All-Terminal Reliability Evaluation through a Monte Carlo simulation based on an MPI implementation

Silvia Pascual Martínez[a], Beatriz Otero Calviño[a], Claudio M. Rocco S.[b*]
[a]Universitat Politècnica de Catalunya, Barcelona, Spain
[b]Universidad Central de Venezuela, Caracas, Venezuela

Abstract: All-terminal reliability (ATR), defined as the probability that every node in a network can communicate with every other node, is an important problem in research areas such as mobile ad-hoc wireless networks, grid computing systems, and telecommunications. The assessment of ATR has also been part of related problems like the reliability allocation problem. However, the exact calculation of ATR is a NP-hard problem. To obtain this probability, there are approaches based on analytic methods for small networks or estimation through Monte Carlo simulation (MCS). In this paper, a parallel MCS implementation, based on the Message Passing Interface (MPI) standard is presented. The implementation can take advantage of the existence of multiprocessor thus reducing the time required for the ATR assessment. Three examples related to real network illustrate the benefits.

Keywords: All-terminal reliability, Monte Carlo Simulation, MPI

1. INTRODUCTION

All-terminal reliability (ATR), defined as the probability that every node in a network can communicate with every other node, is an important problem in research areas such as mobile ad-hoc wireless networks, grid computing systems, and telecommunications [1].

An important area of application of ATR is related to the optimal design of structures, for example the design of computers or communication networks, where cost and reliability are important objectives. There are many formulations of this problem, but in general the idea is to define the layout of the network at minimum cost while meeting a minimum ATR requirement [1].

A common aspect of such methods is that the network reliability must be assessed for each of the possible solutions or candidate topologies. The search space is: $k^{\frac{(|N|x(|N|-1)}{2}}$, where k is the number of choices for the links and $|N|$ is the number of nodes in the network [1]. For example, for a binary choice ($k=2$) a network with $|N|$ = 20 has 1.57 x 10^{57} possible designs. This means that network reliability must be evaluated using a very fast procedure. Of course, heuristic procedures, like genetic algorithm [2,3], evolutionary strategies [4], artificial neural network [1] among others, have been studied so that the search space is not fully explored. In those procedures the ATR of every possible solution considered must be assessed.

However, the exact calculation of ATR is an NP-hard problem, with "computational effort growing exponentially with the number of nodes and link in the network" [1]. To obtain this probability, there are approaches based on analytic methods for small networks, upper or lower bounds or estimation through Monte Carlo simulation (MCS).

Several analytical procedures have been suggested to assess the ATR, like the enumeration of all possible minimal cutsets or the approaches proposed in [5,6]. Upper and lower bounds [1,7] are an interesting approach. The main idea is to obtain a rough estimation of ATR, using a relatively fast procedure (in general with a computational effort of $\boldsymbol{O}(N^3)$. This estimation is used to discard possible solutions that do no meet the reliability constraint. However, if a more precise ATR value is required, the approach does not provide any help. MCS can assess ATR very precisely but requires a high computational effort to obtain a good estimate [8].

In this paper we evaluate the benefits of a MCS parallel implementation. The idea is to take advantage of the existence of multiprocessor and thus reducing the time required for the assessment.

The remainder of this paper is organized as follows: Section 2 describes the Message Passing Interface (MPI) standard and the program implemented. Section 3 presents the comparative assessment performed on three real networks of different size. Finally, section 4 shows the conclusions and future works.

2. MESSAGE PASSING INTERFACE

2.1 Parallel programming model

MPI (Message-Passing Interface) is a message-passing library interface specification to be used on parallel computers, with operations expressed as functions, subroutines, or methods, according to the appropriate language bindings. MPI can be found in multiple implementations. We have chosen an open source MPI implementation called OpenMPI [9] since it includes options that allow us to specify how to assign the MPI processes using the available processors. When a MPI program is launched we can specify how many processes are going to take part in the MPI execution. In order to achieve a faster execution we need to involve as many processes as available processors.

2.2 Parallel implementation

The first step for a parallel implementation is to study how the serial program is performed. For ATR problems the following MCS serial steps are performed: Firstly, a random sample of the link status is generated, according to their reliability. Then, the corresponding network topology is evaluated to detect a spanning tree (for example using the Prim algorithm): if there is at least one spanning tree the nodes are all connected (success) otherwise at least a node is isolated. The process is repeated NSIM times. At the end, the ATR is approximated as the ratio between the number of successful topologies (*ioper*) and NSIM.

In the parallel implementation, the NSIM samples are shared evenly among the available processors. For example, if m=4 processors are available, then NSIM/4 samples are generated and evaluated per processor. Every MPI process executes its own simulation without interfering with other process executions. Figure 1 describes the parallel program implemented.

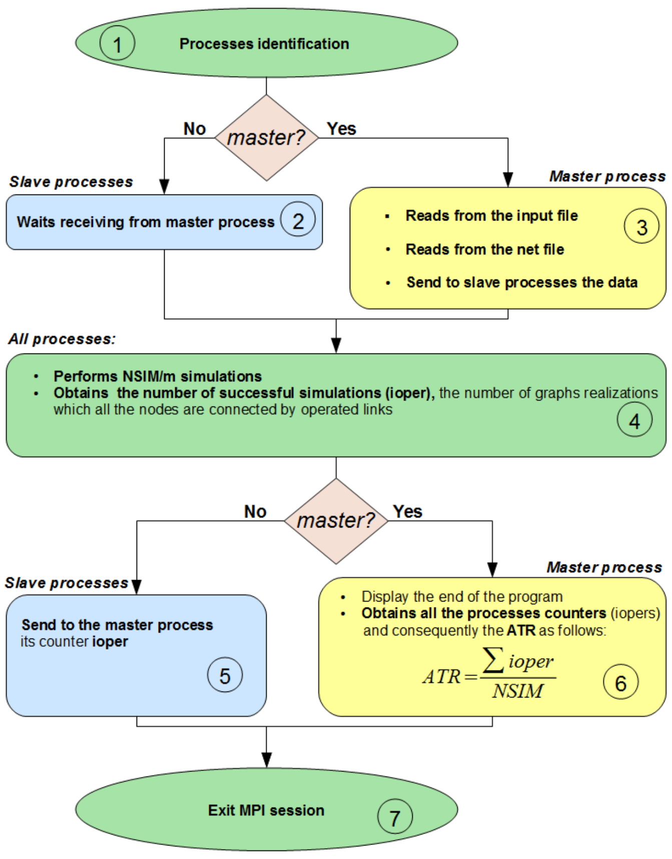
This program is defined by the following seven steps:

1. The program uses the MPI process identifiers to associate them to a “role” process. The program assigns one of the processes as the master process while the others are defined as slave processes.
2. All the slave processes wait for the data before executing their own simulation.
3. The master process, after reading the information from the program inputs, sends it to all the slave processes
4. All MPI processes, master and slaves, execute simultaneously their simulations NSIM/m times, where m is the number of processes. As a result each process obtains the counter of successfully topology configurations ($ioper_i$, i=1,..,m).
5. Every slave process sends its counter to the master process.
6. The master process, after obtaining the number of successful topologies from all the slaves, estimates the ATR as the ratio between the total number of successful topologies and NSIM.
7. All processes finalize their MPI sessions.

In order to get a correct estimation of the reliability each MPI process must use a different seed. If not, those processes with the same seed would evaluate the same NSIM/m samples and the estimation of the reliability would be poor. To avoid this situation, the MPI program sends a different seed to each process (step 3).

3. COMPUTATIONAL EXAMPLES

In this section we describe the procedure used to assess the benefits of the MPI implementation. Three real network topologies are used: A Belgian telephone network (52 nodes, 76 links) [10] and the topology of two electric power systems: The Italian high voltage network (127 nodes and 171 links) [11] and the Venezuela Interconnected Power Systems (635 nodes and 1179 links) [12]. Figures 2 to 4 show the corresponding network topologies. It is important to pinpoint that in these examples the effects of topology size on the MPI implementation are evaluated and no additional consideration are derived from the ATR estimation.



1 Processes identification
master?
No
Yes
Slave processes
Waits receiving from master process 2
Master process
Reads from the input file
Reads from the net file
Send to slave processes the data
3
All processes:
Performs NSIM/m simulations
Obtains the number of successful simulations (ioper), the number of graphs realizations which all the nodes are connected by operated links
4
master?
No
Yes
Slave processes
Send to the master process its counter ioper
5
Master process
Display the end of the program
Obtains all the processes counters (iopers) and consequently the ATR as follows:
ATR = Σ ioper / NSIM
6
Exit MPI session 7

Figure 1: Diagram of parallel implementation

First, each network is evaluated using a “serial” Fortran program and a crude MCS implementation. The time required to complete the ATR estimation is stored as T_{serial}. For each parallel implementation, the time required to estimate the ATR is stored as T_m, m=1,2,4,8. At the end, the index T_m/T_{serial} is evaluated. In all the examples, NSIM is set to 10000. Link reliability is assumed as 0.95.

Both serial and parallel implementations are evaluated 50 times, and minimum-average-maximum results are presented. The MPI program was compiled using the gfortran compiler [13]. All computational examples were executed on an Intel® Core™ i7-720QM with 8 processors [14].

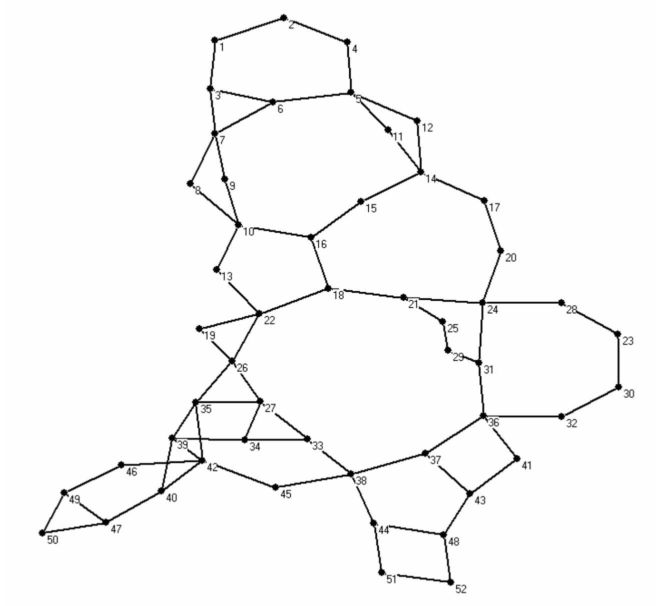


Figure 2: Belgian Network [10]

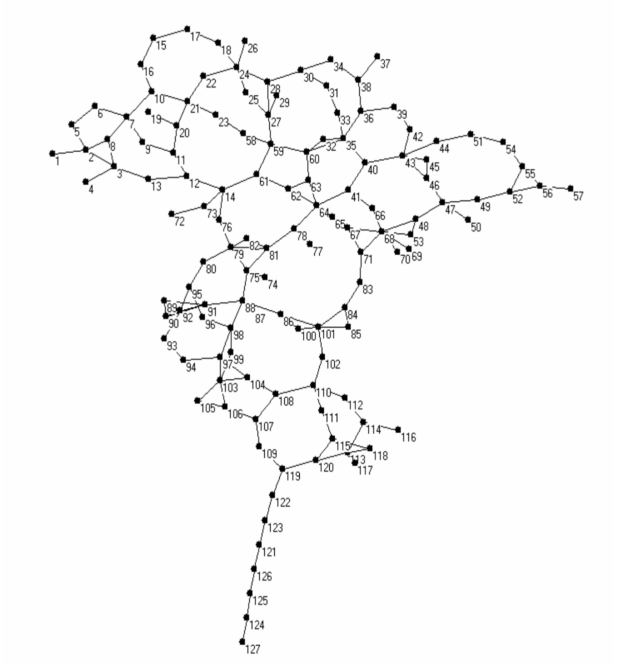


Figure 3: Italian High Voltage Power Network [11]

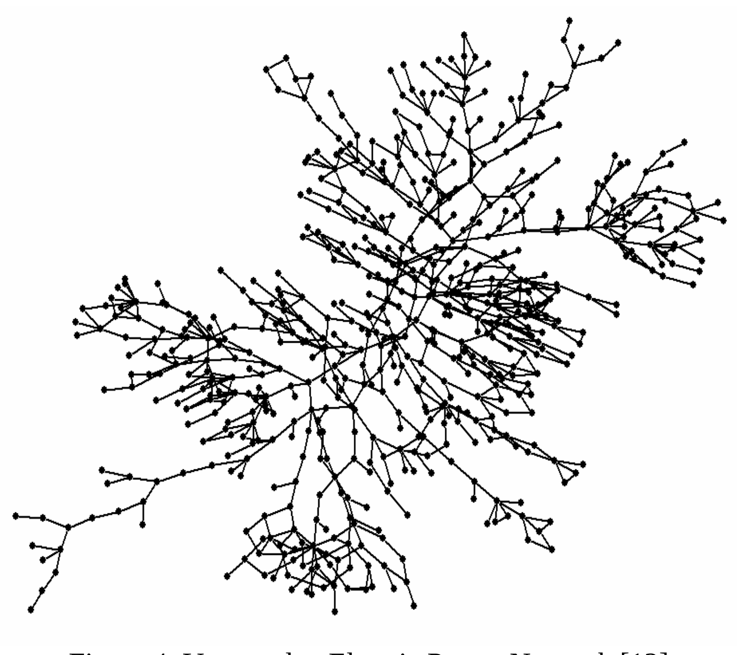


Figure 4: Venezuelan Electric Power Network [12]

Table 1 to 3 present the results for the serial case along with the parallel implementation for each network analyzed: Total execution time (sec.), T_{serial}/T_m and min-average-max ATR. As expected, as the size of the network increases, the ATR estimation is lower.

The analysis of these tables reveals that:

a) Similar estimates of ATR were obtained using different number of processors. That means the number of MPI processes involved does not degrade the ATR estimation.
b) The execution time decreases proportionally with the number of processors used.
c) The maximum relation T_{serial}/T_m could be 7.5 on average.

It is clear that the use of a parallel implementation is able to reduce the execution time. That means that, for the same serial time execution, the number of samples on the parallel implementation could be increased and a better ATR estimation could be achieved.

Table 1: Results for the Belgian Network

Case	T_m(sec)	T_{serial}/T_m	Min ATR estimation	Average ATR estimation	Max ATR estimation
m=1 (serial)	0,46	1	9.05E-01	9,09E-01	9.15E-01
m=2	0,22	2,09	9.05E-01	9,09E-01	9.14E-01
m=4	0,12	3,84	9.06E-01	9,10E-01	9.14E-01
m=8	0,06	7,67	9.05E-01	9,09E-01	9.13E-01

Table 2: Results for the Italian Network

Case	T_m(sec)	T_{serial}/T_m	Min ATR estimation	Average ATR estimation	Max ATR estimation
m=1 (serial)	1,05	1	2.61E-01	2,66E-01	2.72E-01
m=2	0,52	2,02	2.61E-01	2,66E-01	2.72E-01
m=4	0,26	4,04	2.60E-01	2,67E-01	2.73E-01
m=8	0,14	7,5	2.59E-01	2,66E-01	2.74E-01

Table 3: Results for the Venezuelan Network

Case	T_m(sec)	T_{serial}/T_m	Min ATR estimation	Average ATR estimation	Max ATR estimation
m=1(serial)	8,9	1	3.50E-04	7.01E-04	1.25E-03
m=2	4,29	1,96	3.00E-04	7.03E-04	1.30E-03
m=4	2,35	3,58	3.00E-04	7.41E-04	1.20E-03
m=8	1,14	7,39	3.50E-04	7.02E-04	1.15E-03

4. CONCLUSIONS

In this work we presented a parallel MCS implementation based on the MPI standard. Our computational results show that using all of the processors available, this implementation could be almost 7.5 times faster than the serial MCS implementation. Regardless of the number of MPI processes involved in the simulation, the final ATR estimations are almost the same.

As a future work, it could be interesting to implement another version of parallel MCS, using other programming models (e.g., StarSs [15]) that exploit more efficiently the resources available. Moreover, after studying the sequential program we found that the Prim algorithm (used for detecting spanning trees) and its related functions consume a lot of processing time and it is possible to consider a parallel implementation allowing further execution time reduction.

Acknowledgements

This work has been supported by the HiPEAC European Network of Excellence and the Ministry of Science and Technology of Spain, the European Union (FEDER) under contract TIN2007-60625.

References

[1] Srivaree-ratana Ch., Abdullah K., Smith A.E. Estimation of all-terminal network reliability using an artificial neural network Computers & Operations Research 29 pp. 849-868, 2002

[2] Deeter DL, Smith AE. Heuristic optimization of network design considering all-terminal reliability. Proceedings of the Reliability and Maintainability Symposium,. p. 194-49, 1997

[3] Dengiz B, Altiparmak F, Smith AE. Efficient optimization of all-terminal reliable networks using an evolutionary approach. IEEE Transactions on Reliability, 46:18-26, 1997

[4] Ramirez-Marquez, J.E., Rocco, C.M. All-terminal network reliability optimization via probabilistic solution discovery, Reliability Engineering and System Safety, 93: 1689-1697, 2008

[5] Aggarwal KK, Rai S. Reliability evaluation in computer-communication networks. IEEE Transactions on Reliability;R-30:32-5, 1981

[6] Rai S. A cutset approach to reliability evaluation in communication networks. IEEE Transactions on Reliability;R-31:428-31. 1982

[7] Jan R-H, Hwang F-J, Chen S-T. Topological optimization of a communication network subject to a reliability constraint. IEEE Transactions on Reliability, 42:63-70., 1993

[8] Fishman G.S. A Monte Carlo sampling plan for estimating network reliability. Operations Research, 34:581-94.1986

[9] www.mpi-forum.org/

[10] Manzi E, Labbé M, Latouche G, Maffioli F. Fishman's sampling plan for computing network reliability. IEEE Transactions on Reliability; R-50: 41–6, 2001.

[11] Crucitti P, Latora V, Marchiori M. Locating critical lines in high voltage electrical power grids. Fluctuation and Noise Letters; 5(2):L201–8, 2005

[12] Rocco S., Claudio M.: "Analysis of the Venezuela Electric Power Grid through Complex Network Concepts", Revista de la Facultad de Ingeniería de la U.C.V., Vol. 23, N° 1, pp. 103–109, 2008 (In Spanish)

[13] http://gcc.gnu.org/fortran/

[14] http://ark.intel.com/products/43122

[15] https://www.bscmsrc.eu/media/events/barcelona-multicore-workshop-2010/jesus-labarta-abstract