Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 187362, 7 pages
http://dx.doi.org/10.1155/2014/187362

Research Article

New Computing Technology in Reliability Engineering

Radim Briš and Simona Domesová

Technical University of Ostrava, 17. Listopadu 15/2172, Poruba, 708 33 Ostrava, Czech Republic

Correspondence should be addressed to Radim Briš; radim.bris@vsb.cz

Received 6 November 2014; Accepted 11 December 2014; Published 25 December 2014

Academic Editor: Yuji Liu

Reliability engineering is relatively new scientific discipline which develops in close connection with computers. Rapid development of computer technology recently requires adequate novelties of source codes and appropriate software. New parallel computing technology based on HPC (high performance computing) for availability calculation will be demonstrated in this paper. The technology is particularly effective in context with simulation methods; nevertheless, analytical methods are taken into account as well. In general, basic algorithms for reliability calculations must be appropriately modified and improved to achieve better computation efficiency. Parallel processing is executed by two ways, firstly by the use of the MATLAB function parfor and secondly by the use of the CUDA technology. The computation efficiency was significantly improved which is clearly demonstrated in numerical experiments performed on selected testing examples as well as on industrial example. Scalability graphs are used to demonstrate reduction of computation time caused by parallel computing.

1. Introduction

The concept of reliability has become recently a pervasive attribute worth of both qualitative and quantitative connotations. Quantitative treatment of the reliability of engineering systems and plants has led to the rise of reliability engineering as a scientific discipline [1]. Reliability is a fundamental attribute for the safe operation of any modern technological system.

System is a set of components that work together and form a functional unit. Real engineering systems include technical machines, production lines, computer networks, and other devices. In practice, it is often necessary to model, compute, and optimize reliability characteristics of such systems.

In engineering applications which involve availability modeling, we frequently face the following subproblem: the logical structure of the system as well as availability of each component at a given time $t \in \mathbb{R}$ is known and using this knowledge availability of the whole system at the time $t \in \mathbb{R}$ must be computed. In some cases, the time evolution of the availability is required. This problem was solved in the past by many different algorithms that can be roughly divided into analytical and simulation ones. In general, the simulation approach is employed when analytical techniques have failed to provide a satisfactory mathematical model. The principle behind the simulation approach is relatively simple and easy to apply. However, the common simulation techniques are slow and take a lot of time to provide accurate results. Nevertheless, this technique is the only practical method of carrying out reliability or risk studies, particularly when system is maintained and arbitrary failure and repair distributions are used or some special repair or maintenance strategy is prescribed.

The Monte Carlo (MC) method allows complex systems to be modeled without the need to make unrealistic simplifying assumptions, as is inevitably done when using analytical methods. With the increasing availability of fast computers, MC methods become more and more powerful and feasible [2]. Recent reliability analyses of complex systems based on the MC method bring very efficient estimators [3]. Finding of an appropriate estimator must be necessarily connected with a variance-reduction technique. There exist efficient techniques that provide significant reduction in variance, when they are correctly applied [4]. The application of these techniques gives other potentials for optimization of

the simulation algorithms solving complex real reliability problems. The feasibility of application to realistic cases stems from the possibility of evaluating the model in reasonable computing times, for example, by biasing techniques [5].

Another problem connected with the application of the MC method is slow convergence. To mitigate this issue, several techniques have been developed such as those based on the importance sampling and other classical methods [6]. Another possibility of improving the convergence is provided by the so-called conditioning methods which directly modify the modeling or the simulation procedure, as, for example, in the Petri net approach [7, 8]. In spite of all these advanced techniques, in most practical cases whenever reliability characteristics must be determined exactly and efficiently, the analyst faces computing problems, that is, slow convergence and large simulation time. That is because systems are more and more complex and demands on reliability are continually growing. For example, NEC producer [9] has offered system components for use in supercomputers in order to support the need for an extremely high performance. The improving reliability will be an essential feature of the products of the next generation.

The above mentioned computing problems are solvable by applying parallel processing of computing algorithms. This new computing technology is possible to put into practice just on assumption that an adequate computing infrastructure exists, which can be used for the HPC (high performance computing) calculations. In Technical University of Ostrava such a new supercomputing center is recently developed and built.

This paper is organized as follows. Section 2 brings simple method for system representation based on adjacency matrix. Basic methods (both analytical and simulation) for availability quantification derived from system state evaluation are introduced in Section 3. Section 4 demonstrates basic approach to the parallel computing of system reliability characteristics resulting from the MATLAB parfor loop on the one hand and from CUDA (Compute Unified Device Architecture) technology on the other hand. Both algorithms can be further optimized using bitwise operations. Both analytical and simulation methods are discussed in Section 5, which describes numerical experiments in context with computer efficiency related to the parallel computing technology. Section 6 describes availability quantification applied on industrial example from references. Section 7 brings conclusion.

2. System and Its Representation Using Adjacency Matrix

Consider a system consisting of n components, $n \in \mathbb{N}$. At a given time $t \in \mathbb{R}$ each component is in one of two disjoint states, operational or failed. To the ith component a state indicator b_i belongs. If the component is operational, $b_i = 1$; otherwise, $b_i = 0$. The vector $\mathbf{B} = (b_1, b_2, \ldots, b_n)$ is called the state vector of the system. The set of all possible state vectors is called the state space of the system and is denoted by V. Since components with just two possible states are considered, the state space has 2^n elements [10].

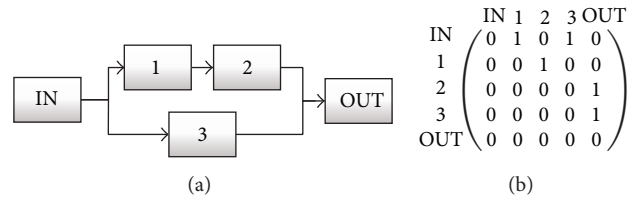

Figure 1: Block diagram and corresponding adjacency matrix.

The system state is determined by the states of its components. The function $S(\mathbf{B}) : V \to \{0, 1\}$ defined as

$$S(\mathbf{B}) = \begin{cases} 1 & \text{if system is operational,} \\ 0 & \text{if system is failed} \end{cases} \quad (1)$$

is called the system function.

The logical structure of systems is often represented using block diagrams. A block diagram is a directed graph with two special vertices (in Figure 1 marked IN and OUT); the remaining vertices symbolize components of the system. A directed edge connecting two vertices indicates that there is a direct path between corresponding system elements in the indicated direction. Le us say that the system is operational if there exists a path between the IN element and the OUT element passing only through functional components. For the system function, the following

$$S(\mathbf{B}) = \begin{cases} 1 & \text{if system is operational,} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

holds.

For implementation purposes, it is suitable to represent the block diagram equivalently using the adjacency matrix M of size $(n + 2) \times (n + 2)$. The first row and column of the matrix are used for the IN element; the last row and column are used for the OUT element. The remaining rows and columns represent the components of the system. If a directed edge between elements i and j exists, $(M)_{ij} = 1$; otherwise, $(M)_{ij} = 0$.

3. Methods of Availability Quantification

As mentioned above, the following problem is solved in this paper. "A system with n independent components and an appropriate system function $S(\mathbf{B})$ is given. Further availabilities of all components are known. It is necessary to quantify the availability of the system."

The state space of the given system includes 2^n state vectors. By substituting each state vector into the system function, the state space can be divided into two disjoint subsets $V_0 = \{\mathbf{B} \in V \mid S(\mathbf{B}) = 0\}$ and $V_1 = V \setminus V_0$. Let us denote the availability of the ith component by p_i. Further denote

$$\beta_i = \begin{cases} p_i & \text{if } b_i = 1, \\ 1 - p_i & \text{if } b_i = 0. \end{cases} \quad (3)$$

Therefore, the probability $P(\mathbf{B})$ that the system is in state defined by a specific state vector $\mathbf{B} = (b_1, b_2, \ldots, b_n)$ can be calculated as

$$P(\mathbf{B}) = \prod_{i=1}^{n} \beta_i. \quad (4)$$

At this point both solving methods can be presented, the analytical one and the simulation one.

3.1. Analytical Method. Availability A is the probability that the system is operational at a considered time, that is, that it is in a state belonging to V_1 in the time. This probability can be calculated as the sum of probabilities $P(\mathbf{B}_j)$ that the system is in state $\mathbf{B}_j \in V_1$. Therefore, the availability A can be expressed as

$$A = P(S(\mathbf{B}) = 1) = \sum_{j=1, \mathbf{B}_j \in V_1}^{2^n} P(\mathbf{B}_j) = \sum_{j=1, \mathbf{B}_j \in V_1}^{2^n} \prod_{i=1}^{n} \beta_i^j. \quad (5)$$

Using this formula, the algorithm for the analytical computation of system availability can be easily constructed. Notice that each state vector of the system is evaluated, so it can be expected that the computation time increases exponentially with respect to the number of components.

3.2. Simulation Method. One of the ways to avoid evaluating the state of the system for each state vector is to solve the problem using the Monte Carlo method. The problem is simulated using appropriate random variable X. Here the following random variable is chosen:

$$X = \begin{cases} 1 & \text{if } S(\mathbf{B}_j) = 1, \\ 0 & \text{if } S(\mathbf{B}_j) = 0. \end{cases} \quad (6)$$

It should be verified that the mean value of X equals the availability A. For the mean value holds true the following formula:

$$E(X) = \sum_{j=1}^{2^n} X_j \cdot P(\mathbf{B}_j) = \sum_{j=1, \mathbf{B}_j \in V_1}^{2^n} P(\mathbf{B}_j) = A. \quad (7)$$

That is, the random variable X is selected properly.

The experiment consists of generating a random state vector and determining the value of X_j. The vectors are generated according to following rule: the probability that a specific state vector $\mathbf{B}_j$ is generated equals the probability $P(\mathbf{B}_j)$. After N repetitions of this experiment a random sample $X_1, X_2, \ldots, X_N$ is obtained. The sample mean $\overline{X}$ of this random sample estimates the mean value of the variable X and therefore the availability A.

3.3. System State Evaluation. The most important part of both approaches from computation point of view is determination of the system state $S(\mathbf{B})$. It is an algorithm which for a system given by an adjacency matrix and for a given state vector $\mathbf{B}$ determines the state of the system, that is, if the system is operational. The algorithm consists of two primary steps:

(i) removal of failed components,

(ii) deciding if the system is operational,

and is defined as follows.

(1) Remove rows and columns that belong to failed components from the adjacency matrix.

(2) Create three empty sets ACTUAL, AVAILABLE, and VISITED. Move system elements that are directly connected to the IN element to the AVAILABLE set.

(3) Move elements from AVAILABLE to ACTUAL. Move elements that are directly connected to some of the elements in ACTUAL and that are not in VISITED to AVAILABLE.

(4) If AVAILABLE contains the OUT element, terminate the algorithm; system is operational.

(5) If AVAILABLE is empty, terminate the algorithm; system is not operational.

(6) Move elements from ACTUAL to VISITED. Continue to step (3).

Note. In fact the word "move" in the steps above means relocation of components, that is, deletion from original set and saving in a new set. If some components already exist in the target set within the process, they remain there.

Considering the estimation of the computation time, notice that steps (3) to (6) are executed at most m times, where m is the number of operational components of the system, $m \leq n$. This means that the state of the system is calculated in n iterations at most.

4. Parallelization

Both analytical and simulation algorithms described in the previous section were implemented in the MATLAB language, at first as a sequential application and then both algorithms were parallelized. The principle of the parallelization consists in dividing the task into multiple simultaneous threads, each of them executing a different part of the calculation. The aim of the process is to optimize the algorithms on the one hand and reduce their computing time on the other hand.

The Monte Carlo method is suitable for parallelization, because the experiments are independent and may be realized in different threads. Assume that $M \cdot r$ experiments should be executed in r threads. This means that in the kth thread M random state vectors are generated and for each of them the system state is evaluated. The output of the thread is the total number of generated vectors for which the system was operational, denoted by S_k. The estimated availability can be easily obtained from the intermediate results as follows:

$$\overline{X} = \frac{1}{M \cdot r} \sum_{k=1}^{r} S_k. \quad (8)$$

Similarly, in case of the analytical algorithm, the state space can be divided into several parts and the state vectors are evaluated in different threads.

4.1. The Parfor Loop. MATLAB programming tool allows a simple parallelization of loops with fixed number of recurrences using the "parfor" keyword. In case of the Monte Carlo method, the loop over all the sample size is parallelized and in case of the analytical algorithm it is the loop over the state space. Number of threads in which the calculation is executed equals the number of used processor cores at most.

4.2. The CUDA Technology. These algorithms can be efficiently parallelized using CUDA (Compute Unified Device Architecture). This technology allows using the NVIDIA graphics cards for parallel computing. The threads are organized in blocks and the blocks are organized in a grid (CUDA C Programming Guide 2013). In case the graphics card is used in this simulation, there can be at most 1024 threads per block and 65535×65535 blocks per grid. Furthermore, the maximum amount of available memory cannot be exceeded.

In each thread the so-called "CUDA kernel" is executed. This special function is written in a file with the ".cu" extension; syntax is similar to the C++ language. There is a possibility of working with these files directly within the MATLAB environment. But first creating a ".ptx" file (Parallel Thread Execution) using external compiler (NVIDIA CUDA Compiler) is needed. From these two files the kernel object can be created in MATLAB.

MATLAB also allows allocation of arrays on the graphics card. These arrays are used as input and output variables of the kernels. For the realization of random samples in MC experiments (generation of random state vectors), an array of length $n \cdot N$ of random numbers from $U(0, 1)$ is preallocated on the graphics card (n is the number of components and N is the number of experiments). To generate the ith state vector, n numbers from this array are used (beginning with the position $i \cdot n$ when indexing from 0).

4.3. Bitwise Operation. A further optimization of both algorithms can be achieved using bitwise operations. Their proper use results in a reduction of computing time as well as in lower memory requirements, which is especially important when using the CUDA technology.

The state indicator of a component is always zero or one. To save the state indicator, it is thus adequate to use one bit. In this context, simulation variables of data type "unsigned int" are used having the length of 32 bits. The binary code of such variable consists of 32 digits (0/1), and each of them represents the state of one component.

In the following numerical experiments the bitwise operations are used in the CUDA kernel code. Their use in the pure MATLAB implementation is not suitable, because the bitwise operations are extremely time-consuming.

5. Testing

For testing purposes the following well-scalable problem is used. "A system of n component with a serial structure is given. Availability of each component equals 0.999. It is necessary to quantify the availability of the system."

Table 1: Sequential analytical algorithm test.

n	2	4	8	10	12	16	20	24
Time [s]	0.0013	0.0014	0.021	0.083	0.33	5.3	87.4	1411

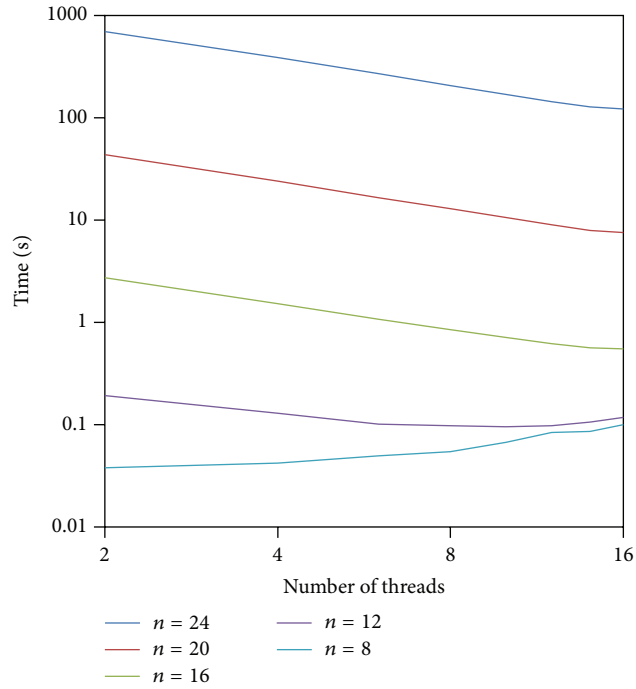

Figure 2: Simulation algorithm using parfor: scalability graph.

The system with serial structure is represented by the adjacency matrix given by

$$M = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & \vdots \\ 0 & 0 & 0 & \ddots & 0 \\ \vdots & & & & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}. \tag{9}$$

The problem was solved for various values of n using both analytical and simulation methods. All of the implementations were tested under equal conditions on two processors Intel Sandy Bridge E5-2665, 8 cores each. The CUDA parallelization was tested on the graphics card NVIDIA GT 555 M. The results of tests are shown in Tables 1–6 and Figures 2 and 3.

The analytical algorithm was tested on small systems with 24 components at most. Tables 1, 2, and 3 show that the analytical method is unsuitable for large systems, because (as expected) the computing time increases approximately exponentially with respect to n. However, for small systems, its use is appropriate since exact results are obtained.

The parallel scalability graph in Figure 2 shows the computation time decreasing when the number of threads grows from 2 to 16 threads (note the logarithmic scale of both axes).

Table 2: Analytical algorithm using parfor (time (s)).

n	2	4	8	10	12	16	20	24
2 threads	0.031	0.029	0.038	0.068	0.19	2.73	43.7	698.8
4 threads	0.072	0.037	0.042	0.058	0.12	1.52	24.0	388.5
8 threads	0.047	0.053	0.055	0.061	0.09	0.84	12.9	208.0
16 threads	0.094	0.095	0.100	0.089	0.11	0.55	7.54	122.1

Table 3: Analytical algorithm using CUDA and bitwise operations.

n	2	4	8	10	12	16	20	24
Time [s]	0.038	0.038	0.038	0.037	0.037	0.039	0.062	0.45

Table 4: Sequential simulation algorithm, sample size 10^5.

n	10	20	40	80	120	160	200
A	0.9900	0.9802	0.9608	0.9231	0.8869	0.8521	0.8186
$\overline{X}$	0.9903	0.9802	0.9613	0.9221	0.8862	0.8518	0.8201
PRSD	3.2036	2.2279	1.5761	1.0881	0.8823	0.7582	0.6752
Time [s]	12.26	19.14	34.25	66.89	100.65	135.85	173.35

Table 5: Simulation algorithm using parfor (time (s)).

n	10	20	40	80	120	160	200
2 threads	6.17	10.59	17.62	33.86	51.29	69.54	89.07
4 threads	3.22	5.53	9.01	17.41	26.36	35.71	45.63
8 threads	1.67	3.01	4.64	8.88	13.45	18.27	23.02
16 threads	0.97	1.56	2.54	4.81	6.86	9.36	11.86

Table 6: Simulation algorithm using CUDA and bitwise operations.

n	10	20	40	80	120	160	200
Time [s]	0.2336	0.027	0.0655	0.1803	0.358	0.5784	0.8684

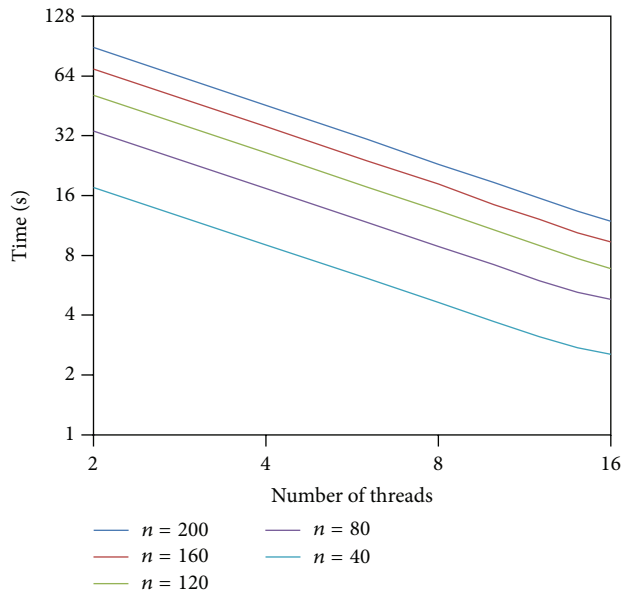

Figure 3: Simulation algorithm using parfor: scalability graph.

In case of the simulation algorithm besides the computation time also the estimated availability $\overline{X}$ and PRSD (percentage relative standard deviation) were recorded, which is demonstrated in Tables 4, 5, and 6. Random samples of size 10^5 were used. The number of system components was significantly increased as well to $n = 200$.

Also in case of the Monte Carlo method, it has been shown that the computation can be significantly reduced by the use of parallel computing. This fact is illustrated in the scalability graph in Figure 3, for several sizes of the system. The scale of both axes is logarithmic.

6. Industrial Example

In this section an example of a concrete engineering problem is presented; for its solution the aforementioned methods are used. The task is to simulate the availability of a fire detector system with independent components; for basic motivation see Rausand and Hoyland [11]. The block diagram is shown in Figure 4, and according to it, the corresponding adjacency matrix can be constructed.

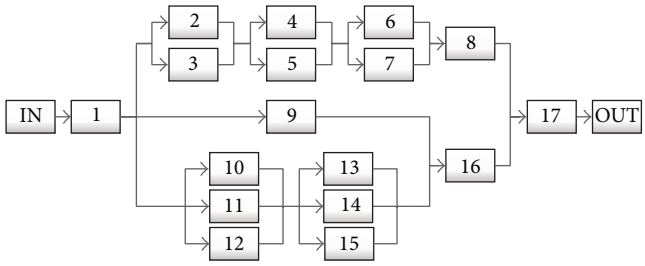

Figure 4: Block diagram of a fire detection system.

Table 7: Failure and repair rates.

i	Description	λ_i [h^{-1}]	μ_i [h^{-1}]
1	DC current source	$3 \cdot 10^{-5}$	0.018
2, . . . , 7	Smoke detector	$8.7 \cdot 10^{-4}$	0.025
8	Voting unit	$1.1 \cdot 10^{-4}$	0.02
9	Manual switch	$2 \cdot 10^{-5}$	0.01
10, . . . , 15	Fuse plug	$1.3 \cdot 10^{-3}$	0.03
16	Pressure switch	$2.5 \cdot 10^{-4}$	0.11
17	Start relay	$4.3 \cdot 10^{-4}$	0.11

Assume that the system is in its useful life period; that is, the failure rate and the repair rate of each component are constant. Therefore, the availability $p_i(t)$ of the ith component is given by Dubi [10]:

$$p_i(t) = \frac{\mu_i}{\lambda_i + \mu_i} + \frac{\lambda_i}{\lambda_i + \mu_i} e^{-(\lambda_i + \mu_i)t}, \quad (10)$$

where λ_i is the failure rate and μ_i is the repair rate of the ith component, exactly specified in Table 7.

Assume that in time $t_0 = 0$ (h) all components are operational. The availability of the system is simulated in time; thus an appropriate time discretization should be chosen. Here, for example, the constant step of 6 hours is chosen. At this point, the simulation of the availability course in time can be computed. For each time step, availability of each component is first calculated using formula (10); then the availability of the whole system is calculated using one of the methods presented in Sections 3 and 4. It is convenient to plot the results for each time step into a graph; see Figure 5.

Table 8 compares the computation time needed to calculate the availability in 50 time steps (plotted in the graphs in Figures 5 and 6) by different types of the algorithm.

7. Conclusions

This paper was aimed at two specific computing methods (analytical versus simulation) which can be used in applications related to availability quantification of systems represented by the adjacency matrix. Both methods were parallelized; that is, their corresponding algorithms (see Section 3) were implemented in parallel processing, by two ways:

(i) parallelization using the MATLAB parfor loop,

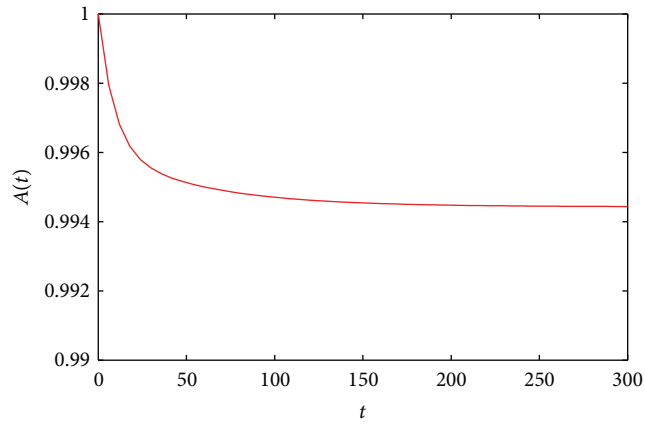

Figure 5: Dependence of the availability on the time: analytical method.

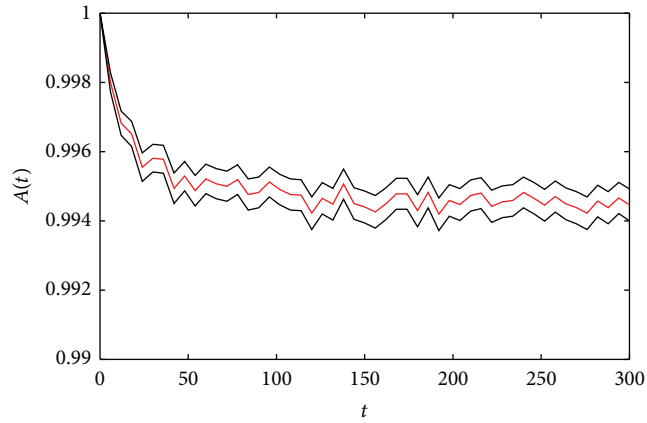

Figure 6: Dependence of the availability on the time, including 95% confidence interval, Monte Carlo method.

Table 8: Computation time comparison.

Algorithm type	Method	Time [s]
Sequential	Analytical	543.09
parfor in 16 threads		47.68
CUDA		1.965
Sequential	Monte Carlo (sample size 10^5)	435.21
parfor in 16 threads		34.27
CUDA		1.034

(ii) parallelization using the CUDA technology together with the use of the bitwise operations.

Using these two parallelization techniques, the computation efficiency of both algorithms was successfully enhanced. More significant reduction of the computing time was achieved using the second technique. For instance, the practical example presented in Section 6 was computed over a computing time which is 275 times shorter than the computing time used for the original sequential implementation.

The parallel scalability was illustrated in the first of these two techniques (the parfor loop). The scalability graphs in

Figures 3 and 4 confirm that the use of parallel computing is always more efficient than the traditional approach using the sequential implementation.

On testing example we showed that the simulation methodology is particularly convenient when the number of system components increases. For instance, if the number of system components approaches $n = 200$, this method gives acceptable results, whereas using analytical method is impossible.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This paper has been prepared in the framework of the IT4Innovations Centre of Excellence Project, Reg. no. CZ.1.05/1.1.00/02.0070, supported by Operational Programme "Research and Development for Innovations" funded by Structural Funds of the European Union and the Government of Czech Republic.

References

[1] E. Zio, "Reliability engineering: old problems and new challenges," *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 125–141, 2009.

[2] R. Briš, "Parallel simulation algorithm for maintenance optimization based on directed Acyclic Graph," *Reliability Engineering & System Safety*, vol. 93, no. 6, pp. 874–884, 2008.

[3] A. Zdráhala and R. Briš, "Generalized test-cases for dynamic reliability: a Monte Carlo approach," in *MENDEL 2011—17th International Conference on Soft Computing; Proc. Intern. Conf. 326–333, Brno, 2011*, R. Matoušek, Ed., Faculty of Mechanical Engineering, University of Technology, Brno, Czech Republic, 1st edition, 2011.

[4] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo Method*, John Wiley & Sons, New York, NY, USA, 2008.

[5] P. Labeau and E. Zio, "Biasing schemes in component-based and system-based Monte Carlo algorithms in system engineering," in *Proceedings of the ESREL European Safety and Reliability Conference*, pp. 903–910, Torino, Italy, September 2001.

[6] G. S. Fishman, *Monte Carlo, Concepts, Algorithms, and Applications*, Springer Series in Operations Research, Springer, New York, NY, USA, 1996.

[7] D. Perez, R. Garnier, M. Chevalier, and J. P. Signoret, "A comprehensive method to speed up generalised stochastic Petri net Monte Carlo simulation for non-Markovian systems," in *Proceedings of the European Safety and Reliability Conference—Probabilistic Safety Assessment & Management Conference (ESREL—PSAM III '96)*, vol. 3, pp. 1941–1946, Crete, Greece, June 1996.

[8] J.-P. Signoret, Y. Dutuit, P.-J. Cacheux, C. Folleau, S. Collas, and P. Thomas, "Make your Petri nets understandable: reliability block diagrams driven Petri nets," *Reliability Engineering and System Safety*, vol. 113, no. 1, pp. 61–75, 2013.

[9] Y. Mochizuki, Y. Hayashi, N. Oda, K. Takeuchi, and K. Takeda, "Technology for high reliability system LSIs," *NEC Technical Journal*, vol. 1, no. 3, pp. 55–59, 2006.

[10] A. Dubi, *Monte Carlo Applications in Systems Engineering*, John Wiley & Sons, New York, NY, USA, 2000.

[11] M. Rausand and A. Hoyland, *System Reliability Theory*, John Wiley & Sons, Hoboken, NJ, USA, 2nd edition, 2004.