A Preconditioning Method for Stochastic Automata Networks

ABDEREZAK TOUZENE
Department of Computer Science, Sultan Qaboos University
P.O. Box 36, Al-khod 123, Oman.
email: touzene@squ.edu.om

Abstract

In this paper we extend the methodology presented in [1] to develop a preconditioning method to solve Markovian models issued from Stochastic Automata Network modeling (SAN). This method is also based on grouping terms and factorization of the SAN descriptor. Stochastic automata networks have gained a high interest because of the ease of modeling parallel systems and also because of the compact structure of the SAN generator. In this paper, we propose a preconditioning method that uses the compact structure of the SAN.

Keywords: Markovian models; Stochastic automata networks; Preconditioning.

1 Introduction

Stochastic Automata Networks (SAN) is a very powerful modeling tool for complex systems [2] [3] [4], they are particularly useful to model parallel activities, such as concurrent and communicating processes. A stochastic automata network consists of a collection of automaton that may interact each others. An automaton models a specific activity or a component of the whole system under study. Each automaton is represented by a finite number of states and a set of probabilistic transition rules, which define the moves from a state to another. At any given time t, the global state of a stochastic automata network consists of the current state of each one of its compound automaton. In the general case, the SAN descriptor Q has the following form:

$$Q = \bigoplus_i A_i + \sum_j \otimes_i \, Q_i^{(j)}, \qquad (1)$$

where the first part of Q is called local term, and the second part corresponds to synchronization events see [4]. Computing the steady states probability vector needs to solve the following system of equations:

$$\pi Q = 0. \qquad (2)$$

where π is the steady states probability vector of size n and Q the SAN descriptor. Because of the compact form of the descriptor, only iterative methods can be applied to solve the above system of equations. Fortunately, all iterative methods involve the multiplication of the SAN descriptor by a vector which can be efficiently computed without expanding the matrix Q [5]. Iterative methods such as Arnoldi, GMRES [6] [7] can be used in solving the system of equations.

Preconditioning is a very important step for solving the system of equations using iterative methods. This step will improve considerably the convergence rate of the original iterative method. Let us consider the general case, where the system to be solved is

$$Ax = b. \qquad (3)$$

Where A is an $n \times n$ matrix and x of size n. The aim of preconditioning is to modify the original system of equations in order to have and equivalent system with a better distribution of its eigenvalues. In fact, it is well known that the convergence of iterative methods and also the stability of direct methods depends on the distribution of the eigenvalues of the system [8] [9]. Preconditioning the system (3) leads to the modified system

$$M^{-1}Ax = M^{-1}b. \qquad (4)$$

The matrix M^{-1} is called the preconditioner and should approximate the inverse of the matrix A. The problem is how to compute the matrix M^{-1} in an inexpensive way. Traditional approaches to calculate the matrix M^{-1} is to obtain an incomplete factorization of the matrix A,

$$A = LU + E \qquad (5)$$

where L is a lower triangular matrix, U is an upper triangular matrix, and E is the error or a residual matrix. For more details on how the factorization is performed $(ILU(0), ILU(k), ILUTH)$ see [9].

Let us focus now on the SAN descriptor, which has a tensor form. It is clear that the traditional methods cannot be applied to built a preconditioner. We will solve the SAN descriptor using the system of equations (2). The system of equations (2) is equivalent to

$$\pi P = \pi, \tag{6}$$

where $P = (I + Q)$, and I is the identity matrix of dimension n. This system can be solved using an iterative method. Preconditioning the system (6) leads to the following iterative formulation:

$$\pi^{(t+1)} = \pi^t (I - (I - P) M^{-1}). \tag{7}$$

And then,

$$\pi^{(t+1)} = \pi^t - \pi^t (I - P) M^{-1}, \tag{8}$$

where M^{-1} is an approximation of the inverse of the matrix
$M = (I - P)$. The problem is the computation of this inverse taking into account the particular structure of the matrix $(I - P)$. In [10], the inverse of $(I - P)$ is computed as a polynomial series,

$$M^{-1} = \sum_{k=0}^{K} P^k, \tag{9}$$

where K is a given order. This Preconditioner gave a good convergence rate, but unfortunately, the cost of powering of P induces a huge amount of time. In this paper we propose a method which avoid the direct computation of the inverse M^{-1} and transforms this problem to solve a linear system of equations [1]. Indeed, if we denote $y^t = \pi^t(I-P)$, the result x^t of multiplying y^t by the matrix M^{-1} can be provided by solving the following system of equations:

$$x^t M = y^t, \tag{10}$$

where M is kept in its compact form (tensor sums).

Our paper is organized as follows: In section 2 we present the methodology. Section 3 describes the SAN models to be tested. In section 4, we summarize our results and we conclude in section 5.

2 Methodology

To simplify the presentation of our method, we will present first the case where the SAN descriptor is a pure tensor sum (it contains only local terms). Then we will show how to extend the method to the case where the SAN is not a pure tensor sum, which is in fact the more general case.

2.1 SAN Descriptor with Pure Tensor Sums

For clarity reasons, let us consider the case where the SAN descriptor consists only on tensor sum with only two factors : $M = (A_1 \oplus A2)$, where A_1 is of dimension $m \times m$, and A_2 is of dimension $n \times n$. The generalization of this method is very simple and will be discussed later on. We recall that our aim is to solve the system of equations (10). As described in the basic algorithm of Bartels and Stewart [11], the system (10) is equivalent to the system:

$$A_1^T X + X A_2 = Y \tag{11}$$

where X is an $m \times n$ matrix. The matrix X can be seen as a matrix of columns : $X = (x_1, x_2, ..., x_n)$, where x_j denotes the jth column of X. The matrix Y which represents y_t is structured in a similar way. The algorithm is as follows:

Algorithm

1. Compute a unitary transformation U and V such that

$$\bar{A_1} = U^T A_1 U \quad \text{and} \bar{A_2} = V^T A_2 V, \tag{12}$$

where $\bar{A_1}$ and $\bar{A_2}$ are upper triangular matrices obtained using a QR algorithm.

2. Transform the original system (11) using step 1 as follows:

$$\bar{A_1}^T \bar{X} + \bar{X} \bar{A_2} = \bar{Y} \tag{13}$$

where

$$\bar{X} = U^T X V$$

and

$$\bar{Y} = U^T Y V.$$

3. The first column of $\bar{X}$ is given by solving the following lower triangular system:

$$(\bar{A_1}^T + \bar{A}_{211} * I)\bar{x_1} = \bar{y_1}. \tag{14}$$

4. The computation of the k^{th} column of the system is given by:

$$(\bar{A_1}^T + \bar{A}_{2kk} * I)\bar{x_k} = \bar{y_k} - \sum_{i=1}^{k-1} \bar{A}_{2ik} * \bar{x_i}. \tag{15}$$

5. Form the solution X using $\bar{X}$: $X = U \bar{X} V^T$.

The generalization of this method is based on a recursive solving process using the previews steps as follows:

Any system $x_t(A_1 \oplus A_2 ... \oplus A_N) = y_t$, where the matrices A_i are of dimension $n_i, i = 1..N$, can be decomposed as $x_t(A \oplus A_N) = y_t$. This leads to solve the system of equations (10) including the matrix A which is a tensor sum. We apply the recursion till reaching a matrix with only two matrices. It is easy to see that the cost of solving the above system of equations (excluding the QR decomposition of the matrices) is given by

$$Cost = \frac{5}{2}(n_1 n_2 ... n_N)(n_1 + n_2 + ... + n_N).$$

In general, matrices $A_i, i = 1..N$ are small matrices. The cost of their QR decomposition is negligible and hopefully they are calculated only one time for all iterations. The cost to solve the preconditioned system is then equal to 2.5 time the cost of the multiplication vector by the tensor sum.

Let us focus now on practical implementation issues. Since M is singular (generator), the inverse of M does not exist. The idea is to alter slightly the system (13) to overcome the singularity problem. One sure way to alter M lies within the following property of the tensor sum: $(A_1^T + \alpha I)X + X(A_2 - \alpha I) = Y$, for any real number α. The resulting system is equivalent to the first one, but the singularity is still present. The second way is to apply a shift to $M = (A_1 \oplus A_2 + \alpha I)$. The new system is not equivalent, but fortunately any shift combined with the power method converge to the right solution.

2.2 SAN Descriptor with Synchronization Terms

We recall that the SAN descriptor has the form given in formula (1). In general, it has two parts, the local terms, which are pure tensor sums, plus the synchronization terms, which are tensor products. In this case our algorithm cannot be applied directly, since it handles only pure tensor sums. One solution is to consider only the local terms as a preconditioner. This leads to lose the effects of the synchronizations in the preconditioner. We recall that in this case according to formula (5), the residual matrix E of the preconditioner will be the sum of all synchronization terms. A more accurate preconditioner is to add as much as possible the synchronization terms, in such a way that the preconditioner remains a pure tensor sums and in this case the residual matrix E will be of smaller magnitude. Intuitively, we will capture the effect of some synchronizations in the preconditioner. This may be possible only if the synchronization events affect only few automata in the stochastic automata network. This assumption is valid for modeling distributed and parallel systems.

Now we focus on how to add the effects of the synchronization in the preconditioner resulting in a pure tensor form preconditioner: The idea is to group the automata that are affected by the same synchronization events using tensor algebra. This will result in reducing the number of terms but within the terms, matrices will be of higher size.

In the next section we describe the models to be tested and we will show using a specific example how to group and factorize the preconditioner using the tensor algebra properties in order to have an accurate preconditioner.

3 Examples of SAN Models

3.1 Resource Sharing Problem

In this model, N processes share P resources, $P \leq N$. Each process $i = 1..N$, has two states: *using* the resource or *idle*. We denote λ_i the resource acquisition rate and μ_i the freeing rate of a resource from the process i. Notice that the number of processes which may access concurrently to the resource is limited to P. When a process is willing to acquire a resource and find that P processes have already got the resource, it will fail and stay in its *idle* state. In case $P = 1$, the model is equivalent to the usual mutual exclusion problem. The SAN descriptor will have the following form: $Q = \bigoplus_{i=1}^{N} A_i$ (pure tensor sums), where A_i is

$$A_i = \begin{pmatrix} -\lambda_i f & \lambda_i f \\ \mu_i & -\mu_i \end{pmatrix}$$

Each automaton contains a functional variable f evaluated to 0 or 1 depending on the availability of a resource or not. For our experiments (Matlab), the descriptor matrix Q is factorized in two factors $Q = Q_1 \oplus Q_2$. Concerning the preconditioner, for simplicity reasons we evaluate all the functions $f = 1$.

3.2 Alterned Bit Protocol

This model concerns the well-known network protocol named alterned bit protocol. This protocol is modeled by a SAN in [3] as follows: It contains four automata :

- Sender automaton, named S, which have four states namely :PrepareMess0, WaitAck0, PrepareMess1, WaitAck1.
- Message automaton, named M, which have four states namely :WaitMess0, Mess0Inline, WaitMess1, Mess1Inline.
- Acknowledge automaton, named A, which have four states namely :WaitAck0, Ack0Inline, WaitAck1, Ack1Inline.
- Receiver automaton named R, which have four states namely :WaitMess0, PrepareAck0, WaitMess1, PrepareAck1.

The descriptor of this model contains nine terms, one term called local term and eight terms called synchronized terms. The local term consists of the local views of each automaton. It does not contains the interaction between all automata. The synchronization terms represent the effect of a synchronization event on the different automaton of the SAN. For each Synchronization event correspond one term. In the following we construct each term of the descriptor of SAN.

1. The local term consists of the matrices L_S, L_M, L_A and L_R, where

$$L_S = \begin{pmatrix} 0 & 0 & 0 & 0 \\ tm & -tm & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & tm & -tm \end{pmatrix},$$

$$L_M = \begin{pmatrix} 0 & 0 & 0 & 0 \\ lm & -lm & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & lm & -lm \end{pmatrix},$$

$$L_A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ la & -la & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & ta & -ta \end{pmatrix},$$

$$L_R = \begin{pmatrix} -ta & 0 & 0 & ta \\ 0 & 0 & 0 & 0 \\ 0 & ta & -ta & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

$$termloc = L_S \oplus L_M \oplus L_A \oplus L_R.$$

2. The term Send-Message1 contains the matrices $Sm1^{(S)}$ and $Sm1^{(M)}$:

   $TermSendMess1 = Sm1^{(S)} \otimes Sm1^{(M)} \otimes I_4 \otimes I_4$.

   This synchronization affects only the automata S and M.

3. The term Send-Mess0 contains the matrices $Sm0^{(S)}$ and $Sm0^{(M)}$:

   $TermSendMess0 = Sm0^{(S)} \otimes Sm0^{(M)} \otimes I_4 \otimes I_4$.

   This synchronization affects only the automata S and M.

4. The term Send-Ack1 contains matrices $Sack1^{(A)}$ and $Sack1^{(R)}$:

   $TermSendAck1 = I_4 \otimes I_4 \otimes Sack1^{(A)} \otimes Sack1^{(R)}$

   This synchronization affects only the automata A and R.

5. The term Send-Ack0 contains matrices $Sack0^{(A)}$ and $Sack0^{(R)}$:

   $TermSendAck0 = I_4 \otimes I_4 \otimes Sack0^{(A)} \otimes Sack0^{(R)}$

   This synchronization affects only the automata A and R.

6. The term Receive-Mess1 contains the matrices $Srm1^{(M)}$ and $Srm1^{(R)}$

   $TermRecMess1 = I_4 \otimes Srm1^{(M)} \otimes I_4 \otimes Srm1^{(R)}$

   This synchronization affects only the automata M and R.

7. The term Receive-Mess1 contains the matrices $Srm0^{(M)}$ and $Srm0^{(R)}$

   $TermRecMess0 = I_4 \otimes Srm0^{(M)} \otimes I_4 \otimes Srm0^{(R)}$

   This synchronization affects only the automata M and R.

8. The term Receive-Ack1 contains the matrices $Srack1^{(S)}$ and $Srack1^{(A)}$

   $TermRecAck1 = Srack1^{(S)} \otimes I_4 \otimes Srack1^{(A)} \otimes I_4$

   This synchronization affects only the automata S and A.

9. The term Receive-Ack0 contains the matrices $Srack0^{(S)}$ and $Srack0^{(A)}$

   $TermRecAck0 = Srack0^{(S)} \otimes I_4 \otimes Srack0^{(A)} \otimes I_4$

   This synchronization affects only the automata S and A.

The SAN descriptor of this model is the sum of all the above terms. For our preconditioner, we will factorize and group as much term as possible in order to obtain the preconditioning matrix M of the form $M = Q_1 \otimes Q_2$ and hence our method will be directly applicable. First, let us group all synchronized terms that have a synchronized effect on the same automata. In the second step, add them if possible to the pure tensor sum using some simple tensor factorization operations (for more details on tensor algebra see [12]. According to the effect of the synchronization on the automata network as described above, we group automata S ,M to form a macro-automaton and we group automata A, R to form another macro-automaton. This will result in the following:

$$Q_1 = (L_S \oplus L_M) + (Sm1^{(S)} \otimes Sm1^{(M)})$$
$$+(Sm0^{(S)} \otimes Sm0^{(M)})$$

and

$$Q_2 = (L_A \oplus L_R) + (SAck1^{(A)} \otimes SAck1^{(R)})$$

$$+(SAck0^{(A)} \otimes SAck0^{(R)}).$$

The other terms:

$$E = TermRecMess1 + TermRecMess0 +$$

$$TermRecAck1 + TermRecAck0,$$

cannot be added to the preconditioner (without a loss of its pure tensor sums property), their effect will be lost. We may think of them as a residual matrix of an incomplete factorization see equation (5), and we hope that their effect is very small.

4 Experimental Results

For testing our preconditioning method, we chose for simplicity reasons the power iterative method. In our experiments we considered a shift $\alpha = 0.25$ to ensure the non-singularity of the preconditioner. The following tables summarizes the number of iterations using the power method and the preconditioned power method. The solution precision is 10^{-10} decimals. For the first table, all the parameters $\lambda_i, i = 1..8$ are equal to a given value λ and similarly $\mu_i = \mu, i = 1..8$. We recall that P is the number of resources.

P	Param.	Iter. Power	Iter. Precond.
1	$\lambda = 1, \mu = .5$	18	10
	$\lambda = 1, \mu = .9$	13	3
4	$\lambda = 1, \mu = .5$	91	19
	$\lambda = 1, \mu = .9$	109	25
6	$\lambda = 1, \mu = .5$	77	22
	$\lambda = 1, \mu = .9$	319	30
8	$\lambda = 1, \mu = .5$	97	32
	$\lambda = 1, \mu = .9$	196	33

The second experimental study concerns the model of the alterned bit protocol. For this model we consider four cases as follows:

- 1: $tm = 2, lm = 2, la = .1, ta = .1, us = 2, ur = 1, ue = 2, uc = 1$.
- 2: $tm = 4, lm = 4, la = .1, ta = .1, us = 4, ur = 1, ue = 4, uc = 1$.
- 3: $tm = .1, lm = .1, la = .1, ta = .1, us = .1, ur = 1, ue = .1, uc = 1$.
- 4: $tm = .1, lm = .1, la = .1, ta = .1, us = .1, ur = .1, ue = .1, uc = .1$.

us is a transition rate in the matrix $Sm0^{(S)}$and $Sm1^{(S)}$. ur is a transition rate in the matrix $Srm0^{(R)}$and $Srm1^{(R)}$. ue is a transition rate in the matrix $Srack0^{(R)}$and $Srack1^{(R)}$. uc is a transition rate in the matrix $Srack0^{(S)}$and $Srack1^{(S)}$.

Case.	Iter. Power	Iter. Precond.
1	1240	185
2	2189	293
3	77	22
4	787	146

These two tables show clearly the good performance of our preconditioning method. Concerning the number of iterations to acheive convergence, this method is nine time faster than the simple power method. If we consider the time cost, our method will be almost 4 times faster than the simple power method.

5 Conclusion

In this paper we contributed in developing a preconditioning method for solving stochastic automata network models. This method has shown a very good results on the tested models and it can be easily adapted to other iterative methods, known to converge very quickly such as the iterative GMRES for SAN models. In our future work we will investigate a general and automatic procedure for grouping automata (affected by the same synchronization events) and factoring the precondioner as a pure tensor sum with the smallest residual matrix possible.

Acknowledgment

The author thanks Prof. B. Plateau and Prof. W. J. Stewart for their help and guidance in conducting this work.

References

[1] G.W Stewart. Stochastic Automata, Tensors Operation, and Matrix Formulas. *Technical Report, University of Maryland, UMIACS TR-96-11 CMSC TR-3598, Jan. 1996.*

[2] B. Plateau. On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms. *Proc. ACM Sigmitrics Conference on Measurement and Modeling of Computer Systems, Austin, Texas, Aug. 1985.*

[3] K. Atif. Modelisation du Parallelism et de la Synchronisation. *These de Docteur de l'Institut National Polytechnique de Grenoble, France, Sep. 1991.*

[4] K. Atif, B. Plateau. Stochastic Automata Network for Modeling Parallel Systems. *IEEE Trans. On Soft. Eng., 17, 10, Oct. 1991 .*

[5] P. Fernandes, B. Plateau, W. J. Stewart. Efficient descriptor-vector multiplication in stochastic automata networks. *J. ACM, (3), 381-414, 1998.*

[6] Y. Saad. Krylov Subspace Methods for Solving Unsymetric Linear Systems. *Mathematics of Computation, 37, 105-126.*

[7] Y. Saad, M.H. Shultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput., 7, 856-869, 1986 .*

[8] R. S. Varga. Matrix Iterative Analysis. *Printice Hall, Englewood Cliffs, N.J, 1963.*

[9] W. J. Stewart An Introduction to the Numerical Solution of Markov Chains. *Princeton University Press, NJ. 1994.*

[10] K. Atif, B. Plateau, and W. Stewart. The numerical solution of stochastic automata network. *European Journal of Operation Research, 86(3), Nov. 1995.*

[11] R. H. Bartels and G. W. Stewart. Algorithm 432: The solution of the matrix equation AX-BX=C. *Communication of ACM, 8:820-826, 1972.*

[12] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Trans. Comp., Vol C-30, No 2, Feb. 1981.*

Biography

ABDEREZAK TOUZENE received an undergraduate degree in computer science from university USTHB of Algiers in 1987 ,M.Sc. from university Paris sud in 1988 and a Ph.D. degree in computer science from Institut Polytechnique de Grenoble (France) in 1992. Dr. Touzene is currently assistant professor in the department of computer science, Soltan Qaboos university in Oman. His area of interest include performance evaluation, interconnection networks and parallel computing. He is a member of IEEE.