Reduced Markov Models of Parallel Programs with Replicated Processes

Markus Siegle
Universität Erlangen–Nürnberg, IMMD VII, Martensstraße 3, 91058 Erlangen, Germany
siegle@informatik.uni-erlangen.de

The behaviour of parallel programs with replicated processes is modelled by continuous time Markov chains. For high-level model description, we use stochastic automata networks. We are particularly interested in models of parallel programs which exhibit symmetries. Exact lumpability is exploited for reducing the model's state space, thus making complex models of large real systems computationally tractable. Complexity analysis of a new reduction algorithm shows that the method can be applied efficiently in practice.

Keywords: Parallel Programming, Markov Models, Lumpability, State Space Reduction

1. Introduction

When developing solutions for real application problems on modern parallel machines, the programmer is faced with a large number of questions. Many degrees of freedom exist during the design of a suitable parallel algorithm and during its subsequent implementation for a class of machines. The writing of parallel programs is a very cumbersome and time-consuming process since environments for parallel software engineering have not yet reached a satisfying degree of maturity. It is usually prohibitive for the programmer to code several possible alternatives in order to decide which one performs best. Among existing implementation alternatives the most promising has to be chosen. This choice can be greatly supported by the use of models for predicting the behaviour of a parallel program.

Markov models have been widely used to study performance issues in computer systems. They provide a framework for describing stochastic behaviour while being amenable to analytic solution. The most eminent problem in Markovian modelling is the state space explosion which often makes large models intractable. State spaces are often so large that they cannot be stored in the memory of available computers, and the solution of a Markov chain with a very large state space can consume an excessive amount of computation time. In this paper, we discuss an approach to overcoming this problem in the context of modern parallel systems containing a number of similar components. This is motivated by the fact that many parallel programs consist of one or more sets of similar processes. Several processes of such replication-type parallel programs are running in parallel on different nodes of the machine. While processing different data they obey the same stochastic rule.

For the specification of Markovian models different formalisms can be employed. High-level description techniques include queueing networks [13] and stochastic Petri nets [1]. Tools exist which support comfortable model design through graphical or textual user interfaces [17, 7, 9, 15]. They perform the translation of the high-level description into the underlying Markov chain, its solution and the computation of performance measures. In this paper we use stochastic automata networks (SANs) for model description and we restrict ourselves to the discussion of models with continuous time scale. An in-depth description of the SAN formalism can be found in [4] and [3] (the latter focuses on discrete time scale SAN models).

Section 2 includes a brief survey of state-of-the-art techniques for structured model description. In the following two Sections, a novel approach to the reduction of the state space of SAN models is presented. It is based on the concept of Markov chain lumpability and can be conveniently applied to SAN models of parallel programs with replicated processes. An algorithm for the efficient computation of the matrices describing the reduced SAN model is discussed in Section 5.

2. Survey of Techniques for Structured Model Description

There is a large number of approaches which fall into the structured model description category. In [14], Plateau showed how interdependent stochastic automata can be combined to a joined model. Tensor algebra is used to obtain the structure of the combined generator matrix. It is shown that an iterative solution technique (power method) can be applied to solve the combined model, without explicitly generating the joined generator matrix. A similar approach is described by Donatelli [11] for superimposed stochastic automata, a special class of stochastic Petri nets. Balbo et al. described a technique in which queueing networks and GSPNs are combined for solving complex models [2]. A similar approach is followed by Buchholz [6] in the hierarchical multi-paradigm modelling approach. The joined model consists of a number of low level models which can be specified by multi-class queueing networks or coloured stochastic Petri nets, and a high

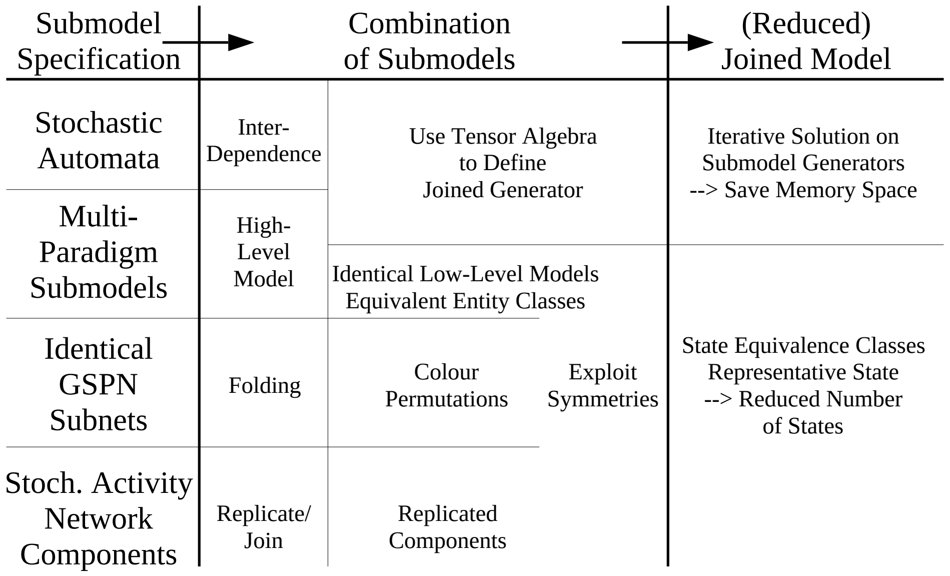
Submodel Specification
Combination of Submodels
(Reduced) Joined Model
Stochastic Automata
Inter-Dependence
Use Tensor Algebra to Define Joined Generator
Iterative Solution on Submodel Generators --> Save Memory Space
Multi-Paradigm Submodels
High-Level Model
Identical Low-Level Models Equivalent Entity Classes
Identical GSPN Subnets
Folding
Colour Permutations
Exploit Symmetries
State Equivalence Classes Representative State --> Reduced Number of States
Stoch. Activity Network Components
Replicate/ Join
Replicated Components

Figure 1: Structured Model Description: Common Features

level model which describes the flow of entities (customers or tokens) between the low level models. Buchholz also uses tensor algebra to obtain the combined generator matrix. He shows that beside the power method, other iterative solution techniques (Jacobi and Gauss-Seidel) can be applied based on the submodel generator matrices. In a recent paper, this technique is extended to the exploitation of symmetries in order to reduce the cardinality of the state space [5]. Symmetry exploitation is also the basic idea which led to the use of coloured Petri nets for the purpose of performance evaluation [8]. A similar technique has been described for another class of stochastic Petri nets – stochastic activity networks – by Sanders et al. [16]. Here also, the exploitation of symmetries allows to solve the model efficiently.

The common features of different structured model description techniques are pointed out by the table in Fig1. The first column states the technique which is used for submodel specification. The second column is divided into two subcolumns: The technique for combining the submodels is given, and special features which are used during the combination process are mentioned. Properties of the resulting joined model and of its solution method are listed in the third column.

It is common to all approaches that submodels are first specified individually, and then combined in order to obtain the joined model. Different mechanisms are used to specify the way in which the combination takes place. For stochastic automata, the combination is determined by the interdependences which exist between the components. In the hierarchical multi-paradigm modelling approach, the high-level model specifies how low-level models are combined. Using coloured GSPNs, submodels cannot be identified quite as easily: They can be seen as identical subnets of a GSPN model, where the interaction is given by the arcs and places between them. To build the coloured GSPN, these identical subnets are folded together, and thereafter only distinguishable by colour domains associated with them. Stochastic activity network components can be combined using two operations: There is the replicate operation to generate n instances of one component, where a subset of distinguished places is not replicated, i.e. is common to all n instances, and there is the join operation, to join two components by merging two subsets of places, one in each component.

For combined stochastic automata and the hierarchical multi-paradigm modelling approach, the generator matrix of the joined model is given as an expression, in which the matrices describing the behaviour of the components are combined by tensor operations. Knowledge about the structure of the joined generator matrix is sufficient in order to apply iterative solution techniques for obtaining the steady-state solution of the model, i.e. the joined generator matrix does not have to be created explicitly. Since the joined generator is often a very large sparse matrix, this helps to save a lot of storage space. A second important advantage may be seen by the fact that this kind of iterative solution technique, based on the components' generator matrices, is amenable to parallel processing.

For the hierarchical multi-paradigm modelling approach, coloured GSPNs and stochastic activity networks, it has been shown how the exploitation of model symmetries may reduce the cardinality of the state space dramatically. In the former, symmetries may be present due to a number

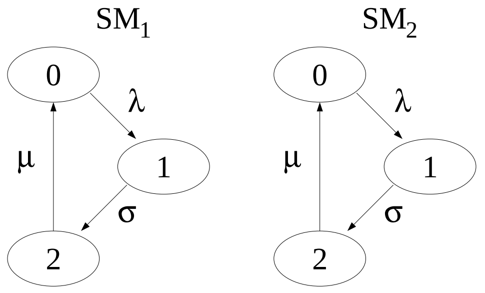

Figure 2: SAN of two independent processes

of identical low-level models, or to the identical behaviour of different entity classes [5]. Entities are either customers (in queueing network submodels) or tokens (in GSPN submodels). In coloured GSPNs, it is also the identical behaviour of different token classes which enables reduction. Here symmetries are recognized by permuting these token classes. Using stochastic activity networks, symmetries are present wherever the replicate operation is used. Symmetries lead to the partition of the state space into classes of equivalent states, and it suffices to choose one state from each equivalence class – usually determined by lexicographical ordering – to represent this class. The steady-state probability of the representative state in the reduced joined model is equal to the sum of the steady-state probabilities in its equivalence class in the original model.

3. Component Replication and State Space Reduction: An Example

We introduce our own ideas with a very simple example: Consider an overall model consisting of two submodels SM_1 and SM_2. The two are similar independent processes which move through states 0,1 and 2 in a cyclic fashion as illustrated in Fig. 2.

The state transitions take place after the elapse of a time which is exponentially distributed with rate λ, σ or μ. The state transitions are described by the infinitesimal generator matrices $Q^{(1)}$ and $Q^{(2)}$ of SM_1 and SM_2 which are given by

$$Q^{(1)} = Q^{(2)} = \begin{bmatrix} -\lambda & \lambda & 0 \\ 0 & -\sigma & \sigma \\ \mu & 0 & -\mu \end{bmatrix}$$

We are interested in the combined stochastic process of SM_1 and SM_2. This process has $3 \times 3 = 9$ states, each state corresponding to a tuple (s_1, s_2), where s_i denotes the state of SM_i. It is known [4] that its generator matrix Q is given by

$$Q = Q^{(1)} \oplus Q^{(2)}$$

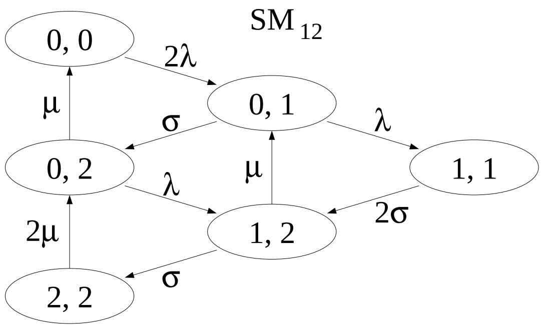

Figure 3: Reduced SAN

where the operator $\oplus$ denotes the tensor sum. For the definition of tensor operations the reader is referred to the Appendix and to [10].

For the computation of the steady-state probability vector π, the linear system of equations $\pi Q = 0$ has to be solved. This is usually done using iterative solution techniques. It is not necessary to construct the matrix Q explicitly. Instead, iteration can be performed on the tensor description of Q [4].

Let us now assume that there is a larger number n of such similar independent processes. While the generator Q of the overall model can still be described in a compact manner by $Q = \oplus_{i=1}^{n} Q^{(i)}$, the number of states is now 3^n. Already for modest values of n, this leads to an overall model with a very large state space. However, due to the symmetry of the SAN we can identify sets of equivalent states which have the same steady-state probability. The idea of symmetry exploitation is to combine similar submodels and to replace them by a reduced model. During the construction of the reduced model, only one state is chosen as a representative for each set of equivalent states. For example, in the case $n = 2$, states (0, 1) and (1, 0) constitute a pair of equivalent states. Fig. 3 shows the reduced model in which this pair is represented by the single state (0, 1). The pairs (0, 2), (2, 0) and (1, 2), (2, 1) are treated similarly such that the reduced model has only 6 states instead of $3^2 = 9$.

We continue with an extension of the same example in order to demonstrate how different SAN submodels can interact. Fig. 4 shows a SAN consisting of 3 submodels. SM_0 represents a resource which can be either idle or busy. SM_1 and SM_2 have the same structure as before. They represent two processes which access the common resource under the mutual exclusion discipline. Each of the two processes behaves in a cyclic manner: Having done some work it performs a synchronization in order to obtain the resource, accesses the resource and goes back to work.

The transition from *work* to *sync* is still a purely local event. It does not depend on the other submodels, nor does it affect any of the other submodels. The transitions

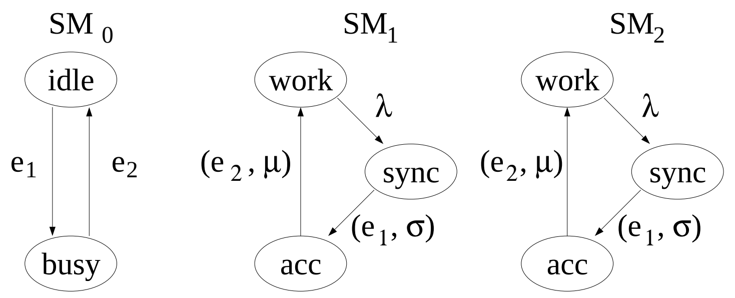

Figure 4: SAN of two Processes Accessing a Common Resource with Mutual Exclusion

from *sync* to *acc* and from *acc* to *work* are labelled e_1 and e_2 and force a synchronization with the same transitions in SM_0.

The infinitesimal generator matrix Q of the overall model can still be constructed from matrices associated with the submodels and by use of tensor algebra [4]. In our example, the matrices $Q_l^{(i)}, Q_{e_k}^{(i)}, Q_{e_k n}^{(i)}$ with $i \in \{0,1,2\}$ and $k \in \{1,2\}$ are defined. Matrices $Q_l^{(i)}$ contain transitions local to SM_i. Since SM_0 does not have any local transitions $Q_l^{(0)}$ is an identity matrix.

$$Q_l^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Q_l^{(1)} = Q_l^{(2)} = \begin{bmatrix} -\lambda & \lambda & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Matrices $Q_{e_k}^{(i)}$ describe the synchronization of submodels by the event e_k, and matrices $Q_{e_k n}^{(i)}$ are needed to correct the diagonal entries of Q accordingly. For the event e_1 we have

$$Q_{e_1}^{(0)} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \; Q_{e_1 n}^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$Q_{e_1}^{(1)} = Q_{e_1}^{(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \sigma \\ 0 & 0 & 0 \end{bmatrix}$$

$$Q_{e_1 n}^{(1)} = Q_{e_1 n}^{(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and the matrices for event e_2 are given by

$$Q_{e_2}^{(0)} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \; Q_{e_2 n}^{(0)} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Q_{e_2}^{(1)} = Q_{e_2}^{(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu & 0 & 0 \end{bmatrix}$$

$$Q_{e_2 n}^{(1)} = Q_{e_2 n}^{(2)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \mu \end{bmatrix}$$

The generator matrix Q of the overall model is then given by

$$Q = \oplus_{i=0}^{2} Q_l^{(i)} + \\ \Sigma_{k=1}^{2} \left(Q_{e_k}^{(0)} \otimes \left(Q_{e_k}^{(1)} \oplus Q_{e_k}^{(2)} \right) - Q_{e_k n}^{(0)} \otimes \left(Q_{e_k n}^{(1)} \oplus Q_{e_k n}^{(2)} \right) \right)$$

In the inner parentheses of this expression, the tensor sum operator (not the tensor product) has to be used, because either SM_1 or SM_2 – but not both – participates in the synchronization with SM_0.

As in the original example, it is again possible to reduce the combined state space of SM_1 and SM_2. The resulting SAN, in which the combined stochastic process of former SM_1 and SM_2 is represented by SM_{12}, is shown in Fig. 5. It leads to an overall model which has only $2 \times 6 = 12$ states instead of $2 \times 3 \times 3 = 18$. It is clear that not all of these states are actually reachable because of the synchronization constraints. In particular, all states in which SM_1 and SM_2 are both in state *acc* are unreachable.

4. Lumpability and Symmetry Exploitation

In this section we will look at a general SAN framework and give a formal description of the reduction of a number of similar SAN submodels.

Suppose we have a SAN consisting of n submodels $SM_1, \ldots, SM_n$. Furthermore, let this set of submodels be the union of c classes such that all similar submodels belong to the same class, i.e. a class contains multiple copies of the same submodel. Let class i contain n_i submodels, each having s_i states. We have the relation $\Sigma_{i=1}^{c} n_i = n$, and the number of states s of the overall model is given by $s = \Pi_{i=1}^{c} s_i^{n_i}$.

Let us now focus our attention to class i, and without loss of generality assume that the submodels in this class are denoted $SM_1, \ldots, SM_{n_i}$. The combined state space S_i of all submodels of class i has $s_i^{n_i}$ states, where every state is an n_i-tuple with elements from $\{0, \ldots, s_i - 1\}$. We consider two states $x = (x_1, \ldots, x_{n_i})$ and $y = (y_1, \ldots, y_{n_i})$ equivalent if y is a permutation of x, i.e.

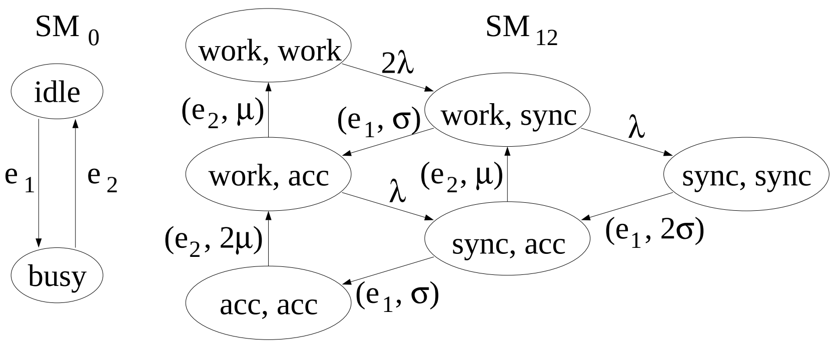

Figure 5: Reduced SAN for the Mutual Exclusion Model

$$\hat{Q} = UQV = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Sigma & \lambda & & \lambda & & & & & \\ & \Sigma & \sigma & & \lambda & & & & \\ \mu & & \Sigma & & & \lambda & & & \\ & & & \Sigma & \lambda & & \sigma & & \\ & & & & \Sigma & \sigma & & \sigma & \\ & & & \mu & & \Sigma & & & \sigma \\ \mu & & & & & & \Sigma & \lambda & \\ & \mu & & & & & & \Sigma & \sigma \\ & & \mu & & & & \mu & & \Sigma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \Sigma & 2\lambda & & & & \\ & \Sigma & \sigma & \lambda & & \\ \mu & & \Sigma & & \lambda & \\ & & & \Sigma & 2\sigma & \\ & \mu & & & \Sigma & \sigma \\ & & 2\mu & & & \Sigma \end{bmatrix}$$

Figure 6: The full structure of the matrix $\hat{Q}$

if there exists a permutation matrix P_i of dimension n_i such that $y = xP_i$. The state space S_i is now partitioned in such a way that all equivalent states are in the same partition. This partition is clearly an equivalence relation. The construction guarantees that the sum of the transition rates from a given state to all states in another partition is the same for all states within the same partition. Therefore the Markov process with state space S_i is lumpable [12] with respect to this partition. A reduced model can then be built which contains only one state per equivalence class.

To illustrate this concept, we return to the first example (from Fig. 2) where the overall model consists of only one class of submodels, class 1. In this class we have two similar submodels each having 3 states, i.e. $n_1 = 2$ and $s_1 = 3$. The generator matrix $Q^{(1)}$ is associated with each submodel in this class 1. The combined state space S_1 of class 1 is given by the set of tuples $S_1 = \{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)\}$. This set is partitioned in the following way: $S_1 = \{(0,0)\} \cup \{(0,1),(1,0)\} \cup \{(0,2),(2,0)\} \cup \{(1,1)\} \cup \{(1,2),(2,1)\} \cup \{(2,2)\}$.

The state space S_1 is lumpable with respect to this partition and the lumped generator matrix $\hat{Q}$ for class 1 can be computed with the help of a projection matrix V and a selection matrix U [12]. It is given by

$$\hat{Q} = UQV = U\left(Q^{(1)} \oplus Q^{(1)}\right)V$$

The product $Q^{(1)} \oplus Q^{(1)}$ is a 9×9 matrix and matrices U and V have dimensions 6×9 and 9×6. The full structure of matrix $\hat{Q}$ is shown in Fig. 6.

Entries Σ denote negative row sums. By multiplying Q and the projection matrix V, columns corresponding to equivalent states are added. In the resulting matrix QV, all rows corresponding to equivalent states are equal. Therefore, for every equivalence class, only one such row is chosen by the selection matrix U.

This reduction technique works also in the presence of submodel synchronization. Returning to the example of

Fig. 4, the matrices describing the behaviour of the reduced model of SM_1 and SM_2 can be computed as

$$\hat{Q}_l^{(12)} = U\left(Q_l^{(1)} \oplus Q_l^{(2)}\right)V$$
$$\hat{Q}_{e_k}^{(12)} = U\left(Q_{e_k}^{(1)} \oplus Q_{e_k}^{(2)}\right)V$$
$$\hat{Q}_{e_kn}^{(12)} = U\left(Q_{e_kn}^{(1)} \oplus Q_{e_kn}^{(2)}\right)V,\ k \in \{1,2\}$$

The reduced overall generator matrix is then given by

$$\hat{Q} = Q_l^{(0)} \oplus \hat{Q}_l^{(12)} + \Sigma_{k=1}^{2}\left(Q_{e_k}^{(0)} \otimes \hat{Q}_{e_k}^{(12)} - Q_{e_kn}^{(0)} \otimes \hat{Q}_{e_kn}^{(12)}\right)$$

In general, the structure of matrices U and V depends only on s_i and n_i. Their dimensions are $C_{s_i}^{n_i} \times s_i^{n_i}$ and $s_i^{n_i} \times C_{s_i}^{n_i}$ where $C_{s_i}^{n_i} = \binom{n_i+s_i-1}{s_i-1}$. Lumping equivalent states in the combined stochastic process of class i reduces the state space of this process from $s_i^{n_i}$ to $\binom{n_i+s_i-1}{s_i-1}$.

5. The Fast Symmetric Lumping Algorithm

In this Section we discuss the problem of building the reduced model efficiently in practice. We present an algorithm for computing the matrices describing the reduced Markov chain of submodel class i. The complexity of the scheme will be analyzed and compared to the complexity of the algorithm to compute the non-reduced tensor descriptor.

The following is a new algorithm to compute matrices of the type $\hat{Q}^{(i)} = U\tilde{Q}^{(i)}V = U\left(\bigotimes_{j=1}^{n_i} Q^{(i)}\right)V$ "on the fly", i.e. directly from the submodel matrix $Q^{(i)}$ without explicitly computing the tensor product inside the parentheses. Matrices U and V do not appear explicitly in the algorithm. The idea is to compute only those rows of $\tilde{Q}^{(i)}$ which will be selected by the left-multiplication with the selector matrix U. Within each row, the projection of all symmetric states onto each other (usually achieved by the right-multiplication with the projection matrix V) is carried out immediately. In order to be able to do this, for each matrix entry the algorithm keeps track of the state description of the originator state $(o_1, \ldots, o_{n_i})$ and of the target state $(t_1, \ldots, t_{n_i})$. Incrementing the state description is done in a special way by the function **increment_ordered()**, which follows the lexicographical ordering but skips those states $(o_1, \ldots, o_{n_i})$ where the condition $o_1 \leq \ldots \leq o_{n_i}$ is violated. Elements of $\tilde{Q}^{(i)}$ are given by

$$\tilde{Q}^{(i)}((o_1, \ldots, o_{n_i}), (t_1, \ldots, t_{n_i})) = \prod_{j=1}^{n_i} Q^{(i)}(o_j, t_j)$$

Therefore the entries of $\hat{Q}^{(i)}$ are computed as

$$\hat{Q}^{(i)}((o_1, \ldots, o_{n_i}), (t_1, \ldots, t_{n_i})) = \sum_{(m_1,\ldots,m_{n_i})=P_i(t_1,\ldots,t_{n_i})} \prod_{j=1}^{n_i} Q^{(i)}(o_j, m_j)$$

where $P_i(t_1, \ldots, t_{n_i})$ denotes a permutation of $(t_1, \ldots, t_{n_i})$. The following is a description of the algorithm in pseudo-code:

```
(1)    row = 1
(2)    $(o_1, \ldots, o_{n_i}) = (1, \ldots, 1)$
(3)    while $((o_1, \ldots, o_{n_i}) \neq (s_i, \ldots, s_i))$
(4)      {col = 1
(5)       $(t_1, \ldots, t_{n_i}) = (1, \ldots, 1)$
(6)       while $((t_1, \ldots, t_{n_i}) \neq (s_i, \ldots, s_i))$
(7)         {$\hat{Q}^{(i)}(row, col) = \sum_{(m_1,\ldots,m_{n_i})=P_i(t_1,\ldots,t_{n_i})} \prod_{j=1}^{n_i} Q^{(i)}(o_j, m_j)$
(8)          increment_ordered($(t_1, \ldots, t_{n_i})$)
(9)          col++
(10)         }
(11)      increment_ordered($(o_1, \ldots, o_{n_i})$)
(12)      row++
(13)     }
```

For the complexity analysis of this algorithm we use the following scheme, which indicates the number of computation steps needed for the respective line.

```
(1)     1
(2)     + $n_i$
(3)     + $C_{s_i}^{n_i}$ *
(4)           {1
(5)           + $n_i$
(6)           + $C_{s_i}^{n_i}$ *
(7)                 {$n_i * s_i^{n_i}/C_{s_i}^{n_i}$

(8)                 + $n_i$
(9)                 + 1
(10)                }
(11)          + $n_i$
(12)          + 1
(13)          }
```

Altogether,

$$1 + n_i + C_{s_i}^{n_i}\left(2 + 2n_i + C_{s_i}^{n_i}\left(1 + n_i + n_i s_i^{n_i}/C_{s_i}^{n_i}\right)\right)$$
$$= 1 + n_i + 2C_{s_i}^{n_i} + 2n_i C_{s_i}^{n_i} + \left(C_{s_i}^{n_i}\right)^2 + n_i\left(C_{s_i}^{n_i}\right)^2 + n_i C_{s_i}^{n_i} s_i^{n_i}$$

steps are required. Here we have used the information, that on the average $s_i^{n_i}/C_{s_i}^{n_i}$ permutations have to be considered for a given target state $(t_1, \ldots, t_{n_i})$.

The non-reduced generator matrix $\tilde{Q}^{(i)}$ can be computed by an algorithm which has similar structure but uses the

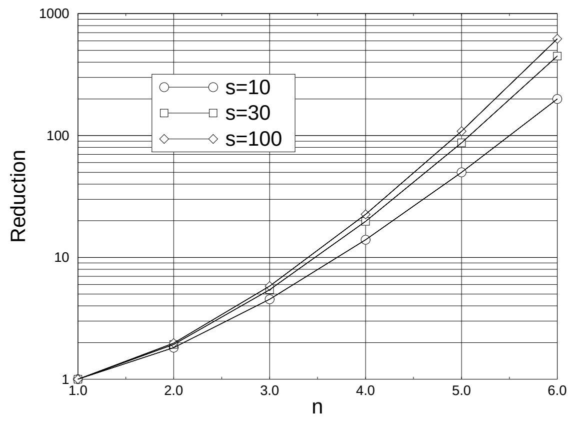

Figure 7: Gain of the Reduction for Various Parameter Sets

ordinary incrementation function and on line (7) employs the expression for $\tilde{Q}^{(i)}((o_1,\ldots,t_{n_i}),(t_1,\ldots,t_{n_i}))$. Its number of computation steps is given by

$$\begin{aligned}&1+n_i+s_i^{n_i}(2+2n_i+s_i^{n_i}(1+n_i+n_i))\\=&1+n_i+2s_i^{n_i}+2n_is_i^{n_i}+s_i^{2n_i}+n_is_i^{2n_i}+n_is_i^{2n_i}\end{aligned}$$

Comparing the dominant terms of both complexities, it can be observed that the construction of the reduced generator is cheaper than the construction of the non-reduced generator by at least a factor of $s_i^{n_i}/C_{s_i}^{n_i}$.

This gain is actually what would be expected. For the computation of one row of the reduced matrix $\hat{Q}^{(i)}$, every entry of matrix $\tilde{Q}^{(i)}$ has to be considered (through the permutations), so within a row we do not expect a gain. But since only $C_{s_i}^{n_i}$ rows instead of $s_i^{n_i}$ have to be computed, we expected the above gain. For the illustration of the gain for various parameter sets s_i and n_i we refer to Fig. 7.

The algorithm to compute $\hat{Q}^{(i)} = U\left(\bigoplus_{j=1}^{n_i} Q^{(i)}\right)V$ has the same form, only the expression on line (7) has to be substituted by

$$\hat{Q}^{(i)}(row,col) = \sum_{(m_1,\ldots,m_{n_i})=P_i(t_1,\ldots,t_{n_i})} \Big(\sum_{j=1}^{n_i} Q^{(i)}(o_j,m_j)1(\forall k\neq j : o_k = m_k)\Big)$$

The function $1()$ used here evaluates to either 1 or 0, depending on the condition given in the parentheses. Complexity results for this case are the same as above.

Further optimizations are possible if the sparsity of the matrices is taken into account. In particular, the product on line (7) of the algorithm equals zero if one of the entries $Q^{(i)}(o_j,m_j)$ is zero.

The analysis of complexity shows that the computation of $\hat{Q}^{(i)}$ "on the fly" is already cheaper than the computation of $\tilde{Q}^{(i)}$. This is a very important result. It states that one does not have to pay anything in order to benefit from the reduction of the state space for the subsequent analysis of the Markov chain.

6. Conclusion

The state space reduction achieved by the lumping of equivalent states can be massive! The reduction of the combined state space of similar submodels implies the reduction of the overall model's state space by the same factor. In this way, the overall model benefits fully from the reduction within its classes. Since the sum of the steady-state probabilities of equivalent states is equal to the steady-state probability of the representative state in the reduced model, the technique provides exact results.

The technique presented in this paper makes it possible to carry out model experiments before making final implementation decisions for parallel programs with replicated processes. Modelling helps to avoid expensive programming effort leading to unsuccessful solutions. Even if a model would be intractable using conventional techniques, its evaluation can be made feasible by the state space reduction technique we have described.

References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.

[2] G. Balbo, S. Bruell, and S.Ghanta. Combining Queueing Networks and Generalized Stochastic Petri Nets for the Solution of Complex Models of System Behaviour. *IEEE Transactions on Computers*, 37(10):1251–1268, Oct. 1988.

[3] B.Plateau and K. Atif. Stochastic Automata Network for Modeling Parallel Systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[4] B.Plateau and J.-M. Fourneau. A Methodology for Solving Markov Models of Parallel Systems. *Journal of Parallel and Distributed Computing*, 12:370–387, 1991.

[5] P. Buchholz. Hierarchical Markovian Models - Symmetries and Reduction. In R. Pooley and J. Hillston, editors, *6th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 305–319, Edinburgh, September 1992.

[6] P. Buchholz. Numerical Solution Methods Based on Structured Descriptions of Markovian Models. In G. Balbo and G. Serazzi, editors, *Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 242–258. Elsevier Science Publisher B.V., 1992.

[7] G. Chiola. GreatSPN 1.5 Software Architecture. In G. Balbo and G. Serazzi, editors, *Proceedings of the 5th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Torino, Feburary 1991*, pages 117–132. Elsevier Science Publisher B.V., 1992.

[8] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed Coloured Nets and their Symbolic Reachability Graph. In *Proceedings of the 11th International Conference on Application and Theory of Petri Nets*, pages 387–410, Paris, June 1990.

[9] G. Ciardo and J. Muppala. *Manual for the SPNP Package Version 3.1*. Duke University, October 1991.

[10] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Transactions on Computers*, C-30(2):116–125, February 1981.

[11] S. Donatelli. Superposed Stochastic Automata: a class of Stochastic Petri Nets amenable to parallel solution. In *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models*, pages 54–63, Melbourne, December 1991.

[12] J. Kemeny and J. Snell. *Finite Markov Chains*. Springer, 1976.

[13] L. Kleinrock. *Queueing Systems*, volume 1: Theory. John Wiley & Sons, 1975.

[14] B. Plateau. On the Synchronization Structure of Parallelism and Synchronization Models for Distributed Algorithms. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 147–154, Austin, TX, August 1985.

[15] B. Plateau, J.-M. Fourneau, and K.-H. Lee. PEPS: A Package for Solving Complex Markov Models of Parallel Systems. In *Proceedings of the 4th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 341–360, Palma (Mallorca), September 1988.

[16] W. Sanders and J. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, January 1991.

[17] M. Veran and D. Potier. QNAP2: A Portable Environment for Queueing Systems Modelling. In *Proceedings of the First International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Paris, May 1984.

Appendix

Let $M(r,c)$ be the set of matrices with r rows and c columns. The tensor product (Kronecker product) of two matrices $A \in M(r_A, c_A)$ and $B \in M(r_B, c_B)$ is the matrix $C \in M(r_A r_B, c_A c_B)$ such that

$$C = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1c_A}B \\ a_{21}B & & & \\ \vdots & & & \vdots \\ a_{r_A 1}B & & \dots & a_{r_A c_A}B \end{bmatrix}$$

For example,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$
$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} & a_{13}b_{11} & a_{13}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} & a_{13}b_{21} & a_{13}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} & a_{23}b_{11} & a_{23}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} & a_{23}b_{21} & a_{23}b_{22} \end{bmatrix}$$

The tensor sum of two square matrices $A \in M(d_A, d_A)$ and $B \in M(d_B, d_B)$ is defined as $A \oplus B = A \otimes I_{d_B} + I_{d_A} \otimes B$ where I_d denotes an identity matrix of dimension d.

For example,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & b_{12} & a_{12} & & a_{13} & \\ b_{21} & a_{11}+b_{22} & & a_{12} & & a_{13} \\ a_{21} & & a_{22}+b_{11} & b_{12} & a_{23} & \\ & a_{21} & b_{21} & a_{22}+b_{22} & & a_{23} \\ a_{31} & & a_{32} & & a_{33}+b_{11} & b_{12} \\ & a_{31} & & a_{32} & b_{21} & a_{33}+b_{22} \end{bmatrix}$$