2011 IEEE International Parallel & Distributed Processing Symposium

# Exploiting multi-core architectures in clusters for enhancing the performance of the parallel Bootstrap simulation algorithm

César A. F. De Rose, Paulo Fernandes, Antonio M. Lima, Afonso Sales, and Thais Webber

*Pontifícia Universidade Católica do Rio Grande do Sul*
*Av. Ipiranga, 6681 – Prédio 32 – 90619-900 – Porto Alegre, Brazil*
{*cesar.derose, paulo.fernandes, antonio.lima, afonso.sales, thais.webber*}*@pucrs.br*

***Abstract*—The solution of Markovian models is usually non-trivial to be performed using iterative methods, so it is well-fitted to simulation approaches and high performance implementations. The Bootstrap simulation method is a novel simulation technique of Markovian models that brings a considerable improvement in the results accuracy, notwithstanding its higher computation cost when compared to other simulation alternatives. In this paper, we present three parallel implementations of the Bootstrap simulation algorithm, exploiting a multi-core SMP cluster. We discuss some practical implementation issues about processing and communication demands, as well as present an analysis of speedup and efficiency considering different models' sizes and simulation trajectory lengths. Finally, future works point out some improvements to achieve even better results in terms of accuracy.**

***Keywords*-Markovian models; Discrete-event simulation; Statistical techniques; Parallel algorithms; Multi-core SMP cluster; Performance evaluation;**

## I. Introduction

Markovian models are very useful in many domains (*e.g.*, bioinformatics, economics, engineering, among others) to describe complex system interactions using mathematical methods for solving the linear systems of equations [1]. The results of these models provide quantitative performance indices that can be used to indicate bottlenecks/configurations where the system will degrade or malfunction [2], [3], [4], [5], [6].

However, due to a large amount of possible configurations of those models, an efficient numerical solution becomes intractable and easily dependable on the available computational resources [1]. Iterative solutions (*e.g.*, SOR [7] or Power Method [1]) are usually bounded by memory and computational power and, hence, simulation techniques become an alternative to the solution of Markovian models.

In Markovian simulation contexts, an elementary goal is to generate independent samples for latter statistical analysis, which means to collect observed states in a chain or network of chains, *e.g.*, Stochastic Automata Networks (SAN) [8]. The main idea is to perform a *random walking* given the set of possible states that the system assumes and, from an initial state, jump to another state, if a transition is defined. This process defines a *simulation trajectory* and the user must choose the way to compute the state probability distribution, accumulating the number of times each state is visited given a trajectory length. Many examples use the generation of discrete events combining them with non-trivial data structures that save important information regarding the simulation execution [9], [10], [11], [12].

It is common to run the simulation for long trajectories with the purpose of accuracy improvement, since the precision is directly related to the number of samples that was produced. Many improvements have been introduced in different simulation techniques, such as Monte Carlo [13], Perfect sampling [14] and even in traditional simulation [15], in order to achieve even more precise results. Indeed, there are several advances concerning the parallel sampling technique for Markovian models. Recently, the Bootstrap simulation demonstrates remarkable results to improve accuracy [16], however, its usage is impaired due to high computational costs to produce repeated batches of samples. A first parallel implementation of this novel technique has been proposed [17], but there is room for improvements and broader discussions.

Our main contribution is to introduce faster parallel implementations of the Bootstrap simulation. In order to exploit parallelism in a multi-core SMP cluster [18], [19], we use pure MPI and also a hybrid programming model with MPI and OpenMP. Moreover, we present broader discussions about performance issues of the parallel Bootstrap implementations, focusing our attention on the simulation results accuracy.

The remainder of this paper is presented as follows. Section II presents the Bootstrap simulation technique applied in the context of Markovian simulation, discussing about its accuracy and processing time. Section III describes three parallel implementations of the Bootstrap simulation algorithm. In Section IV, we present three Markovian models used to measure the parallel implementations' performance and show the results for different simulation trajectory

Authors receive grants from Petrobras (0050.0048664.09.9). Paulo Fernandes is also funded by CNPq-Brazil (PQ 307272/2007-9). Afonso Sales receives grants from CAPES-Brazil (PNPD 02388/09-0). The order of authors is merely alphabetical.

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.297

lengths and model's sizes. Finally, the conclusion points out some parallelization issues and future works draws the possibility of blending the Bootstrap technique with other sophisticated simulation methods.

## II. Bootstrap Method Application

*Bootstrap* is a well known statistical technique [20] applied to many fields to improve accuracy when performing sample estimations for complex distributions [21]. The method's objective is to discover the distribution in random samplings $\Lambda$ of size $n$ directly extracted from the population $\tilde{\Lambda}$, *i.e.*, $\Lambda \subset \tilde{\Lambda}$. When observing $\Lambda$, the $n$ samples are drawn with probability $\frac{1}{n}$ and that is proved to represent the unknown population as closely as possible. That concept helps improving the methods' accuracy since the main feature of using Bootstrap is to work with sample replacement, *i.e.*, the obtained values could be repeated throughout the process.

Figure 1 presents the Bootstrap method applied to the Markovian models simulation context. For more detail of this new simulation technique using the Bootstrap method, we suggest the reader to consult [16].

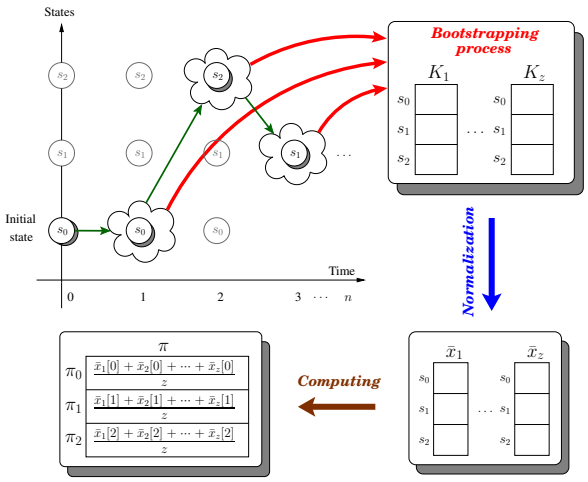


Figure 1. Bootstrap simulation method schema.

Algorithm 1 presents the procedure of the Bootstrap simulation method depicted in Figure 1. In this algorithm, variables and vectors are initialized at lines 1-4. We denote $S$ as the set of model states, where $|S|$ is the cardinality of this set, *i.e.*, the model's size. The random walking that traces the simulation trajectory of length $n$ is performed at lines 5-15. The next-state $s_{new}$ on the trajectory is calculated from state $s$ at line 6, using a pseudorandom generator $U$ uniformly distributed in $(0..1)$ by a transition function $\phi$. The core of the bootstrapping process, which is done by counting repeated batches of samples in the $z$ bootstraps' vectors $K$, is performed at lines 7-13. Those vectors are normalized at lines 16-24 and stored in $\bar{x}$. At last (lines 25-30), the average probabilities of every state, which are computed according the vectors $\bar{x}$, are stored in $\pi$.

Note that $z \times \bar{n}$ samplings take place for every trajectory step, comparing each pseudorandom value against an arbitrarily chosen constant value $\alpha$ between 0 and $\bar{n}-1$. If the pseudorandom generated value is equal to $\alpha$ (line 9), the state is counted in the correspondent bootstrap $K$ (line 10). Previous work [16] demonstrates that it is possible to only perform $\bar{n}$ trials instead of $n$, where $\bar{n} \ll n$.

**Algorithm 1** Bootstrap simulation method

1: $\alpha \leftarrow U(0..\bar{n}-1)$ {*choose a constant value* $\alpha$ *initialization*}
2: $\pi \leftarrow 0$ {*initialization of the probability vector* $\pi$ *of size* $|S|$}
3: $K \leftarrow 0$ {*initialization of all* $z$ *bootstraps* $K$}
4: $s \leftarrow s_0$ {*set state* $s$ *as initial state* $s_0$}
5: **for** $t = 1$ to $n$ **do**
6: $s_{new} \leftarrow \phi(s, U(0..1))$ {*computes* $s_{new}$ *from* $s$ *according to* $U(0..1)$}
7: **for** $b = 1$ to $z$ **do**
8: **for** $c = 1$ to $\bar{n}$ **do**
9: **if** $(U(0..\bar{n}-1) == \alpha)$ **then**
10: $K_b[s_{new}] \leftarrow K_b[s_{new}] + 1$ {*counts a sample in the bootstrap*}
11: **end if**
12: **end for**
13: **end for**
14: $s \leftarrow s_{new}$ {*current state* $s$ *is updated to* $s_{new}$}
15: **end for**
16: **for** $b = 1$ to $z$ **do**
17: $\omega \leftarrow 0$
18: **for** $i = 1$ to $|S|$ **do**
19: $\omega \leftarrow \omega + K_b[i]$ {*accumulates in* $\omega$ *the* $K_b$ *values*}
20: **end for**
21: **for** $i = 1$ to $|S|$ **do**
22: $\bar{x}_b[i] \leftarrow \frac{K_b[i]}{\omega}$ {*normalizes the probability of* $i$*-th state*}
23: **end for**
24: **end for**
25: **for** $i = 1$ to $|S|$ **do**
26: **for** $b = 1$ to $z$ **do**
27: $\pi[i] \leftarrow \pi[i] + \bar{x}_b[i]$
28: **end for**
29: $\pi[i] \leftarrow \frac{\pi[i]}{z}$ {*computes the final average probabilities*}
30: **end for**

Figure 2 presents the simulation times (depicted by the bars) and the obtained maximum absolute errors (plotted by dotted lines) for three Markovian models: *ASP*, *FAS* and *RS*, which are described in detail in Section IV-A. For these models, we have performed the Bootstrap simulation method varying the trajectory lengths from 1e+05 to 1e+09.

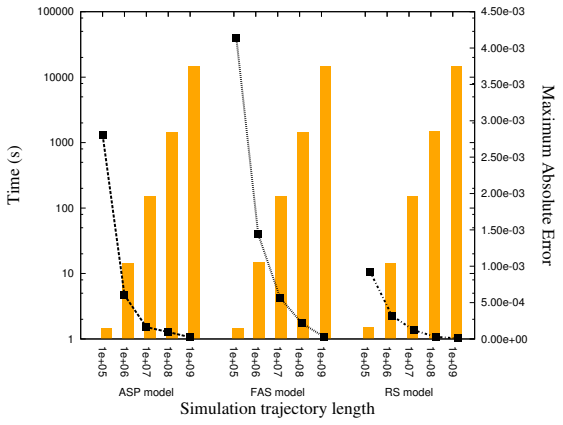


Figure 2. Simulation times *vs.* Maximum Absolute Errors.

Observing the results presented in Figure 2, it is evident that the simulation of long trajectories (*i.e.*, 1e+09) using the Bootstrap method can directly impact on the results accuracy, since they produce maximum absolute errors inferior to 1e-05 . However, the execution of those long trajectories demands a considerable simulation time.

As the bootstrap vectors can be independently computed, one can devise parallel algorithms, focusing on parallel resamplings to enhance the execution time of long run trajectories. Once the samples can be generated in a fast manner, the number of samples could also be augmented in order to improve results accuracy.

## III. Parallel Bootstrap implementations

Considering that previous performance analysis of the sequential simulation process have pointed out a need of optimization in bootstrapping process (Figure 2), we present three parallel implementations for multi-core SMP clusters in order to achieve better performance results. These implementations differ in the workload distribution and programming model.

We have started with a pure MPI implementation (*pure-MPI*) to profit of different machines, measuring the impact of communication [17]. The second approach named *Hybrid-I* was implemented using a hybrid programming model with MPI and OpenMP. Due to thread management overheads found in the *Hybrid-I* implementation, we have developed an optimized approach, named *Hybrid-II*.

### A. Pure MPI implementation (pure-MPI)

This approach uses only MPI primitives and aims splitting the $z$ bootstraps among a given number $C$ of cluster nodes. The implementation also uses a static scheduling strategy based on a fairly workload distribution (Figure 3).

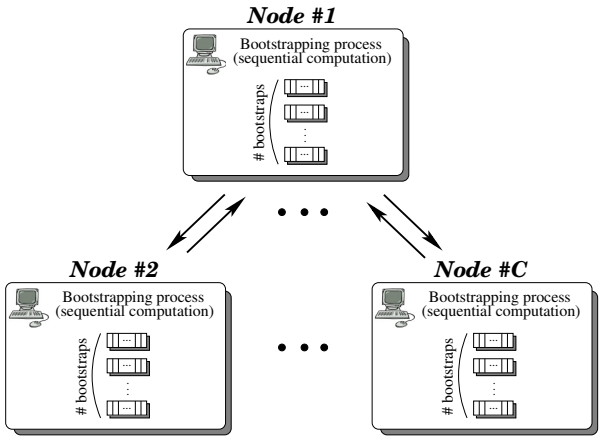


Figure 3. *Pure-MPI* parallel approach schema.

In this approach, each node performs the bootstrapping process for the number of bootstraps assigned until the end of the trajectory length. At the end of the computation, the results stored in each individual probability vector need to be synchronized with the first node, which is responsible for computing the final probability vector. The synchronization process is accomplished via the *MPI_REDUCE* routine that performs a global update of the probability vector. Note that we have used only one MPI process for each node, since our application demands a high memory cost *per* process. The use of more than one MPI process *per* node in our application can exceed memory bounds and a hybrid MPI/OpenMP implementation is more suitable.

### B. Hybrid MPI/OpenMP implementation (Hybrid-I)

An advantage of the Bootstrap simulation method is that the steps related to the bootstrapping process are completely independent, so beyond a parallel approach that equally distributes the workload among nodes, one can take advantage of the possible intra-node parallelism (Figure 4).

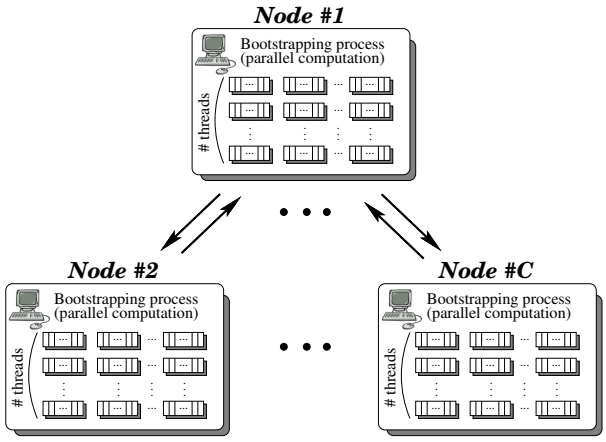


Figure 4. *Hybrid-I* parallel approach schema.

In this approach, named *Hybrid-I*, the bootstrapping process in each node is also parallelized using the combined parallel work-sharing loop construct *#pragma omp parallel for* from OpenMP [22]. This construct is used to parallelize the loop at line 7 (Algorithm 1), following a static scheduling strategy defined via the *static* schedule clause. Therefore, the intra-node computation is distributed among a given number of threads on each node.

### C. Enhancing the Hybrid-I approach (Hybrid-II)

This approach uses the same parallel schema presented in Figure 4, taking also advantage of the possible intra-node parallelism. However, instead of create a parallel region at each step in the simulation process (*Hybrid-I* approach), *Hybrid-II* implementation creates only once the parallel region. Therefore, a parallel region integrates the whole simulation process (Algorithm 1, lines 5-15), minimizing possible overheads related to the thread management. The workload distribution is performed splitting the number of bootstraps assigned for each node among threads.

## IV. Parallel performance evaluation

The Bootstrap simulation method presents a computational cost related to the trajectory length ($n$), the number of bootstraps ($z$) and the number of trials ($\bar{n}$) in the resampling process. However, some Markovian models present different characteristics (*e.g.*, large state spaces, sparse transition matrices, among others) that could impact on the method's performance, depending also on the parallel approach. In order to verify the trade-offs of using both MPI and OpenMP based solutions, we have executed simulations for three classes of models [23], [9], [24], varying the models' sizes (*i.e.*, very small models and considerably large models) for a more comprehensive analysis.

Following, we present models originally described by the Stochastic Automata Networks (SAN) formalism [8], which is a structured Markovian formalism [25], *i.e.*, they could have a large underlying Markov chain depending on the number of automata and their complex transitions.

We have performed experiments in a multi-core SMP cluster composed of eight homogeneous machines interconnected via a Gigabit Ethernet network. Each machine consists of two Intel Xeon E5520 (Nehalem) Quad-core processors with the Intel Hyper-Threading technology (16 logical processors) and 16 GB of memory. Each processor runs at 2.27 GHz and 8 MB L3 shared by all cores. The software stack is a Linux O.S. with the libraries OpenMPI 1.4.2 and OpenMP 2.5.

### A. The simulated Markovian models

**ASP (Alternate Service Pattern) model** describes an Open Queueing Network [1] with servers that map different service patterns. The model has four queues represented by four automata, an additional automaton representing the service patterns. This model has reachable state space (RSS) given by $(K+1)^4 \times P$ states, where $K$ is the capacity of the queues and $P$ the number of service patterns.

**FAS (First Available Server) model** indicates the availability of $N$ servers, where every server is composed of a two state automaton, representing the two possible server conditions: *available* or *busy*. In the model, requests are firstly assigned to the first server. If the server is busy, the task must be assigned to the second server and so on, *i.e.*, the first available server is assigned to the request. This model has RSS equal to $2^N$ states.

**RS (Resource Sharing) model** maps $R$ shared resources to $P$ processes. Each process is represented by an automaton with two states: *idle* or *busy*. The number of available resources is represented by a function that only grants access to the *busy* state if there is less than $R$ process in the *busy* state. This model represents $2^P$ states and, the number of reachable states is just $\sum_{i=0}^{R} \binom{P}{i}$, where $\binom{P}{i}$ is the number of $i$-combination of a $P$-sized set, *i.e.*: $\binom{P}{i} = \frac{P!}{i!(P-i)!}$.

The set of small models was parametrized as follows: *ASP model* - every queue with capacity two ($K = 2$) and two service patterns ($P = 2$); *FAS model* - with nine servers ($N = 9$); and *RS model* - with 10 processes ($P = 10$) and five resources ($R = 5$), giving respectively models with reachable state space sized 162, 512 and 638 states. And for large models: *ASP model* - every queue with capacity fifty ($K = 50$) and four service patterns ($P = 4$); *FAS model* - with twenty five servers ($N = 25$); and *RS model* - with 25 processes ($P = 25$) and fifteen resources ($R = 15$), giving respectively models with reachable state space sized around, respectively, 27, 33 and 29 million states.

### B. Experimental results

We have performed a set of experiments[1] to verify the performance of the parallel implementations, running the Bootstrap simulation for our models (Section IV-A) over 1 (sequential implementation), 2, 3, 4, 5, 6, 7, and 8 nodes. The experiments were conducted considering 36 bootstraps ($z$ = 36) and trajectory lengths ($n$) of 1e+06, 1e+07, 1e+08, and 1e+09. Previous work [16] has analyzed the impact of the number of bootstraps on the results precision considering the aforementioned Markovian models. We have assumed 36 bootstraps in our experiments in order to obtain a fair workload distribution among nodes. Table I describes how the bootstraps were distributed among the nodes.

Table I
NUMBER OF BOOTSTRAPS ASSIGNED TO EACH CONFIGURATION.

| Configuration | Number of bootstraps in each node | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ | $5^{th}$ | $6^{th}$ | $7^{th}$ | $8^{th}$ |
| **1** | 36 | | | | | | | |
| **2** | 18 | 18 | | | | | | |
| **3** | 12 | 12 | 12 | | | | | |
| **4** | 9 | 9 | 9 | 9 | | | | |
| **5** | 8 | 7 | 7 | 7 | 7 | | | |
| **6** | 6 | 6 | 6 | 6 | 6 | 6 | | |
| **7** | 6 | 5 | 5 | 5 | 5 | 5 | 5 | |
| **8** | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 4 |

Figure 5 presents, respectively, the time spent in seconds for the simulation of large and small models. There are one chart to each trajectory length (from 1e+06 to 1e+09), where there are three sets of bars showing the results of each model (ASP, FAS and RS). At each set of bars representing a given model, there are eight bars representing the simulation times over each configuration. Each bar is split in five colors:

- *green* - concerns the process of discovering the next-state in the simulated trajectory (Algorithm 1, line 6);
- *red* - related to the bootstrapping process (lines 7-13);
- *blue* - related to the normalization of the bootstrap vectors (lines 16-24);
- *yellow* - concerns the time spent in communication among nodes;
- *brown* - related to the computation of the average state probabilities (lines 25-30).

[1]Our results were computed considering the average of 30 trials taking a 95% confidence intervals into account. Therefore, the time in seconds presented in $Y$-axis is statistically validated.

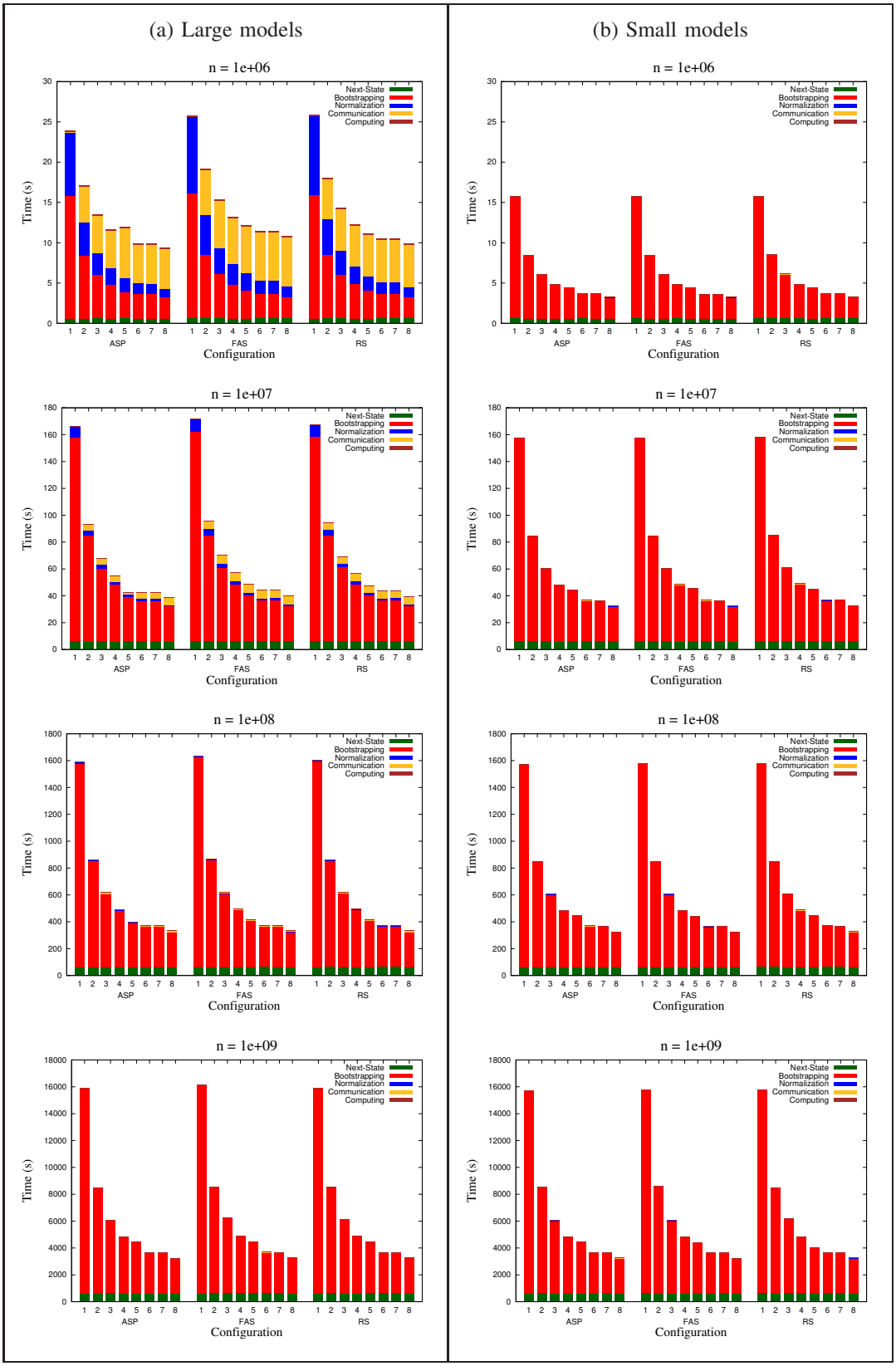


Figure 5. Parallel simulation performance analysis (*pure-MPI*) for large models (a) and small models (b).

Observing Figure 5 (a), we notice a deficient optimization regarding the parallel simulation times for the *pure-MPI* approach considering a trajectory length equal to 1e+06. These deficient parallel results for large models are due to the fact that increasing the number of nodes there is an increasing on the communication costs, where large vectors must be updated. Remark that increasing the trajectory length to 1e+07, the bootstrapping process time augments but the communication costs remain approximately the same. It is important to notice that the $Y$-axis increases one order of magnitude as the trajectory length augments. This observation is confirmed looking at the results for $n$=1e+09, where there are gains in terms of speedup, since the communication time is irrelevant in comparison to the bootstrapping process time. Moreover, the maximum speedup factor is limited to *eight*, since we have used only one MPI processes *per* node, as described in Section III-A.

Figure 5 (b) depicts the simulation performance for the small models. It is possible to perceive the major drives for the parallel implementation of the Bootstrap simulation. Small models have an irrelevant communication cost due to the size of the vectors that must be updated (around hundred of states for our experiments). On the contrary, large models need to update large probability vectors (around 30 million position).

Observing the processing time requirements for small and large models (Figure 5), we observe that those models have quite similar demands to the bootstrapping process. That fact indicates that the processing demand is almost bounded only by $n$, and it is independent of the model's size.

Figure 6 shows the simulation time in seconds for the ASP, FAS and RS models for $n$=1e+06, where 36 bootstraps were distributed among 1, 2, 4 and 8 threads. Surprisingly, the *Hybrid-I* results were extremely bad in comparison to *pure-MPI*.

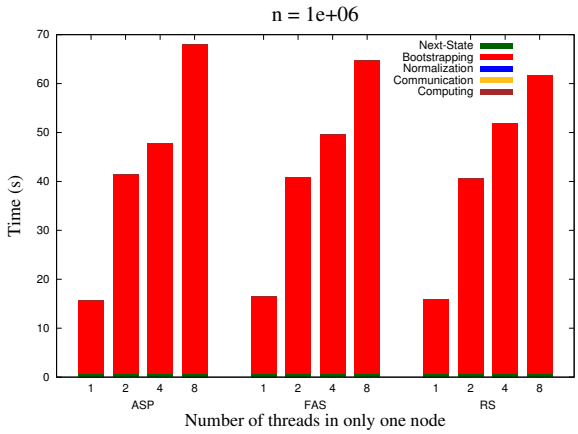


Figure 6. Parallel simulation performance analysis (*Hybrid-I*).

Observing Figure 6, it is possible to notice that the simulation time increases as the number of threads augments. For instance, the sequential time for $n$=1e+06 (ASP model - $K$=2; $P$=2) using *pure-MPI* - Figure 5 (b) - is around 15 seconds, whereas *Hybrid-I* with one node and 8 threads is about 70 seconds. The long execution time can be explained by the thread management overhead due to the parallel region is created and terminated $n$ times.

Figure 7 presents the simulation time results using *Hybrid-II*. At the $X$-axis, the number in parenthesis in each configuration means the maximum number of threads created in each node considering the assigned workload. It is also evident a deficient optimization concerning the simulation times for $n$=1e+06 for large models. This poor optimization is also due to the communication costs. These costs are less evident for $n$=1e+07 and even less for $n$=1e+09. In this approach, the bootstrapping process times for small and large models have also quite similar demands. Moreover, the occurrence of the flattening behavior in the parallel simulation times (configurations 5, 6, 7, and 8) is due to the similar workloads distributed for each node (*i.e.*, 36 bootstraps fairly distributed among a given number of nodes). This observation is confirmed by Figure 7, where the bootstrapping process times for all three models are really close to each other.

Table II presents the achieved speedups for the ASP, FAS and RS models, where $n$ varies from 1e+06 to 1e+09 using the *pure-MPI* approach (configuration 8) and *Hybrid-II* (configurations 4, 5 and 8 with, respectively, a maximum of 9, 8 and 5 threads *per* node). Remark that the speedups are inferior to 5 times for *pure-MPI* for all models and different trajectory lengths. However, *Hybrid-II* has significantly improved the speedups to more than 24 times for small models - Table II (a). But, configuration 4 (9) and 5 (8) have reached about 80% efficiency, whereas configuration 8 (5) has only reached around 50%. There is a considerable improvement in terms of speedup between the first (*pure-MPI*) and the last (*Hybrid-II*) parallel approach.

Similar results are obtained to large models using *pure-MPI* - Table II (b) - where the speedups are also inferior to 5 times. For *Hybrid-II*, the speedups for $n$=1e+06 are not significant either, since the communication times are evident regarding the total simulation time. Nevertheless, for long run trajectories ($n$=1e+09), besides these trajectories provide small maximum absolute errors (*i.e.*, inferior to 1e-05), those experiments for configuration 5 (8) have achieved the best speedups (around 35 times, which are really close to the ideal in this case) and really good efficiency, nearly, 90%.

Figure 8 presents the speedups (plotted by dotted lines) and efficiencies (depicted by the bars) for small and large models (for long run trajectories, $n$ =1e+09). It is more evident the fact that the speedup between configurations 5 and 8 remains the same, however the efficiency considerably drops. Observe that the efficiency for large models (configuration 3) is superior to 100% due to the Hyper-Threading technology.

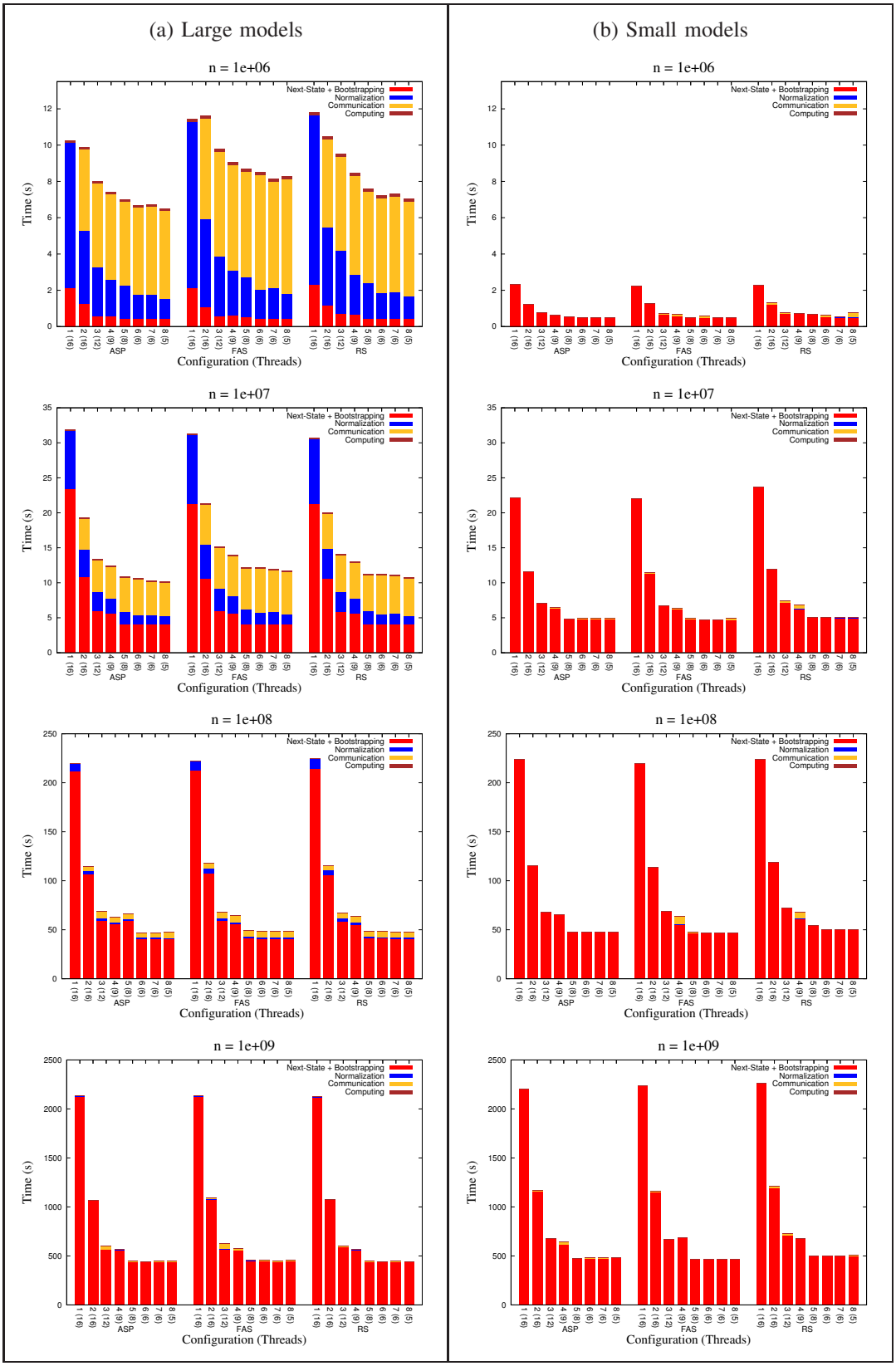


Figure 7. Parallel simulation performance analysis (*Hybrid-II*) for large models (a) and small models (b).

Table II
PARALLEL SIMULATION PERFORMANCE ANALYSIS.

| Model | $n$ | (a) Small models | | | | | | | | | (b) Large models | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Sequential | pure-MPI | | Hybrid-II | | | | | | Sequential | pure-MPI | | Hybrid-II | | | | | |
| | | | configuration 8 | | configuration 4 (9) | | configuration 5 (8) | | configuration 8 (5) | | | configuration 8 | | configuration 4 (9) | | configuration 5 (8) | | configuration 8 (5) | |
| | | Time (s) | Speedup | Effic. (%) | Speedup | Effic. (%) | Speedup | Effic. (%) | Speedup | Effic. (%) | Time (s) | Speedup | Effic. (%) | Speedup | Effic. (%) | Speedup | Effic. (%) | Speedup | Effic. (%) |
| ASP | 1e+06 | 15.73 | 4.85 | 7.59 | 24.58 | 76.81 | 30.25 | 75.63 | 30.84 | 48.19 | 23.86 | 2.57 | 4.02 | 3.22 | 10.05 | 3.41 | 8.52 | 3.67 | 5.74 |
| | 1e+07 | 157.23 | 4.84 | 7.57 | 24.30 | 75.94 | 32.96 | 82.41 | 31.76 | 49.63 | 165.70 | 4.34 | 6.78 | 13.34 | 41.69 | 15.22 | 38.04 | 16.33 | 25.51 |
| | 1e+08 | 1,573.43 | 4.84 | 7.56 | 24.23 | 75.72 | 32.91 | 82.28 | 32.54 | 50.85 | 1,585.07 | 4.83 | 7.54 | 25.24 | 78.87 | 23.99 | 59.97 | 33.57 | 52.46 |
| | 1e+09 | 15,727.80 | 4.84 | 7.56 | 24.39 | 76.21 | 32.97 | 82.43 | 32.66 | 51.03 | 15,845.82 | 4.95 | 7.73 | 28.01 | 87.54 | **35.50** | **88.76** | 35.51 | 55.49 |
| FAS | 1e+06 | 15.76 | 4.85 | 7.58 | 23.52 | 73.51 | 31.52 | 78.80 | 31.52 | 49.25 | 25.78 | 2.40 | 3.74 | 2.85 | 8.91 | 2.96 | 7.40 | 3.11 | 4.86 |
| | 1e+07 | 157.66 | 4.86 | 7.59 | 24.91 | 77.83 | 31.91 | 79.79 | 31.79 | 49.67 | 171.52 | 4.31 | 6.74 | 12.28 | 38.37 | 14.12 | 35.29 | 14.66 | 22.91 |
| | 1e+08 | 1,575.08 | 4.86 | 7.59 | 24.75 | 77.36 | 33.18 | 82.95 | 33.63 | 52.54 | 1,628.67 | 4.94 | 7.71 | 25.12 | 78.49 | 32.78 | 81.96 | 33.66 | 52.60 |
| | 1e+09 | 15,768.10 | 4.87 | 7.60 | 22.92 | 71.62 | 33.79 | 84.47 | 33.90 | 52.96 | 16,132.05 | 4.99 | 7.80 | 28.18 | 88.07 | **35.30** | **88.25** | 35.44 | 55.38 |
| RS | 1e+06 | 15.78 | 4.81 | 7.52 | 22.23 | 69.45 | 23.21 | 58.01 | 20.23 | 31.61 | 25.80 | 2.62 | 4.09 | 3.05 | 9.53 | 3.40 | 8.51 | 3.66 | 5.73 |
| | 1e+07 | 157.84 | 4.82 | 7.54 | 23.08 | 72.11 | 30.89 | 77.22 | 31.13 | 48.64 | 166.89 | 4.30 | 6.72 | 12.85 | 40.15 | 14.80 | 36.99 | 15.54 | 24.28 |
| | 1e+08 | 1,579.74 | 4.83 | 7.55 | 23.27 | 72.73 | 29.14 | 72.85 | 31.15 | 48.67 | 1,595.79 | 4.89 | 7.64 | 25.07 | 78.34 | 32.98 | 82.44 | 33.45 | 52.27 |
| | 1e+09 | 15,788.10 | 4.83 | 7.54 | 23.31 | 72.86 | 31.44 | 78.59 | 31.20 | 48.76 | 15,812.67 | 4.90 | 7.65 | 27.94 | 87.30 | **35.13** | **87.83** | 35.70 | 55.78 |

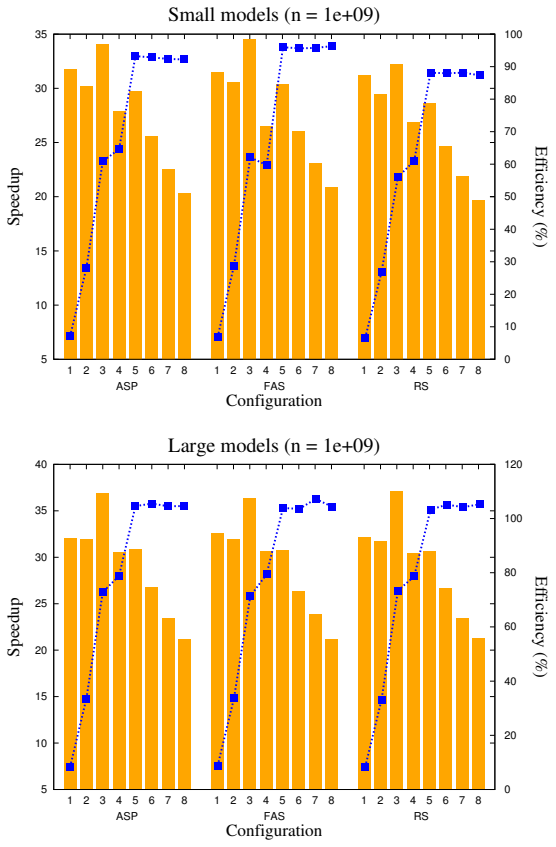


Figure 8. Speedup *vs.* Efficiency (*Hybrid-II*).

Remark that the computational problem of simulating Markovian models is mainly related to the large amount of samples needed to obtain an accurate probability distribution among states, *i.e.*, long run trajectories are recommended. Additionally, using the Bootstrap method, we have included an overhead of computations to generate resamplings at each step of the simulated trajectory. Regarding all performed experiments, *Hybrid-II* have demonstrated to be a good option to speedup the Bootstrap simulation algorithm, maintaining results accuracy using long run trajectories. Observing the results we identify that the challenge to achieve best efficiency is to guarantee an adequate load balance in a minimum communication overhead. In our experiments we are aware that the number of bootstraps can directly influence in the overall efficiency, which is also dependent on the number of nodes/processors and the correspondent distributed workload.

## V. CONCLUSION

The execution of a process where independent tasks are distributed among several processors lead us a better performance than when the pure sequential version is performed. Nevertheless, the Bootstrap simulation method allows the generation of samples in an independent manner which is trivial in terms of parallelization efforts. We have noticed that the computational cost issues, and related open challenges to face these classes of problems, are specifically directed to two concerns:

- an efficient implementation of *transition function* $\phi$ to compute samples in simulation trajectories;
- adapting parallel sampling techniques to mitigate the efforts related to the simulation of structured Markovian models.

An important contribution of this paper is to propose an efficient parallel implementation of a novel simulation algorithm, achieving a considerable speedup for very large models (*i.e.*, tens of millions states), specially when quite long run trajectories were needed (*i.e.*, billions of transitions). The speedups were also consistent for different models, delivering a nearly 5 times speedup for the configuration 8 using the *pure-MPI* approach, and about 30 times for configurations 5 and 8 using the *Hybrid-II* approach. In addition, our contribution consists in the comprehension that *the computational demands of the bootstrapping process depend only on the simulation trajectory length*, while *the communication demands depend only on the model's size*. That fact is confirmed in both parallel approaches *pure-MPI* and *Hybrid-II*, allowing a better choice of configuration according to the simulation to be performed.

The focus of this research is on the parallel performance analysis of the Bootstrap simulation algorithm, exploiting different characteristics of multi-core SMP clusters. But it is also important to notice the obtained gains related to the simulation times needed to perform long run trajectories, which allow us to achieve more accurate results (where the maximum absolute errors are inferior than 1e-05).

Previous works [16] have analyzed the usage of tens of bootstraps in the bootstrapping process over one single node using only one processor. However, multi-core cluster environments allow us to perform the simulation of a model using hundred of bootstraps distributed over many nodes with many processors. Regarding future works, a further study may be considered about the impact of the number of bootstraps on the simulation results accuracy.

Moreover, it is also possible to foresee improvements in the Bootstrap simulation method itself in order to consider the subsystems' state in a global dynamic, rather than inspecting directly the wide global state. We also foresee to incorporate the parallel sampling process with more sophisticated simulation approaches, such as *Perfect Simulation* [26].

Nonetheless, the results already obtained by the parallel approaches of the Bootstrap simulation algorithm allow quite important time savings in the solution of huge Markovian systems, which are impractical to solve with the current numerical software tools [27], [28]. That is an important fact that itself justifies the research effort involved in this work.

## REFERENCES


[1] W. J. Stewart, *Probability, Markov Chains, Queues, and Simulation*. USA: Princeton University Press, 2009.

[2] H. Wang, D. I. Laurenson, and J. Hillston, "Evaluation of RSVP and Mobility-Aware RSVP Using Performance Evaluation Process Algebra," in *Proceedings of the IEEE International Conference on Communications*, 2008, pp. 192–197.

[3] R. Marculescu and A. Nandi, "Probabilistic Application Modeling for System-Level Performance Analysis," in *Design Automation & Test in Europe (DATE)*, Munich, Germany, March 2001, pp. 572–579.

[4] R. Chanin, M. Corrêa, P. Fernandes, A. Sales, R. Scheer, and A. F. Zorzo, "Analytical Modeling for Operating System Schedulers on NUMA Systems," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 151, no. 3, pp. 131–149, 2006.

[5] C. Bertolini, L. Brenner, P. Fernandes, A. Sales, and A. F. Zorzo, "Structured Stochastic Modeling of Fault-Tolerant Systems," in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2004)*. Vollendam, The Netherlands: IEEE Computer Society, 2004, pp. 139–146.

[6] K. Sayre, "Improved techniques for software testing based on Markov chain usage models," Ph.D. dissertation, University of Tennessee, Knoxville, USA, 1999.

[7] G. H. Golub and C. F. V. Loan, *Matrix Computations*. The Johns Hopkins University Press, 1996.

[8] B. Plateau, "On the stochastic structure of parallelism and synchronization models for distributed algorithms," in *Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*. Austin, Texas: ACM Press, 1985, pp. 147–154.

[9] L. Brenner, P. Fernandes, and A. Sales, "The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations," *International Journal of Simulation: Systems, Science & Technology (IJSIM)*, vol. 6, no. 3-4, pp. 52–60, February 2005.

[10] W. H. Sanders and J. F. Meyer, "Reduced Base Model Construction Methods for Stochastic Activity Networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 1, pp. 25–36, 1991.

[11] J. Hillston, *A compositional approach to performance modelling*. New York, USA: Cambridge University Press, 1996.

[12] G. Balbo, "Introduction to Stochastic Petri Nets," in *European Educational Forum: School on Formal Methods and Performance Analysis*, Berg en Dal, The Netherlands, 2000, pp. 84–155.

[13] O. Häggström, *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[14] J. G. Propp and D. B. Wilson, "Exact Sampling with Coupled Markov Chains and Applications to Statistical Mechanics," *Random Structures and Algorithms*, vol. 9, no. 1–2, pp. 223–252, 1996.

[15] S. M. Ross, *Simulation*. Orlando, FL, USA: Academic Press, Inc., 2002.

[16] R. M. Czekster, P. Fernandes, A. Sales, D. Taschetto, and T. Webber, "Simulation of Markovian models using Bootstrap method," in *Summer Computer Simulation Conference (SCSC'10)*. Ottawa, Canada: ACM, July 2010, pp. 564–569.

[17] R. M. Czekster, P. Fernandes, A. Sales, and T. Webber, "Performance Issues for Parallel Implementations of Bootstrap Simulation Algorithm," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'10)*. Petropolis, Brazil: IEEE Computer Society, October 2010, pp. 167–174.

[18] N. Drosinos and N. Koziris, "Performance Comparison of Pure MPI vs Hybrid MPI-OpenMP Parallelization Models on SMP Clusters," *International Parallel and Distributed Processing Symposium (IPDPS 2004)*, vol. 1, p. 15a, 2004.

[19] Y. He and C. H. Q. Ding, "MPI and OpenMP paradigms on cluster of SMP architectures: the vacancy tracking algorithm for multi-dimensional array transposition," in *ACM/IEEE Supercomputing Conference*, 2002, pp. 1–14.

[20] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979.

[21] B. F. J. Manly, *Randomization, Bootstrap and Monte Carlo Methods in Biology*, 2nd ed. Chapman & Hall/CRC, 1997.

[22] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, 2007.

[23] A. Sales and B. Plateau, "Reachable State Space Generation for Structured Models which use Functional Transitions," in *International Conference on the Quantitative Evaluation of Systems (QEST'09)*, Budapest, Hungary, 2009, pp. 269–278.

[24] L. Brenner, P. Fernandes, and A. Sales, "Why you should care about *Generalized Tensor Algebra*," PUCRS, Porto Alegre, Tech. Rep. TR 037, 2003, http://www3.pucrs.br/pucrs/files/uni/poa/facin/pos/relatoriostec/tr037.pdf.

[25] P. Fernandes, B. Plateau, and W. J. Stewart, "Efficient descriptor-vector multiplication in stochastic automata networks," *Journal of the ACM (JACM)*, vol. 45, no. 3, pp. 381–414, May 1998.

[26] P. Fernandes, J.-M. Vincent, and T. Webber, "Perfect Simulation of Stochastic Automata Networks," in *International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'08)*, ser. LNCS, vol. 5055, 2008, pp. 249–263.

[27] R. M. Czekster, P. Fernandes, and T. Webber, "GTA express - A Software Package to Handle Kronecker Descriptors," in *Proceedings of the 6th International Conference on Quantitative Evaluation of SysTems (QEST 2009)*. IEEE Computer Society, September 2009, pp. 281–282.

[28] L. Brenner, P. Fernandes, B. Plateau, and I. Sbeity, "PEPS2007 - Stochastic Automata Networks Software Tool," in *International Conference on the Quantitative Evaluation of Systems (QEST'07)*. Edinburgh, UK: IEEE CS, 2007, pp. 163–164.