0251
5	0.3811	0.1939	0.0003	0.0177
6	0.3808	0.1961	0.0002	0.0128
7	0.3805	0.1975	0.0001	0.0092

Table 4. Results for the case study 65536.

slaves	Tx_{Master}	Rx_{Master}	Cx_{Master}	Cx_{Slave}
2	0.3687	0.2258	0.0007	0.0797
3	0.3788	0.2034	0.0002	0.0449
4	0.3808	0.1987	0.0001	0.0289
5	0.3809	0.1984	0.0001	0.0197
6	0.3806	0.1988	3.79e-5	0.0139
7	0.3804	0.1992	2.31e-5	0.0099

Another interesting remark can be done looking comparatively the results of Tab. 3, Tab. 4 and Tab. 5. In fact, the probability of the slaves remain in the state Cx becomes higher as the number of states at the input model grows. This is another coherent result, because if the system has a higher workload, it is expected that the nodes spent more time computing.

Until this point, we have worked with SAN models that represent a parallel environment with an homogeneous event $finish$ rate for all slaves automata. This homogeneity means that all slaves must compute their tasks at the same time, ignoring aspects such as different architectures for each node or different workloads for the nodes in terms of the matrices computational cost (they do not have the same values and can be more or less sparse). In order to verify the importance of the workload balance for the parallel PEPS implementation, heterogeneity in the slaves behavior was introduced in the SAN model for the parallel version of PEPS. That was done by using different computing rates for each slave. Table 6 presents the results of this experiment, where using the input model with 16384 states and fixing the number of slaves on 6, four different configurations were analyzed: none, low, medium or high heterogeneity. The results show that as heterogeneity grows, the degree of discrepancy among of the slaves be in state Cx increases. Through this analysis, the model seems to be capable of indicate the impact that heterogeneity in the slaves behavior could represent over a parallel implementation.

Table 5. Results for the case study 327680.

slaves	Tx_{Master}	Rx_{Master}	Cx_{Master}	Cx_{Slave}
2	0.3628	0.2422	6.95e-5	0.0926
3	0.3764	0.2098	2.45e-5	0.0490
4	0.3796	0.2018	1.14e-5	0.0306
5	0.3802	0.2000	6.18e-6	0.0206
6	0.3802	0.1999	3.58e-6	0.0144
7	0.3802	0.1999	2.18e-6	0.0102

Table 6. Heterogeneous results for the case 16384.

$degree$	Cx_{S1}	Cx_{S2}	Cx_{S3}	Cx_{S4}	Cx_{S5}	Cx_{S6}
none	0.0128	0.0128	0.0128	0.0128	0.0128	0.0128
low	0.0120	0.0144	0.0120	0.0144	0.0120	0.0114
medium	0.0161	0.0161	0.0051	0.0161	0.0161	0.0016
high	0.0218	0.0190	0.0087	0.0087	0.0044	0.0044

6 Conclusion

The main contribution of this paper is to point out the advantages and problems of the analytical modelling approach applied to predict the performance of a parallel implementation. According to authors best knowledge, some others efforts to predict performance of parallel systems were not so formally. The recent work of Gemund [9] exploits

a formal definition used for simulations to automatically generate a formal description of a parallel implementation. More conservative works [5, 10] tend to use unappropriated formalisms (queueing networks) where the synchronization processes are not easy to describe. The SAN formalism, instead, was quite adequate to describe the behavior of a master-slave implementation. The definition of the events was very intuitive; their rates and probabilities were a little bit more hard to obtain but still intuitive. The stationary solution of the model provided enough prediction information about the behavior of the parallel version of the PEPS tool. The master and slaves nodes behaviors were clearly understood, and an analysis about the workload balance was also possible.

The natural future work for this paper is to verify the accuracy of the proposed model by comparison with the real parallel implementation. It is necessary to run the parallel algorithm in completely known environment to do the fine-tuning of the proposed model, *i.e.*, to verify the modelling choices made. In fact, the successive refinements in the model is a quite common technique in analytical modelling.

Finally, it is the authors opinion that the use of a SAN model to predict the behavior of the parallel PEPS tool was useful. This analysis allowed us to clearly identify some key aspects that we will have to face during the implementation of the parallel version.

References

1. Hu, L., Gorton, I.: Performance Evaluation for Parallel Systems: A Survey. Technical Report 9707, University of NSW, Sydney, Australia (1997)
2. McLean, T., Fujimoto, R.: Predictable Time Management for Real-Time Distributed Simulation. In: Proceedings of the 17^{th} Workshop on Parallel and Distributed Simulation (PADS'03), San Diego, California, USA, ACM (2003) 89–96
3. Nicol, D.: Utility Analisys of Parallel Simulation. In: Proceedings of the 17^{th} Workshop on Parallel and Distributed Simulation (PADS'03), San Diego, California, USA, ACM (2003) 123–132
4. Stewart, W.J.: Introduction to the numerical solution of Markov chains. Princeton University Press (1994)
5. Gelenbe, E., Lent, R., Montuoria, A., Xu, Z.: Cognitive Packet Network: QoS and Performance. In: 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02), Fort Worth, Texas, USA (2002)
6. Atif, K., Plateau, B.: Stochastic Automata Networks for modelling parallel systems. IEEE Transactions on Software Engineering **17** (1991) 1093–1108
7. Fernandes, P., Plateau, B., Stewart, W.J.: Efficient descriptor - Vector multiplication in Stochastic Automata Networks. Journal of the ACM **45** (1998) 381–414
8. Benoit, A., Brenner, L., Fernandes, P., Plateau, B., Stewart, W.J.: The PEPS Software Tool. In: Proceedings of the 13^{th} International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Urbana and Monticello, Illinois, USA (2003)
9. van Gemund, A.J.C.: Symbolic Performance Modeling of Parallel Systems. IEEE Transactions on Parallel and Distributed Systems **14** (2003) 154–165
10. Menascé, D.A., Almeida, V.A.F.: Capacity planning for web services: metrics, models, and methods. Prentice Hall (2002)