HYDRA: **HY**pergraph-based **D**istributed **R**esponse-time **A**nalyser

Nicholas J. Dingle* Peter G. Harrison William J. Knottenbelt

Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ, United Kingdom.
Email: {njd200,pgh,wjk}@doc.ic.ac.uk

March 5, 2003

Abstract

It is important for almost all transaction processing and computer-communication systems to satisfy response time quantile targets. This paper describes HYDRA, a scalable parallel tool for the analytical determination of response time densities in large, structurally-unrestricted Markov models derived from high-level specifications. The tool exploits an efficient distributed uniformization-based algorithm, combined with hypergraph partitioning to balance computational load across processors while minimising communication. We demonstrate our tool on a 1.6 million state Generalized Stochastic Petri Net model of a flexible manufacturing system, comparing the accuracy of our results with simulation and contrasting the run-time performance of our technique with an approach based on numerical Laplace transform inversion.

1 Introduction

Stochastic performance models (such as Petri nets or queueing networks) provide a formal way to capture and analyse the dynamic behaviour of computer and communication systems. Traditionally, performance statistics for these models are derived by solving a Markov chain corresponding to the model's behaviour at the state transition level. From the chain's equilibrium probability distribution, standard performance measures (such as utilization and throughput) and *expected* values of various sojourn times can be obtained.

This paper addresses the harder problem of calculating full response time densities in structurally unrestricted Markov models. This has important practical applications since response time quantiles are often specified as quality of service metrics in Service Level Agreement contracts and industry standard benchmarks such as TPC. In the past, numerical computation of analytical response time densities has proved prohibitively expensive except in some Markovian systems with restricted structure such as overtake-free queueing networks [1]. However, with the advent of high-performance parallel computing and the widespread availability of PC clusters, direct numerical analysis on Markov chains has now become a practical proposition.

Our contribution is a parallel algorithm implemented in the HYDRA tool for computing passage time densities in Markov chains with large state spaces. By using state-of-the-art hypergraph partitioning techniques

*Corresponding author. Telephone: +44 (0)20 7594 8253 Fax: +44 (0)20 7581 8024

we achieve a scalable algorithm that yields excellent performance on a distributed memory parallel computer and that effectively utilizes the compute power and RAM provided by a network of workstations. To the best of our knowledge, this is the first application of hypergraph partitioning in the domain of performance analysis (and one of the few application areas of hypergraphs outside VLSI circuit design). Further, we are not aware of any other distributed uniformization-based tools for computing response time densities, and our implementation improves substantially on both the solution time and capacity of contemporary distributed response time density analysers based on numerical Laplace transform inversion [2].

2 Response time densities in Markov models

2.1 First Passage Times

Consider a finite, irreducible, continuous time Markov Chain (CTMC) with n states $\{1, 2, \ldots, n\}$ and $n \times n$ generator matrix $\mathbf{Q}$. If $X(t)$ denotes the state of the CTMC at time $t \geq 0$, then the first passage time from a source state i into a non-empty set of target states $\vec{j}$ is:

$$T_{i\vec{j}}(t) = \inf\{u > 0 : \ X(t+u) \in \vec{j} \mid X(t) = i\} \qquad (\forall t \geq 0)$$

For a stationary, time-homogeneous CTMC, $T_{i\vec{j}}(t)$ is independent of t, so:

$$T_{i\vec{j}} = \inf\{u > 0 : \ X(u) \in \vec{j} \mid X(0) = i\}$$

$T_{i\vec{j}}$ is a random variable with an associated probability density function $f_{i\vec{j}}(t)$. To determine $f_{i\vec{j}}(t)$ we must convolve the exponentially distributed state holding times over all possible paths (including cycles) from state i into any of the states in the set $\vec{j}$. As shown in the next section, the problem can be readily extended to multiple initial states by weighting first passage time densities.

2.2 Uniformization

Uniformization [3, 4] transforms a CTMC into one in which all states have the same mean holding time $1/q$, by allowing 'invisible' transitions from a state to itself. This is equivalent to a discrete-time Markov chain (DTMC), after normalization of the rows, together with an associated Poisson process of rate q. The one-step DTMC transition matrix $\mathbf{P}$ is given by:

$$\mathbf{P} = \mathbf{Q}/q + \mathbf{I} \tag{1}$$

where $q > \max_i |q_{ii}|$ (to ensure that the DTMC is aperiodic). The number of transitions in the DTMC that occur in a given time interval is given by a Poisson process with rate q.

While uniformization is normally used for transient analysis, it can also be employed for the calculation of response time densities [5, 6]. We add an extra, absorbing state to our uniformized chain, which is the sole successor state for all target states (thus ensuring we calculate the *first* passage time density). We denote by $\mathbf{P}'$ the one-step transition matrix of the modified, uniformized chain. Remembering that the time taken to traverse a path with n hops in this chain will have an Erlang distribution with parameters n and q, the density of the time taken to pass from a set of source states $\vec{i}$ into a set of target states $\vec{j}$ is given by:

$$f_{\vec{i}\vec{j}}(t) = \sum_{n=1}^{\infty} \frac{q^n t^{n-1} e^{-qt}}{(n-1)!} \sum_{k \in \vec{j}} \pi_k^{(n)} \tag{2}$$

where

$$\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)} \mathbf{P}' \qquad \text{for } n \geq 0$$

with

$$\pi_k^{(0)} = \begin{cases} 0 & \text{for } k \notin \vec{i} \\ \pi_k / \sum_{j \in \vec{i}} \pi_j & \text{for } k \in \vec{i} \end{cases} \tag{3}$$

and in which $\boldsymbol{\pi}$ is any non-zero solution to $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}$.

Truncation is employed to approximate the infinite sum in Eq. 2, terminating the calculation when the Erlang term drops below a specified threshold value. Concurrently, when the convergence criterion

$$\frac{||\boldsymbol{\pi}^{(n+1)} - \boldsymbol{\pi}^{(n)}||_\infty}{||\boldsymbol{\pi}^{(n)}||_\infty} < \epsilon \tag{4}$$

is met, for given tolerance ϵ, the steady state probabilities of $\mathbf{P}'$ are considered to have been obtained with sufficient accuracy and no further multiplications with $\mathbf{P}'$ are performed.

3 Hypergraph Partitioning

The key opportunity for parallelism in the uniformization algorithm is the sparse matrix-vector product $\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)}\mathbf{P}'$ (or equivalently $\boldsymbol{\pi}^{(n+1)T} = \mathbf{P}'^T\boldsymbol{\pi}^{(n)T}$, where the superscript T denotes the transpose operator). To perform these operations efficiently it is necessary to map the non-zero elements of $\mathbf{P}'$ onto processors such that the computational load is balanced and communication between processors is minimized. To achieve this, we use hypergraph-based partitioning techniques to assign matrix rows and corresponding vector elements to processors in a row-striped partitioning.

Hypergraphs are extensions of graph data structures that, until recently, were primarily applied in VLSI circuit design. Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined by a set of vertices $\mathcal{V}$ and a set of nets (or hyperedges) $\mathcal{N}$, where each net is a subset of the vertex set $\mathcal{V}$ [7]. In the context of a row-wise decomposition of a sparse matrix, matrix row i ($1 \leq i \leq n$) is represented by a vertex $v_i \in \mathcal{V}$ while column j ($1 \leq j \leq n$) is represented by net $N_j \in \mathcal{N}$. The vertices contained within net N_j correspond to the row numbers of the non-zero elements within column j, i.e. for matrix $\mathbf{A}$, $v_i \in N_j$ if and only if $a_{ij} \neq 0$. The weight of vertex i is given by the number of non-zero elements in row i, while the weight of a net is its contribution to the hyperedge cut, defined as one less than the number of different partitions spanned by that net.

The overall objective of a hypergraph sparse matrix partitioning is to minimize the total hyperedge cut while maintaining a balance criterion. The key advantages of the hypergraph approach over traditional graph partitioning are that hyperedge cut quantifies the communication cost exactly and that off-diagonal non-zeros tend to be positioned within rows so as to minimize the number of remote vector elements required. In graph partitioning, the sole objective is to minimize the number of off-diagonal non-zeros. Like graph partitioning, optimal hypergraph partitioning is NP-complete. However, there are a small number of hypergraph partitioning tools which implement fast sub-optimal heuristic algorithms, for example PaToH [7] and hMeTiS [8].

4 The HYDRA Tool

Fig. 1 shows the architecture of the HYDRA tool. The process of calculating a response time density begins with a high-level model specified in an enhanced form of the DNAmaca interface language [9, 10]. This language supports the specification of queueing networks, stochastic Petri nets and stochastic process algebras. Next, a probabilistic, hash-based state generator [11] uses the high-level model description to produce the generator matrix $\mathbf{Q}$ of the model's underlying Markov chain as well as a list of the initial and target states. $\mathbf{P}$ is constructed from $\mathbf{Q}$ according to Eq. 1 and normalized weights for the initial states are determined from Eq. 3 by the solution of $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}$. This is readily done using any of a variety of steady-state solution techniques (e.g. [12, 13]). From $\mathbf{P}$, $\mathbf{P}'^T$ is constructed by transposing the underlying

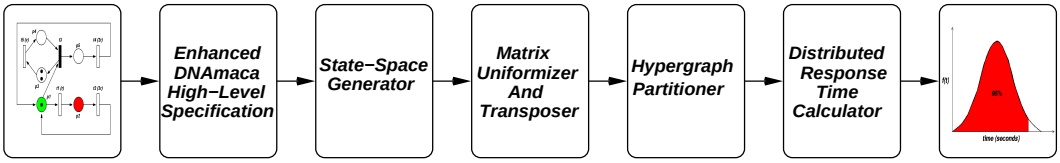

Figure 1: HYDRA tool architecture.

Markov chain and adding an extra terminal state that becomes the sole successor state of all target states. $\mathbf{P}'^T$ is then partitioned using a hypergraph partitioning tool.

The pipeline is completed by our distributed response time density calculator, which is implemented in C++ using the Message Passing Interface (MPI) [14] standard. Initially each processor tabulates the Erlang terms for each t-point required (cf. Eq. 2). Computation of these terms ends when they fall below a specified threshold value. In fact, this is safe to use as a truncation condition for the entire passage time density expression because the Erlang term is multiplied by a summation which is a probability. The terminating condition also determines the maximum number of hops m used to calculate the right-hand factor, a sum which is independent of t.

Each processor reads in the rows of the matrix $\mathbf{P}'^T$ that correspond to its allocated partition into two types of sparse matrix data structure and also computes the corresponding elements of the vector $\boldsymbol{\pi}^{(0)}$. *Local* non-zero elements (i.e. those elements in diagonal matrix blocks that will be multiplied with vector elements stored locally) are stored in a conventional compressed sparse row format. *Remote* non-zero elements (i.e. those elements in off-diagonal matrix blocks that must be multiplied with vector elements received from other processors) are stored in an ultrasparse matrix data structure – one for each remote processor – using a coordinate format. Each processor then determines which vector elements need to be received from and sent to every other processor on each iteration, adjusting the column indices in the ultrasparse matrices so that they index into a vector of received elements. This ensures that a minimum amount of communication takes place and makes multiplication of off-diagonal blocks with received vector elements efficient.

The vector $\boldsymbol{\pi}^{(n)}$ is then calculated for $n = 1, 2, 3, \ldots, m$ by repeated sparse matrix-vector multiplications of form $\boldsymbol{\pi}^{(n+1)T} = \mathbf{P}'^T\boldsymbol{\pi}^{(n)T}$. Actually, fewer than m multiplications may take place since a test for steady state convergence is made after every iteration (cf. Eq. 4).

For each matrix-vector multiplication, each processor begins by using non-blocking communication primitives to send and receive remote vector elements, while calculating the product of local matrix elements with locally stored vector elements. The use of non-blocking operations allows computation and communication to proceed concurrently on parallel machines where dedicated network hardware supports this effectively. The processor then waits for the completion of non-blocking operations (if they have not already completed) before multiplying received remote vector elements with the relevant ultrasparse matrices and adding their contributions to the local matrix-vector product cumulatively.

From the resulting local matrix-vector products each processor calculates and stores its contribution to the sum $\sum_{k\in\vec{j}} \pi_k^{(n)}$. After m iterations have completed, these sums are accumulated onto an arbitrary master processor where they are multiplied with the tabulated Erlang terms for each t-point required for the passage time density. The resulting points are written to a disk file and are displayed using the GNUplot graph plotting utility.

5 Numerical Results

In this section we demonstrate the applicability of the HYDRA tool by computing a first passage time density in a Petri net model of a manufacturing system. We validate the density produced against a simulation and consider the scalability of our algorithm on two different parallel architectures.

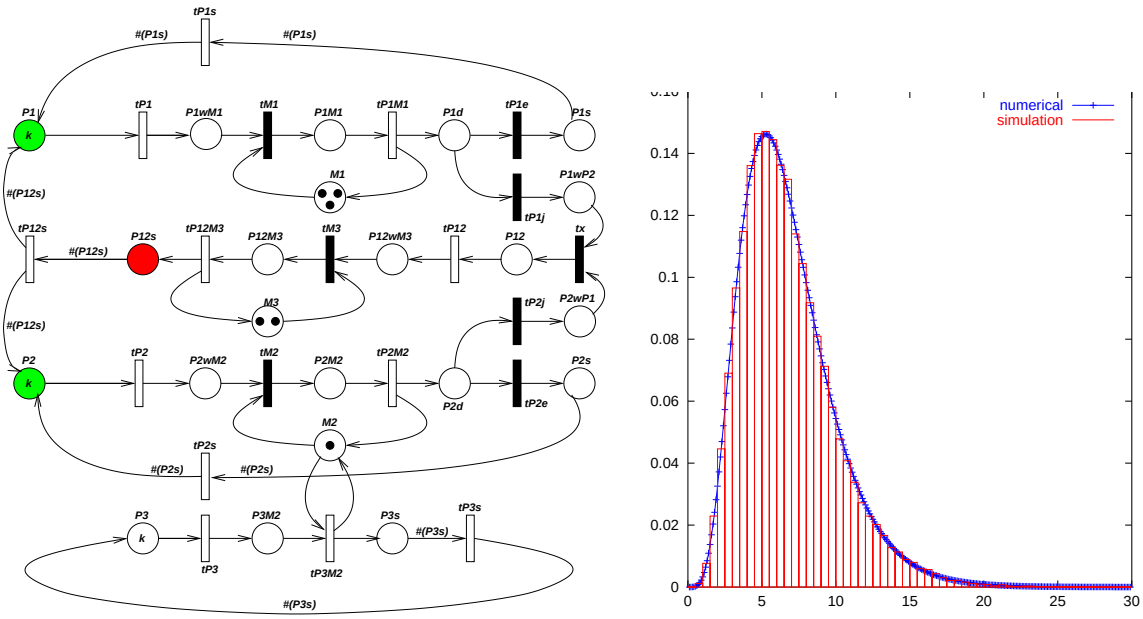

Figure 2: The Flexible Manufacturing System (FMS) GSPN [15] (left) and corresponding numerical and simulated passage time densities for the time taken to produce a finished part of type $P12$ starting from states in which there are $k = 7$ unprocessed parts of types $P1$ and $P2$ (right).

Fig. 2 shows a 22-place Generalized Stochastic Petri net (GSPN) model of a flexible manufacturing system. A full description of this model, which we will refer to as the FMS model, can be found in [15]. For our purposes it suffices to note that the model describes an assembly line with three types of machines ($M1$, $M2$ and $M3$) which assemble four types of parts ($P1$, $P2$, $P3$ and $P12$). Initially there are k unprocessed parts of each type $P1$, $P2$ and $P3$ in the system. There are no parts of type $P12$ at start-up since these are assembled from processed parts of type $P1$ and $P2$ by the machines of type $M3$. When parts of any type are finished, they are stored for shipping on places $P1s$, $P2s$, $P3s$ and $P12s$.

For $k = 7$, the GSPN's underlying Markov chain has $1\,639\,440$ states and $13\,552\,968$ non-zero off-diagonal entries in its generator matrix $\mathbf{Q}$. For this model, we calculate the density of the time taken to produce a finished part of type $P12$ starting from any state in which there are 7 unprocessed parts of type $P1$ and 7 unprocessed parts of type $P2$. That is, the source markings (of which there are 36) are those where $M(P1) = M(P2) = 7$ and the target markings (of which there are 429 624) are those where $M(P12s) = 1$. After modification of the state graph to allow for transitions from target states to a new terminal state, the uniformized matrix $\mathbf{P}'$ has $11\,001\,408$ non-zero entries. The hypergraph tool PaToH is then used to partition the rows of the transposed matrix $\mathbf{P}'^T$. The resulting numerically calculated passage time density and a histogram of simulated passage time density are shown on the right in Fig. 2. There is very good agreement between the numerical and simulated passage time densities.

Table 1 shows the performance of our algorithm on two architectures: a Fujitsu AP3000 distributed memory parallel computer running Solaris and a Linux-based PC workstation cluster. The AP3000 is based on a grid of 60 processing nodes, each of which has a UltraSPARC 300MHz processor and 256MB RAM. These nodes are interconnected by a 2D wraparound mesh network that uses wormhole routing and that has a peak throughput of 520Mbps (megabits per second). The PC cluster is a vanilla network of workstations, consisting of 32 Althon 1.4GHz PCs each with 512MB RAM linked together by a 100Mbps switched Ethernet network. Distributed run-time is measured as the maximum processor run-time from the start of the first uniformization iteration. The speedup for p processors, denoted by S_p, is given by the run-time of the sequential solution ($p = 1$) divided by the run-time with p processors. Efficiency for p processors, denoted by E_p, is defined as $E_p = S_p/p$.

Corresponding graphs of the speedup and efficiency achieved on each architecture are presented in Fig. 3.

	AP3000			PC Cluster			Comm. per iteration	
p	time (s)	S_p	E_p	time (s)	S_p	E_p	Messages	Vol (MB)
1	1243.3	1.00	1.000	325.0	1.00	1.000	0	0
2	630.5	1.97	0.986	258.7	1.26	0.628	2	1.5
4	328.2	3.78	0.947	197.1	1.65	0.412	12	3.2
8	182.3	6.82	0.853	143.0	2.27	0.284	51	5.3
16	99.7	12.47	0.779	114.6	2.84	0.178	207	7.3
32	58.6	21.22	0.663	71.7	4.53	0.142	663	9.6

Table 1: Run-time, speedup (S_p), efficiency (E_p) and per-iteration communication overhead for p-processor passage time density calculation in the FMS model with $k = 7$. Results are presented for an AP3000 distributed memory parallel computer and a PC cluster.

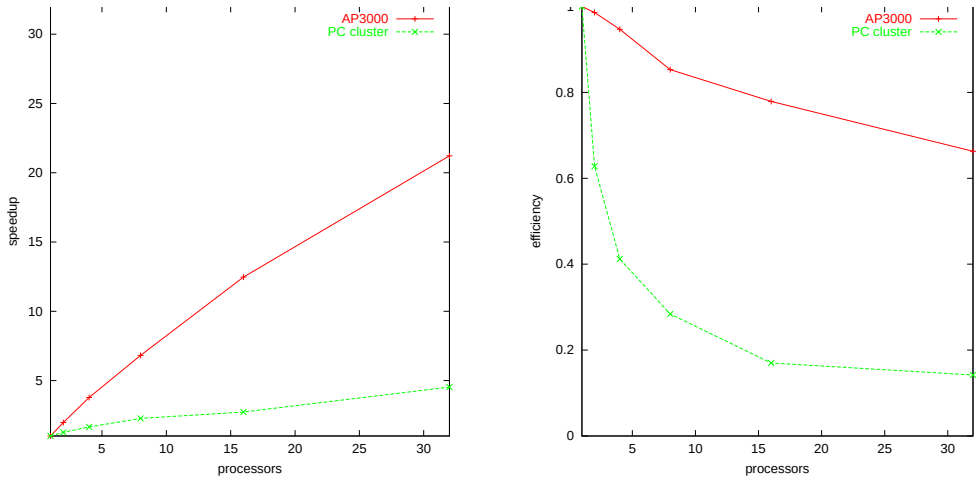

Figure 3: Speedup (left) and efficiency (right) for the FMS model with $k = 7$ on the AP3000 and a PC cluster.

The speedups and efficiencies achieved on the AP3000 are excellent. Solution time on a single AP3000 node is 20 minutes 43 seconds whereas on 32 processors it takes just 58.6 seconds (i.e. 21.22 times faster, an efficiency of 66.3%). With processors that are about 4 times faster and a communication network that is about 6 times slower than the AP3000, and without exclusive access to either processors or the interconnection network, we cannot expect such good results on the PC cluster. However, unusually for problems of this type, reasonable speedups are still achieved, requiring 5 minutes 25 seconds on a single PC and 1 minute 12 seconds on 32 PCs (i.e. 4.53 times faster, an efficiency of 14.2%). The speedup trend for the PC cluster is shallow but linear, suggesting that speedup will continue to improve for an even larger number of processors. Adding extra workstations also boosts solution capacity through additional RAM.

Not only does our distributed algorithm exhibit good scalability but it is also efficient in absolute terms – using a technique based on Laplace transform inversion to calculate the same passage time density requires 1566 seconds (26 minutes 6 seconds) on 32 PCs [2].

6 Conclusion

We have developed a scalable, parallel, uniformization-based algorithm that computes passage time densities in large Markov chains. Key to our scalability is the hypergraph partitioning scheme employed. The method has been validated against simulation and found to be extremely accurate. The capability has been built into the HYDRA tool, thus facilitating the detailed analysis of quality of service in non-trivial high-level models previously considered intractable.

7 Acknowledgements

The authors would like to thank the Imperial College Parallel Computing Centre for the use of the Fujitsu AP3000 supercomputer.

References

[1] P. G. Harrison, "Laplace transform inversion and passage-time distributions in Markov processes," *Journal of Applied Probability*, vol. 27, pp. 74–87, 1990.

[2] P. G. Harrison and W. J. Knottenbelt, "Passage-time distributions in large Markov chains," in *Proceedings of ACM SIGMETRICS 2002*, pp. 77–85, Marina Del Rey, USA, June 2002.

[3] W. Grassman, "Means and variances of time averages in Markovian environments," *European Journal of Operational Research*, vol. 31, no. 1, pp. 132–139, 1987.

[4] A. Reibman and K. Trivedi, "Numerical transient analysis of Markov models," *Computers and Operations Research*, vol. 15, no. 1, pp. 19–36, 1988.

[5] B. Melamed and M. Yadin, "Randomization procedures in the computation of cumulative-time distributions over discrete state Markov processes," *Operations Research*, vol. 32, pp. 926–944, July–August 1984.

[6] J. Muppala and K. Trivedi, "Numerical transient analysis of finite Markovian queueing systems," *Queueing and Related Models, U.N. Bhat, I.V. Basawa (eds.)*, pp. 262–284, 1992.

[7] U. Catalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, pp. 673–693, July 1999.

[8] G. Karypis and V. Kumar, "Multilevel k-way hypergraph parititioning," Tech. Rep. #98-036, University of Minnesota, 1998.

[9] W. J. Knottenbelt, "Generalised Markovian analysis of timed transitions systems," MSc thesis, University of Cape Town, South Africa, July 1996.

[10] W. Knottenbelt, *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Imperial College, London, United Kingdom, February 2000.

[11] W. Knottenbelt, P. Harrison, M. Mestern, and P. Kritzinger, "A probabilistic dynamic technique for the distributed generation of very large state spaces," *Performance Evaluation*, vol. 39, pp. 127–148, February 2000.

[12] D. Deavours and W. Sanders, "An efficient disk-based tool for solving very large Markov models," in *Lecture Notes in Computer Science 1245: Proceedings of the 9th International Conference on Modelling, Techniques and Tools (TOOLS '97)*, (St. Malo, France), pp. 58–71, Springer Verlag, 3–6 June 1997.

[13] W. Knottenbelt and P. Harrison, "Distributed disk-based solution techniques for large Markov models," in *Proceedings of the 3rd International Meeting on the Numerical Solution of Markov Chains (NSMC '99)*, (Zaragoza, Spain), pp. 58–75, September 1999.

[14] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Cambridge, Massachussetts: MIT Press, 1994.

[15] G. Ciardo and K. Trivedi, "A decomposition approach for stochastic reward net models," *Performance Evaluation*, vol. 18, no. 1, pp. 37–59, 1993.