Parallel simulation of stochastic Petri nets using recursive equations

François Baccelli, Miguel Canales

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Sophia Antipolis
B.P. 109
06561 Valbonne Cedex
France
Tél.: 93 65 77 77

Rapports de Recherche

N°1520

Programme 1
Architectures parallèles, Bases de données, Réseaux et Systèmes distribués

PARALLEL SIMULATION OF STOCHASTIC PETRI NETS USING RECURSIVE EQUATIONS

François BACCELLI
Miguel CANALES

1

Septembre 1991

Parallel Simulation of Stochastic Petri Nets using Recurrence Equations

François Baccelli (baccelli@sophia.inria.fr)
and
Miguel Canales (canales@sophia.inria.fr)

INRIA-Sophia 06565 Valbonne (France)

December 16, 1992

Abstract

A new parallel simulation method is proposed for the class of stochastic decision free Petri nets, that is amenable to an SIMD implementation. This method is based on the max, +-linear structure of recurrence equations that were established for this type of systems. Two variants are analyzed, the spatial and the temporal methods. The spatial method allows one to simulate large networks. The temporal method, which generalizes to Petri nets a method that was introduced recently for queues, is of more use for simulating systems for a long time interval. The emphasis is put on the spatial approach which is shown to provide a simple way of estimating both the cycle time and the statistics of the marking process. The theoretical parallel complexity of this algorithm is first investigated. In particular, a few examples of practical interest are provided (blocking queues in tandem and a stochastic job shop model) for which the cost of simulating $O(NT)$ events of a net of size T is in $O(N \log T)$ with this parallel simulation method, while the classical sequential discrete event simulation is in $O(NT)$ at least. These theoretical considerations are confirmed by experimental results obtained from a prototype that was implemented on the Connection Machine.

1 Introduction.

Petri nets provide a powerful modeling formalism, which allows one to describe and study various classes of systems, such as synchronous and asynchronous processes, parallel or sequential ones, and distributed or centralized ones [1, 8, 4].

Petri nets can be used both for qualitative and quantitative analysis. In the first case, one is interested in analyzing certain characteristics of the system, such as its liveness, to determine if the net (and, in consequence, the modeled system) has deadlocks or not [8]. In quantitative analysis, Petri nets are used to determine numerical measures of the system such as its throughput or cycle time (the inverse of the throughput), the statistics of the number of tokens in various places (e.g. the number of customers in a buffer of a queuing system), the statistics of response or sojourn times, the stability condition, etc.

Analytical tools have been developed for studying various qualitative properties of Petri nets, such as their liveness. But for most quantitative properties, the only analytical method that is available consists of a direct Markovian analysis [1]. This approach requires linear algebra manipulations on matrices of the size of the set of all reachable markings, which is not feasible for most systems.

Classical discrete event simulation is not always practical either because of its long execution times. Even the use of very powerful parallel computers is not sufficient to significantly accelerate classical simulations, which is not so surprising since the discrete events approach is essentially sequential [9].

An alternative to discrete events methods is the *Recurrence Equations Approach*, where equations giving epochs at which certain events occur are used to express the evolution of the net.

Event graphs are a subclass of Petri nets, where each place has no more than one input and one output arc. Cohen et al. [5] have established that recurrence equations of a deterministic event graph are linear in the semi-field $(\mathbb{R}, \oplus, \otimes)$, where $\mathbb{R}$ is the real line, $\otimes$ denotes classical addition and $\oplus$ maximization. These results were extended to the case of stochastic nets, i.e., the case where holding times (associated with places and transitions) are random variables [2, 4]. The interested reader should refer to these references for more details on this class of networks.

This paper describes the algorithms used by a software package, currently under development, that allows the user to specify a stochastic event graph by using either a graphical interface or a specification language. From this specification, a simulation program for a Single Instruction Multiple Data (SIMD) parallel machine is generated. A Connection Machine 2 (CM) is used as the architecture for running this program.

The paper is structured as follows: first, a summary of the recurrence equation approach is presented; then the algorithms used to implement the simulation of the event graph are described. Finally some comments are given on the performance that is obtained by the prototype running on the CM. These comments are based on a few examples, including a network of queues with blocking and several variations on the stochastic job shop model.

2 Petri Nets.

A Petri net is defined as a tuple $PN = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{M}_0)$, where

$$\begin{aligned}
&\mathcal{P} = \{p_1, p_2, \ldots, p_P\} \text{ is the set of places}\\
&\mathcal{T} = \{t_1, t_2, \ldots, t_T\} \text{ is the set of transitions}\\
&\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \bigcup (\mathcal{T} \times \mathcal{P}) \text{ is the set of arcs}\\
&\mathcal{M}_0 : \mathcal{P} \to \{0, 1, 2, \ldots, M\} \text{ is the initial marking.}
\end{aligned}$$

A *Timed Petri Net* is a net with firing times associated with the transitions, and/or holding times associated with places. The firing time of a transition is the time that elapses between the starting and the completion of the firing of the transition, and the holding time of a place is the time a token must spend in the place before contributing to the enabling of the downstream transitions. In this paper, the main analysis is made for the case where only the places are timed (i.e., transitions all have zero firing time). We also briefly show on section 7 how these techniques can be extended to the case where both places and transitions are timed. If this holding times are random variables, we speak of *Stochastic Nets*. The following results are summarized from [2, 4], and show the recurrence equation which describe the evolution of a stochastic event graph.

2.1 Timed Event Graphs.

From the specification of the net, matrices $A_0(n), A_1(n), \ldots, A_M(n)$ (of dimension $T \times T$) are constructed. The element ij of matrix $A_k(n)$ corresponds to the holding time of the nth token arriving to the place located between transition t_i and transition t_j and that had k tokens in the initial marking (it is assumed that there is only one such place). In the stochastic case, this holding time is a random variable. If there is no place between t_i and t_j with k tokens in the initial marking, the entry ij of the corresponding matrix is given the value ε $(= -\infty)$.

In what follows, it will be assumed that the holding times associated with each place are i.i.d., and that the holding times of the various places are mutually independent.

Let $x_i(n)$ be the epoch when the nth firing of transition t_i starts and let $x(n)$ be the vector $x(n) = (x_1(n), x_2(n), \ldots, x_T(n))$.

Consider the semi-field $(\mathbb{R}, \oplus, \otimes)$, where $\mathbb{R}$ is the real line, $\otimes$ denotes classical addition and $\oplus$ maximization. Let L be the length of the longest path without tokens in the initial marking of the net, and E be the identity matrix:

$$E = \begin{pmatrix} e & \varepsilon & \cdots & \varepsilon & \varepsilon \\ \varepsilon & e & \cdots & \varepsilon & \varepsilon \\ \vdots & & \ddots & & \\ \varepsilon & \varepsilon & \cdots & e & \varepsilon \\ \varepsilon & \varepsilon & \cdots & \varepsilon & e \end{pmatrix},$$

where e is the null element for $\otimes$ ($e = 0$).

The evolution of a FIFO net can be described through the following recurrence equation [2, 4]

$$\begin{aligned} x(n) &= x(n-1) \otimes A_1(n) \otimes A_0(n)^\star \oplus \cdots \oplus x(n-M) \otimes A_M(n) \otimes A_0(n)^\star & (1) \\ &= (x(n-1) \otimes A_1(n) \oplus \cdots \oplus x(n-M) \otimes A_M(n)) \otimes A_0(n)^\star & (2) \end{aligned}$$

where $A_0(n)^\star$ is defined by

$$A_0(n)^\star = E \oplus A_0(n) \oplus A_0(n)^2 \oplus \cdots \oplus A_0(n)^L$$

In this equation, the product of two matrices A and B is given by the relation

$$(A \otimes B)_{ij} = \bigoplus_k A_{ik} \otimes B_{kj},$$

with a similar interpretation for vector-matrix products. One can check that $(A_0(n)^m)_{ij}$ is the largest weight of a path of length m with 0 tokens between transitions i and j [2, 4].

The matrix $A_0(n)^\star$ exists (L is finite) if the net is live, i.e., if there is at least one token in each circuit in the initial marking. It is easy to see that $L < T$ for a T-transition live net and that $A_0(n)^{L'}$ is the null matrix in $(\oplus, \otimes)$ $\forall L' > L$.

Theorem 2.1 $E \oplus A_0(n) \oplus A_0(n)^2 \oplus \cdots \oplus A_0(n)^L = (E \oplus A_0(n))^L \quad \forall L \in I\!N$

Proof: By induction on L and using $A \oplus A = A \quad \forall$ matrix A
For $L = 1$

$$E \oplus A(n) = (E \oplus A(n))^1.$$

Then, suppose the claims holds for L, i.e.

$$E \oplus A(n) \oplus A(n)^2 \oplus \cdots \oplus A(n)^L = (E \oplus A(n))^L.$$

Then, it holds for $L+1$ too

$$\begin{aligned} & E \oplus A(n) \oplus A(n)^2 \oplus \cdots \oplus A(n)^L \oplus A(n)^{L+1} \\ = \; & (E \oplus A(n) \oplus A(n)^2 \oplus \cdots \oplus A(n)^L) \otimes (E \oplus A(n)) \\ = \; & (E \oplus A(n))^L \otimes (E \oplus A(n)) \\ = \; & (E \oplus A(n))^{L+1}. \end{aligned}$$

■

This theorem, allows one to compute $A_0(n)^\star$ as

$$A_0(n)^\star = (E \oplus A_0(n))^L \triangleq \widehat{A}_0(n)^L. \qquad (3)$$

Let

$$\tilde{A}(n) = \begin{pmatrix} A_1(n)A_0(n)^\star & E & \Theta & \cdots & \Theta \\ A_2(n)A_0(n)^\star & \Theta & E & \cdots & \Theta \\ \vdots & & & \ddots & \\ A_{M-1}(n)A_0(n)^\star & \Theta & \Theta & \cdots & E \\ A_M(n)A_0(n)^\star & \Theta & \Theta & \cdots & \Theta \end{pmatrix}_{MT\times MT}$$

with Θ the null matrix:

$$\Theta = \begin{pmatrix} \varepsilon & \varepsilon & \cdots & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \cdots & \varepsilon & \varepsilon \\ \vdots & & \ddots & & \\ \varepsilon & \varepsilon & \cdots & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \cdots & \varepsilon & \varepsilon \end{pmatrix}_{T\times T}$$

and let

$$\tilde{x}(n) = (x(n), x(n-1), \cdots, x(n-M+1)).$$

Equation (1) can be rewritten in the following standard form

$$\tilde{x}(n) = \tilde{x}(n-1) \otimes \tilde{A}(n). \tag{4}$$

The main interest of this standard form is that it consists of a recurrence of order 1, an essential property in some of the algorithms of interest here, and particularly for those used in the temporal approach.

Given the possibly big size of $\tilde{A}(n)$, and due to implementation considerations that will be analyzed later, it is proposed to use instead of $\tilde{A}(n)$ the matrix $\widehat{A}(n)$, defined as follows

$$\widehat{A}(n) = \begin{pmatrix} A_1(n) \\ A_2(n) \\ \vdots \\ A_M(n) \end{pmatrix}_{MT\times T} .$$

In this case, the recurrence equation (2) is re-written as

$$x(n) = \tilde{x}(n-1) \otimes \widehat{A}(n) \otimes A_0(n)^\star = \tilde{x}(n-1) \otimes \widehat{A}(n) \otimes \underbrace{\widehat{A}_0(n) \otimes \cdots \otimes \widehat{A}_0(n)}_{L} \tag{5}$$

and a shift and concatenation function f is used to get

$$\tilde{x}(n) = f(x(n), \tilde{x}(n-1)). \tag{6}$$

In the sequel, the dimension of the vector $\tilde{x}(n)$, namely MT, will be referred to as the dimension of the problem, or the dimension of the net.

Remark: If we compute $\tilde{x}(1), \ldots, \tilde{x}(N)$, we know the epochs of $O(N)$ events (here, an event is understood as a transition firing) taking place on each transition $t_1, \ldots, t_T$. In other words, the computation of these vectors is equivalent to the simulation of $O(N)$ events on each transition, that is $O(NT)$ events all over the net.

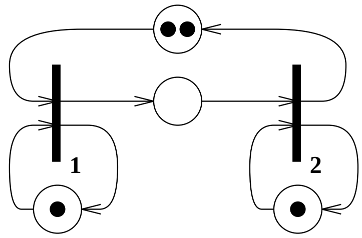

Figure 1: A simple net with stochastic timed places.

2.2 Example.

Let us consider the net depicted on Figure 1. The following equations describe its evolution:

$$
\begin{aligned}
x_1(n) &= \max(x_1(n-1) + \sigma_{11}(n); x_2(n-2) + \sigma_{21}(n)) \\
x_2(n) &= \max(x_1(n) + \sigma_{12}(n); x_2(n-1) + \sigma_{22}(n))
\end{aligned}
$$

p_{11} and p_{22} have exponential holding times, and p_{12} and p_{21} have constant ones equal to 0.

Rewritting this as in equation 5, one get:

$$A_0(n) = \begin{pmatrix} \varepsilon & 0 \\ \varepsilon & \varepsilon \end{pmatrix}$$

For this net, $L = 1$ ($t_1 \to t_2$), $M = 2$ and

$$A_1(n) = \begin{pmatrix} \sigma_{11}(n) & \varepsilon \\ \varepsilon & \sigma_{22}(n) \end{pmatrix} \qquad A_2(n) = \begin{pmatrix} \varepsilon & \varepsilon \\ 0 & \varepsilon \end{pmatrix}$$

3 Net Specification.

A user interface allows one to draw transitions and places, and arcs connecting them. This interface allows one to associate with each place the initial number of tokens, and a parameter indicating the probability law of the holding (or firing) times (nature of the law and specification of its parameters) [Figure 2].

From this specification, a parallel program is generated, which is executed on the CM in order to perform the simulation and to obtain statistical results on the system.

This program is generated with a data structure containing the specification of the graph to be simulated. This data structure corresponds to several matrices, where the entry ij provides the number of tokens between transitions t_i and t_j, or the identification of the probability law for the holding times and its parameters.

The rest of the paper focuses on the simulation algorithms to be executed on the CM.

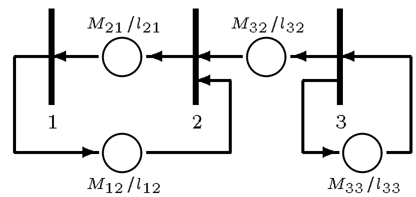

Figure 2: In this net M_{ij} represents the initial marking and l_{ij} represents the parameter indicating the probability law of holding times of place ij.

4 The Connection Machine.

The Connection Machine [7], on which the system presented in this article is being developed, is an example of SIMD architecture (*Single Instruction Multiple Data*), i.e., an architecture where all processors perform the same instruction at the same time. In general, a sequential computer (a host) is necessary to have access to this kind of machine.

A 16K one bit processors configuration is being used, where each processor has 32K bytes of memory. The processors are organized in an hypercube topology. However, a different logical topology can be specified by a program. A bidimensional grid has been chosen for the implementation of the algorithms presented here, because this topology is well adapted to work with matrices.

In the CM the Virtual Processor Ratio (VPR) indicates how many virtual processors are assigned to each real processor. In case of a VPR larger than 1, each physical processor shares its processing power among its virtual processors; this allows one to use exactly the same program for problems of different sizes, with however a cost in performance.

5 Computation of Cycle Times.

This section focuses on the calculation of the cycle time γ_i of transition t_i, which is known to be given by the limit

$$\lim_{n\to\infty} \frac{x_i(n)}{n} = \gamma_i \quad i = 1, \ldots, T.$$

The limit must be considered in the almost sure sense, and $\gamma_1, \ldots, \gamma_T$ are constants. In the case when the net is strongly connected, γ_i does not depend on i and is called the *Cycle Time* of the net [2, 4]. It is denoted γ. In what follows, we shall limit ourselves to the strongly connected case for sake of easy explanation. The general case can be treated in a very similar way.

The aim of this section is to estimate γ by computing the ratio $x_i(n)/n$ for large n. The analysis of the quality of this estimator (namely, the computation of confidence intervals) will not be addressed here, our aim being to show how the recurrence equation (2) can be used for parallelizing the simulation of the net and particularly for the estimation of γ.

In order to do the computations, two main approaches can be used: a *Spatial Approach* - where the parallelism is applied to the dimension of the problem (in this case, the size of the net) - and a *Temporal Approach* - where the parallelism is applied to the simulation interval (in this case $[0, N]$). A characterization and analysis of each approach will be given, although the emphasis will be put more on the technique that allows one to simulate large systems, namely the spatial approach.

5.1 Spatial Approach.

Roughly speaking, the spatial approach is the one to be used when the size of the network to simulate is large (say of the order of the square root of the number of processors in the CM, or larger).

The algorithm is based on the representation of equation (5) under the form

$$x(n+1) = \tilde{x}(n) \otimes \widehat{A}(n) \otimes \widehat{A}_0(n) \otimes \cdots \otimes \widehat{A}_0(n),$$

where that last product has L factors.

Each matrix and vector is mapped directly to a bidimensional grid of CM processors, in such a way that each element is assigned to a different processor. To minimize the number of involved processors, different vectors and matrices are superposed on this grid.

In order to obtain $\widehat{A}(n)$, the random matrices $A_1(n), A_2(n), \ldots, A_M(n)$ are sampled in parallel by the corresponding processors. Each processor knows the law of the matrix element assigned to it (through a set of parameters). There are no correlations among the random variables of different matrices, because the random variables $A_k(n)_{ij}$, $i, j = 1, \ldots, T; k = 1, \ldots, M$, are mutually independent.

The algorithm is the following (using a C-like notation); the complexity analysis is done assuming the number of processors of the CM is greater than or equal to MT^2.

```
for(n=1; n<N; n++)
{
   Generate Â(n)                          O(1)
   x(n) = x̃(n − 1) ⊗ Â(n) [1]             O(log MT)
   Generate Â0(n)                         O(1)
   for(i=0; i<L; i++)                     O(L log T)
      x(n) = x(n) ⊗ Â0(n)
   x̃(n) = f(x(n), x̃(n − 1))               O(log MT)
}
```

The vector-matrix multiplication is performed using the logarithmic algorithm shown in appendix A. Note that only vector-matrix multiplications are used.

[1] see Appendix A on the way to perform a matrix-vector product on the CM.

As it can be seen in the algorithm using the spatial approach, the estimate of the cycle time based on $O(NT)$ events takes $O(N(L\log T + \log MT))$. The case where $\widehat{A}_0(n)$ is deterministic is of particular interest, because $\widehat{A}_0(n)$ must be generated only once and $A_0(n)^\star$ must be calculated only once too (equation (5)). Therefore, the algorithm for this case has a cost of $O(N\log MT)$.

In the case when the quantity MT^2 is greater than the number of processors of the CM (K), the VPR is $v = \lceil MT^2/K \rceil$ (greater than 1) and the costs are $O(vN(L\log T + \log MT))$ or $O(vN\log MT)$ for $\widehat{A}_0(n)$ stochastic or deterministic, respectively.

5.2 Temporal Approach.

In the temporal approach, the associative property of matrix multiplication in $(\oplus, \otimes)$ is used, following the method proposed by Greenberg et al. [6] for queues, which is here applied to the case of event graphs, by using equation (4). The typical case where this approach is of interest is when the product MT^2 is small (compared to the number of processors in the CM) and when one wants to simulate a large number of events. To compute $\tilde{A}(0) \otimes \tilde{A}(1) \otimes \cdots \otimes \tilde{A}(N-1)$ (due to equation (4)), m matrices are associated with each of the K processors in the CM ($m = N/K$ is assumed to be an integer), their multiplication is done in each processor and finally, the intermediate results are retrieved using a tree structure.

Each processor $i = 1, \ldots, K$ calculates

$$\bigotimes_{n=m(i-1)}^{mi-1} \tilde{A}(n). \tag{7}$$

To achieve this, it generates $A_0(n)$ from a centralized description in the host, and computes $A_0(n)^\star$ using the following algorithm, which applies equation (3):

$$\begin{array}{ll} \mathtt{A_0(n)^\star = \widehat{A}_0(n)} & \\ \mathtt{for(k=1;\ k<L;k=k*2)} & \qquad O(T^3\log L) \\ \quad \mathtt{A_0(n)^\star = A_0(n)^\star \otimes A_0(n)^\star} & \end{array}$$

It is clear that, instead of $\widehat{A}_0(n)^L$, the algorithm calculates $\widehat{A}_0(n)^{p(L)} = E \oplus A_o(n) \oplus \cdots \oplus A_0(n)^L \oplus A_0(n)^{L+1} \oplus \cdots \oplus A_0(n)^{p(L)}$, where $p(L)$ is the smallest power of 2 greater or equal to L. However, the final result is not affected because $A_0^{L+1}(n), \ldots, A_0^{p(L)}(n)$ are null matrices in $(\mathbb{R}, \oplus, \otimes)$.

Afterwards, the matrices $A_1(n), \ldots, A_M(n)$ are generated, and the multiplications $A_1(n)A_0(n)^\star, \ldots, A_M(n)A_0(n)^\star$ are calculated. This way, $\overline{A}(m(i-1)+j-1)_{j=i,\ldots,m}$ is found in processor i at the jth iteration, where $\overline{A}(n)$ is defined as

$$\overline{A}(n) = \begin{pmatrix} A_1(n) \otimes A_0(n)^\star \\ \vdots \\ A_M(n) \otimes A_0(n)^\star \end{pmatrix}$$

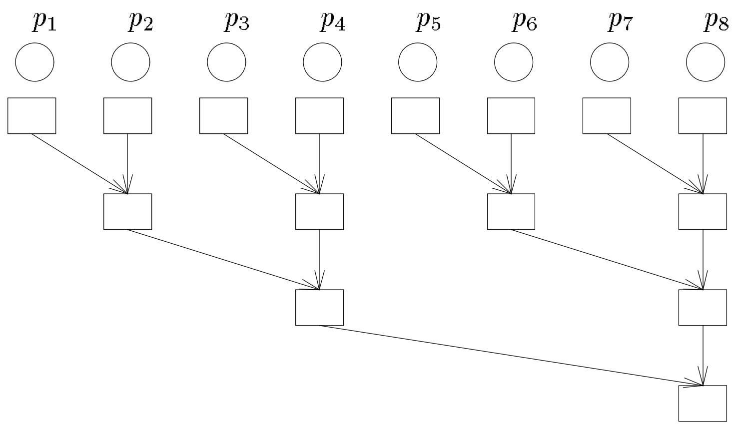

Figure 3: The partial results are retrieved using a tree topology.

$\overline{A}(n)$ is transformed in $\tilde{A}(n)$ by adding the e and ε elements, and this last matrix is multiplied by the previous result $\bigotimes_{n=m(i-1)}^{m(i-1)+j-2} \tilde{A}(n)$.

Finally, the partial results obtained by each processor applying (7), are retrieved and multiplied using a tree topology as it is shown in Figure 3.

The algorithm is the following:

```
B = E
for(i=0;i<m;i++)
{
   Generate A0(n)                          O(T^2)
   Compute A0(n)*                          O(T^3 log L)

   /* Computation of Ai(n) ⊗ A0(n)* */     O(MT^3)
   for(j=1; j ≤ M; j++)
   {
      Generate Aj(n)                       O(T^2)
      Compute Aj(n) ⊗ A0(n)*               O(T^3)
   }
   Transform Ā(n) into Ã(n)                O(1)
   B = B ⊗ Ã(n)                            O((MT)^3)
}
Transmit and multiply matrices B           O((MT)^3 log K)
Multiply by x̃(0)                           O((MT)^2)
```

Accordingly, this algorithm has a cost of $O(mT^3(\log L + M^3) + (MT)^3 \log K)$ for simulating mKT events. If $A_0(n)$ happens to be deterministic, this complexity drops down to $O(m(MT)^3 + (MT)^3 \log K)$.

Using this approach the VPR is always 1 (no virtual processors are needed), and the limitation is given by the memory required to keep $\tilde{A}(n)$ (of size $(MT)^2$) in a single processor.

6 Statistics of the Stationary Marking.

This section describes the calculation of more refined statistical measures such as the number of tokens in places, in the stationary regime of the net (which will be referred to as the *Stationary Marking*). The general assumption is that the stochastic process describing the marking couples in finite time with a stationary and ergodic process. Sufficient conditions for this to hold are given in [2].

The average sojourn time of the tokens in the place between transition t_i and t_j corresponds to the limit

$$E[W_{ij}] = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} W_{ij}(n),$$

where $W_{ij}(n) = (x_j(n) - x_i(n-m))$ if the place had m tokens in the initial marking (in this expression, we take $x_i(n) = 0\ \forall n \le 0$).

In order to avoid the calculation of the expression $x_j(n) - x_i(n-m)$, which would require keeping in memory the m previous values of $x_i(n)$, the next transformation is used:

$$\begin{aligned}
E[W_{ij}] &= \lim_{N\to\infty} \frac{1}{N} \left(\sum_{n=m+1}^{N} (x_j(n) - x_i(n-m)) + \sum_{n=1}^{N} x_i(n) - \sum_{n=1}^{N} x_i(n) \right) \\
&= \lim_{N\to\infty} \frac{1}{N} \left(\sum_{n=1}^{N} (x_j(n) - x_i(n)) + \sum_{n=N-m+1}^{N} x_i(n) \right) \\
&= \lim_{N\to\infty} \frac{1}{N} \left(\sum_{n=1}^{N} (x_j(n) - x_i(n)) \right) + m\gamma. \qquad (8)
\end{aligned}$$

The average number of tokens in the place between transitions t_i and t_j can be easily obtained applying Little's formula:

$$E[N_{ij}] = \frac{1}{\gamma} E[W_{ij}] = \frac{1}{\gamma} \lim_{N\to\infty} \frac{1}{N} \left(\sum_{n=1}^{N} (x_j(n) - x_i(n)) \right) + m. \qquad (9)$$

Remark: The reader may wonder why we assume γ to be known in (8). In fact, this expression is computed as the limit

$$\lim_{N\to\infty} \frac{1}{N} \left(\sum_{n=1}^{N} (x_j(n) - x_i(n)) + m \frac{x_j(N)}{N} \right).$$

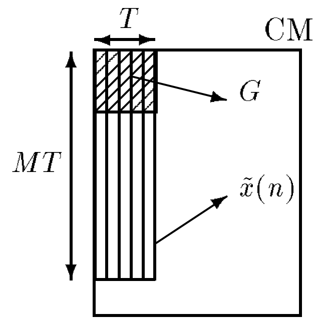

Figure 4: In the grid G each processor ij is associated with the place without tokens between transitions t_i and t_j (existing or not). G is superposed to the vector $\tilde{x}(n)$, replicated T times.

6.1 Spatial Approach.

The algorithm introduced for the computation of the cycle time (based on equations (5) and (6)) is also used to compute the statistics of the marking, the only difference being the introduction of the calculation of $x_j(n) - x_i(n)$.

A grid of processors of size $T \times T$ (called G) is in charge of the calculation. Processor ij in G know $x_j(n)$ before the transposition of vector $x(n)$, and knows $x_i(n)$ after the transposition. [1] Therefore it calculates the difference $x_j(n) - x_i(n)$ (Figure 4).

Since the algorithm used is nearly the same as the one presented in section 5.1, adding the calculation of $x_j(n) - x_i(n)$ which is done in constant time, does not change the cost, so that the estimation of the stationary marking is also in $O(N(L \log T + \log MT))$. The remarks made for the case where $\widehat{A}_0(n)$ is deterministic can also be applied.

6.2 Temporal Approach.

Using the temporal approach, as it has been shown in section 5.2, the value of $x(N)$ can be found. However the knowledge of the whole path $x(n)_{n=0,\ldots,N}$ (the evolution of the system) is required in order to perform the statistical analysis, but this path is not given by the temporal approach. Because of this, it seems more difficult to apply the temporal approach in order to estimate the statistics of the stationary marking, and this question will not be addressed here.

[1] See Appendix A

7 Timed Transitions.

In this section, we focus on the evolution of a FIFO net where timing is on both places and transitions. Let $B(n)$ be the matrix:

$$B(n) = \begin{pmatrix} \tau_1(n) & \cdots & \tau_1(n) \\ \tau_2(n) & \cdots & \tau_2(n) \\ & \vdots & \\ \tau_T(n) & \cdots & \tau_T(n) \end{pmatrix}_{T \times T}$$

where $\tau_i(n)$ is the firing time of the nth firing of transition t_i.

The matrix $B_0(n)^\star$ is defined as

$$B_0(n)^\star = E \oplus (A_0(n) + B(n)) \oplus (A_0(n) + B(n))^2 \oplus \cdots \oplus (A_0(n) + B(n))^L.$$

Therefore, the evolution of the net is described by the equation

$$x(n) = (x(n-1) \otimes (A_1(n) + B(n-1)) \oplus \cdots \oplus x(n-M) \otimes (A_M(n) + B(n-M)) \otimes B_0(n)^\star. \quad (10)$$

Let

$$\widehat{A}(n) = \begin{pmatrix} A_1(n) \\ \vdots \\ A_M(n) \end{pmatrix}_{MT \times T} \qquad \widehat{B}(n-1) = \begin{pmatrix} B(n-1) \\ \vdots \\ B(n-M) \end{pmatrix}_{MT \times T},$$

and let $\widehat{B}_0(n) = E \oplus (A_0(n) + B(n))$. Using Theorem 2.1, equation (10) can be written in the following standard form:

$$x(n) = \tilde{x}(n-1) \otimes (\widehat{A}(n) + \widehat{B}(n-1)) \otimes \underbrace{\widehat{B}_0(n) \otimes \cdots \otimes \widehat{B}_0(n)}_{L}. \quad (11)$$

$\tilde{x}(n)$ is retrieved using the shift and concatenation function:

$$\tilde{x}(n) = f(x(n), \tilde{x}(n-1)). \quad (12)$$

The algorithm for implementing equations (11) and (12) is the following (using the spatial appproach):

```
for(n=1; n<N; n++)
{
    Generate Â(n)                              O(1)
    Â(n) + B̂(n − 1)                            O(1)
    y = x̃(n − 1) ⊗ (Â(n) + B̂(n − 1))          O(log MT)
    Generate A0(n)                             O(1)
    Generate B(n)                              O(log T)
    A0(n) + B(n)                               O(1)
```

```
    for(i=0;i<L;i++)                          O(L log T)
        y = y ⊗ (A0(n) + B(n))
    Update B̂(n)                               O(log MT)
    Update x̃(n)                               O(log MT)
}
```

So the total cost of simulating $O(NT)$ events is $O(N(L\log T + \log MT))$.

8 Sparse Matrices.

In most of the systems modeled by larged marked graphs, $A_i(n), B_i(n-i)$, etc, are regular and sparse matrices. A simple but important modification has been made to the original algorithm in order to take advantage of this fact. Instead of using the complete matrices $\widehat{A}(n)$ (or $\widehat{B}(n)$) and $\widehat{A}_0(n)$ (or $\widehat{B}_0(n)$), of size MT^2 and T^2 respectively, compressed versions are used.

In these compressed matrices, we still have T columns, but in each of them, only the non ε elements are kept, together with their position in the columns of the non compressed matrices. For instance, when timing is on transitions only, the size of the compressed matrices is $Y \times T$ (instead of $MT \times T$), where Y is defined by

$$Y = \max(\max_{i=1\ldots T} \nu(\widehat{B}(0), i),\ \max_{i=1\ldots T} \nu(\widehat{B}_0(0), i),\ M). \tag{13}$$

In this formula, $\nu(A, i)$ is the number of non null elements on column i of matrix A. This results in an economy of processors, which allows one to simulate larger systems without using virtual processors (notion introduced in section 4). Section 9.4 presents an example where this technique is used.

This technique, and the corresponding expression for Y (equation (13)), extends immediately to the case when timing is on places or both places and transitions.

Remark: The dependency of M in (13) is due to the fact that, in the original method, the vector $\tilde{x}(n)$ was stored in a column of MT processors, one element by processor (section 5.1). In the compressed method, the vector is stored in M lines of processors, each line of size T.

9 Experimental Results.

For evaluating the performance of the implementation of the algorithms presented here, different models have been simulated. The parallel simulation programs were implemented using the language C/Paris, on a Connection Machine 2 with 16K one bit processors. The sequential simulations of the models were implemented using the QNAP2 package, on a Sun Sparcstation 2.

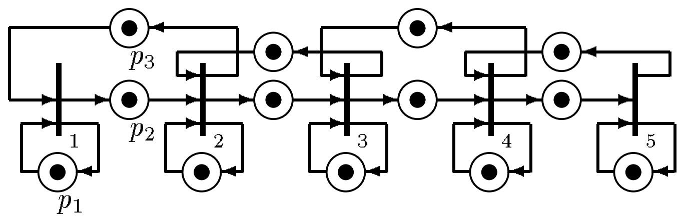

Figure 5: A blocking queues system.

In each case, the number of events simulated by the parallel and the sequential simulation is the same; because of this, the quality of the estimators is also the same (and we will hence omit the discussion on confidence intervals).

The experiments consist in comparing the execution time of both methods (the parallel and the sequential one). The reader should be warned that the obtained ratio is in no case a "speed up" in any theoretical sense. These experiments just intend to describe how our method runs on an available CM architecture compared to a commercial package running on a powerful sequential workstation.

Here follows a description of the models and an analysis of the simulation performance.

9.1 Blocking Queues in Tandem.

The first model, shown in Figure 5, is a system of finite capacity, single server, FIFO queues in tandem with blocking, where each queue can only send its customers to the next one when the number of customers in this other queue is strictly less than its capacity. The places of type p_1 represent the set-up time of the servers (a random variable, indicating the minimum time elapsed between the end of a service and the beginning of the next one). The places of type p_2 represent the transportation time between servers (a constant time $c_{i,i+1}$). Finally, the places of type p_3 are used to enforce the blocking and there is no timing here. It can be noticed that the number of tokens in circuits of type p_2-p_3 represents the maximal capacity for each server (in service and in the queue), which is two in this example. In the initial state, every server has one customer in its queue.

For this system, the matrix $A_0(n)$ is the null matrix, and $A_1(n)$ is:

$$A_1(n) = \begin{pmatrix} \sigma_{11}(n) & c_{12} & \varepsilon & \varepsilon & \varepsilon \\ 0 & \sigma_{22}(n) & c_{23} & \varepsilon & \varepsilon \\ \varepsilon & 0 & \sigma_{33}(n) & c_{34} & \varepsilon \\ \varepsilon & \varepsilon & 0 & \sigma_{44}(n) & c_{45} \\ \varepsilon & \varepsilon & \varepsilon & 0 & \sigma_{55}(n) \end{pmatrix}$$

where $\sigma_{ij}(n)$ denotes the timing of the nth token in the place between transitions t_i and t_j. In the simulated model $\{\sigma_{ii}(n)\}_n$ is a sequence of i.i.d. exponentially distributed random variables.

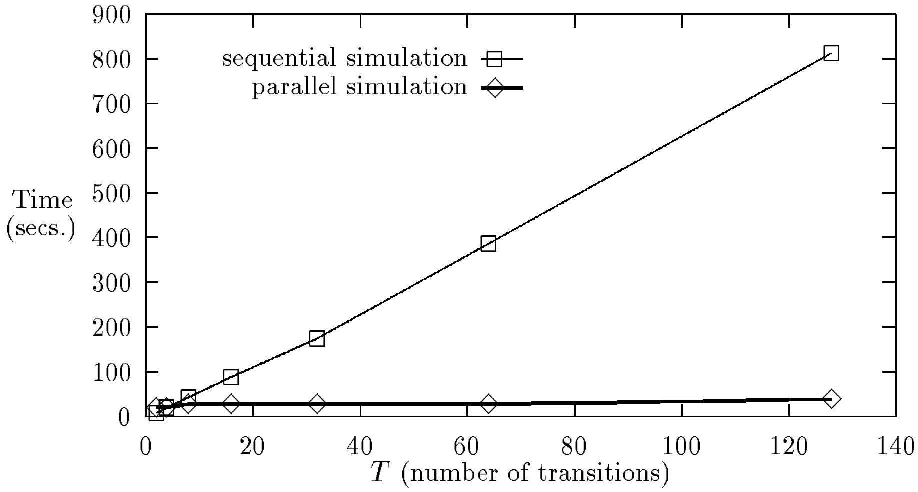

Figure 6: Simulation of the Blocking Queues System.

In this model, there are no places without tokens in the initial marking. Therefore, there is no need to calculate $A_0(n)^\star$. The recurrence equation is of the form $x(n) = x(n-1) \otimes A_1(n)$ ($M = 1$), which takes $O(N \log T)$ for computing $O(NT)$ events (provided there are enough processors).

This system can be expanded horizontally, to represent bigger systems. For the simulation program, the number of transitions in the system (the size of the problem) is given by the number of servers. Figure 6 presents a comparison of the performances obtained with the parallel simulation proposed here and the QNAP 2 discrete event sequential simulation of the model.

The sequential discrete event simulation of $O(NT)$ events takes at least $O(NT)$ (and possibly up to $O(NT \log T)$ due to the variations in the length of the event list).

Remark: In these experiments, one can observe that the parallel simulation time seems to be constant in T, instead of logarithmic. This can be explained by the following fact that is specific to the CM: all processor topologies declared in a CM program must be such that the total number of nodes in this topology is equal to P, the total number of processors of the CM ($16K$ here). Thus in order to perform, for example, the product of a vector by a square matrix, all of size T, where $T < \sqrt{P}$ (which can in principle be done in $O(\log(T))$ using the algorithm of Appendix A), the algorithm must actually declares a larger matrix and a larger vector obtained from the initial ones by adding empty lines and columns in such a way that the total number of entries in this extended matrix is equal to P. Accordingly, the CM always performs vector-matrix product via these extended objects, in a time that depends on P and not on T, provided $T^2 \leq P$.

Therefore, in our experiments, only the rightmost point of the parallel simulation curve is

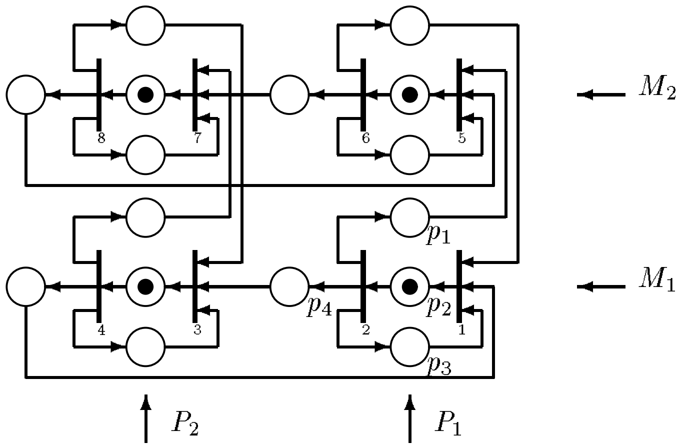

Figure 7: A job shop model with two differents machine types, and two differents product types.

actually obtained without such an extension of the matrices, namely via the algorithm described in the previous sections. In other words, the other points of the parallel simulation curve are obtained via an algorithm that is worse than the one described in the previous sections (since an extension is needed). If no such constraint was imposed by the CM, the simulation times of these other points could in principle be further reduced.

9.2 Job Shop Model.

A job shop model has different types of products, manufactured by different types of machines (Figure 7). Each machine manufactures sequentially each type of product (product 1, product 2, etc.) and each product is processed sequentially by each machine before being finished (machine 1, machine 2, etc.). The places of type p_2 are associated with the manufacturing times of the machines, and in the model these times are random variables. The places of type p_3 are used to enforce the FIFO discipline of the service, and the timing here is null. The places of type p_4 are associated with the set-up time of the machine, necessary to begin the production of a new kind of product (a constant time equal to 2 in the model), and the places of type p_1 are associated with the transportation times between machines (also constant and equal to 4).

For this system, the matrices $A_0(n)$ and $A_1(n)$ are respectively:

$$\begin{pmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 0 & \varepsilon & 2 & \varepsilon & 4 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 2 & \varepsilon & 0 & \varepsilon & \varepsilon & \varepsilon & 4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 4 & \varepsilon & \varepsilon & \varepsilon & 0 & \varepsilon & 2 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 4 & \varepsilon & 2 & \varepsilon & 0 & \varepsilon \end{pmatrix} \qquad \begin{pmatrix} \varepsilon & \sigma_{12}(n) & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \sigma_{34}(n) & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \sigma_{56}(n) & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \sigma_{78}(n) \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix}.$$

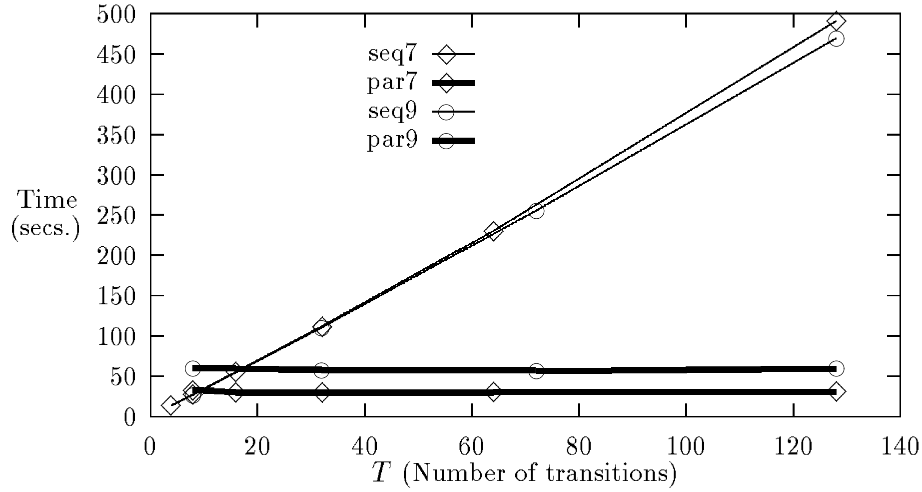

Figure 8: seq7 and par7 correspond to the sequential and parallel simulations of the job shop model depicted in Figure 7; seq9 and par9 are analogous, but for the model in Figure 9.

This model can be expanded in the two directions: machines in the vertical direction, and products in the horizontal direction. It can be seen that, with the given initial marking, the number of machines of the same type is equal to the number of different products (these numbers are represented by the quantity of tokens in the machine circuits and in the product circuits, respectively).

The number of transitions used in this model (8 in the Figure) is twice the number of machine types times the number of product types. Note that $A_0(n)$ is deterministic, therefore $A_0(n)^\star$ can be calculated only once for the whole simulation.

The recurrence equation is of the form $x(n) = x(n-1) \otimes A_1(n) \otimes A_0^\star$. The complexity of simulating $O(NT)$ events in this system is also $O(N \log T)$. Figure 8 shows the performance of the parallel simulation of the job shop model described here (par7), compared to the performance of a discrete event sequential simulation (seq7).

9.3 Another Job Shop Model.

Figure 9 shows the same job shop model, but with a different initial marking. In this case $A_0(n)$ is stochastic and $A_0(n)^\star$ must be recalculated in each iteration of the simulation algorithm.

The recurrence equation is written in this case as

$$x(n) = x(n-1) \otimes A_1(n) \otimes \underbrace{\widehat{A}_0(n) \otimes \cdots \otimes \widehat{A}_0(n)}_{L}.$$

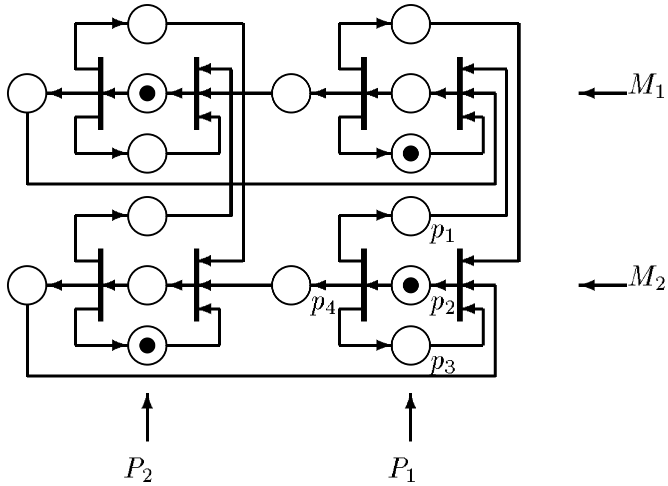

Figure 9: A job shop model similar to the one shown before, but with a different initial marking.

Therefore, when applying the spatial method, a complexity $O(NL\log T)$ is obtained. Clearly, in the worst case, L is nearly equal to T, but in many practical situations L is sublinear in T. This is the case of the model presented in Figure 9, where L is 3, independently of the size of the net.

Figure 10 shows how the same model, expanded to the size of four machines times four products, still has L equal to 3. In fact, in the worst case of the square job shop model, L is $O(\sqrt{T})$. This is achieved when there is only one machine and only one product of each type. This shows that the parallel simulation of a square job shop model is $O(N\sqrt{T}\log MT)$ or better. The simulation performance of this model is shown in Figure 8 (par9 and seq9).

9.4 The Job Shop Model with Timed Transitions.

A job shop model similar to the one depicted in Figure 10 is shown in Figure 11. In the last model, timing is on transitions. Given the marking, there are p machines types and p products types. For each type of machine, there are p machines able to work in parallel. For each type of product, there are p instances of the product that can be manufactured in parallel. The initial marking is the one indicated in Figure 11, where each place has one token; therefore, in this model, $L = 0$.

The technique described in section 8 for manipulating sparse matrices has been implemented, in order to allow the simulation of graphs larger than the ones simulated using the original algorithm.

Given the topology of the graph, the original matrix $\widehat{B}(n)$ of size $MT \times T$, is reduced to a compressed matrix of size $3 \times T$ (where $T = p^2$). Given that the CM must configure the processors in a grid where the sizes are powers of two, the biggest system that can be simulated with 16Kprocesors (without using virtual processors) has 4,096 transitions ($4 \times 4096 = 16,384$).

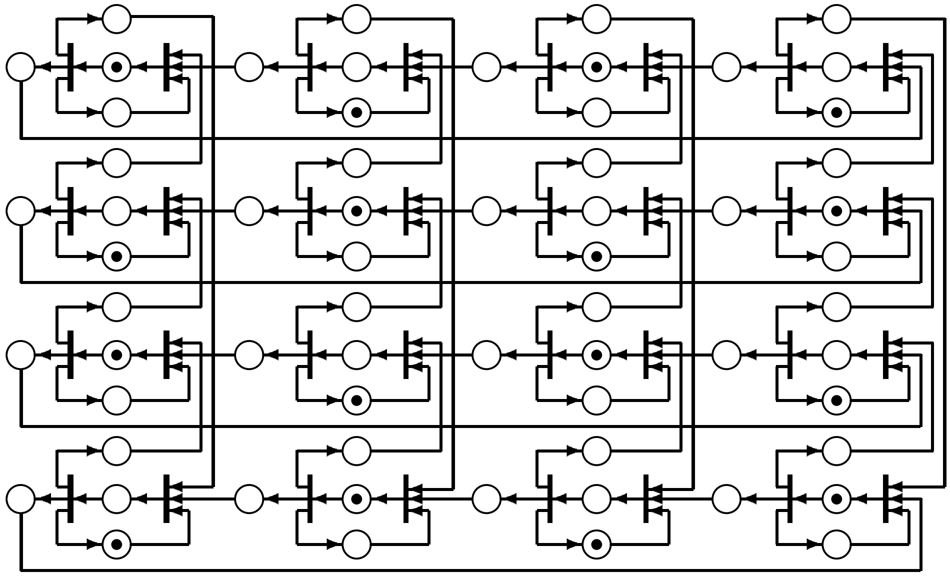

Figure 10: A job shop model as shown in Figure 9, but of size four different types of machines times four different types of products.

The performance of the parallel simulation of this model is presented in Figure 12, and compared to the sequential one. Note that for 4,096 transitions, the parallel simulation is 1760 faster than the sequential one running on a SPARCstation 2.

Remark: Using the normal algorithm (without compressed matrices), the biggest system simulated, without using virtual processors, has 128 transitions ($128^2 = 16,384; M = 1$). In this case, the parallel simulation only runs 40 times faster than the sequential one.

10 Conclusions.

The Spatial and the Temporal approaches have been applied to the parallel simulation of marked graphs. The spatial approach uses the whole CM to store the description of the model, while the temporal approach needs to store it on each processor. Because of this, the spatial approach can simulate larger systems than the temporal one.

Furthermore, the spatial approach allows the computation of both the cycle time, and the average sojourn time or average number of tokens in places, during the run of the simulation, which seems more difficult to do with the temporal approach.

Concerning the performance shown by the different simulation programs, examples have been shown where the parallel program shows a logarithmic behavior in the size of the problem, while the sequential one is always at least linear in this variable.

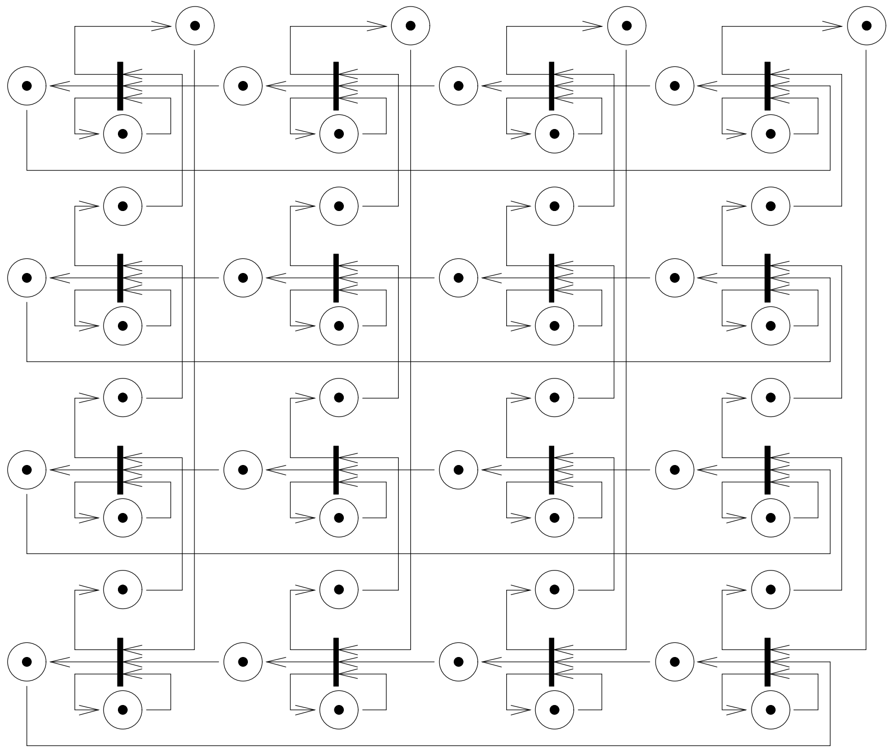

Figure 11: Job Shop Model with timed transitions representing the service time. The transitions has been recycled in order to assure the FIFO discipline of the service.

Future work will be oriented to accelerate the logarithmic behavior presented by the algorithm when $\widehat{A}_0(n)$ is stochastic. To achieve this, it is proposed to redistribute the tokens (in the initial marking) all over the net, so as to obtain another reachable marking which leads to the same stationary regime as the initial one. The aim of this redistribution is to reduce L, the longest path without tokens, to a new value $L^\star \leq L$. This means that the cost $O(NL \log MT)$ is reduced to $O(NL^\star \log MT)$, and in particular cases, when $\widehat{A}_0(n)$ becomes deterministic, the recalculation of $A_0(n)^\star$ is avoided and the cost is $O(N \log MT)$. An example of redistribution is given in the Figure 13.

Another potential application of the redistribution of tokens is the (possible) decrease of M, the maximum number of tokens in a place in the initial marking. It will imply a reduction in the component $\log MT$ of the cost, and the number MT^2 of processors required to perform the simulation (which could imply a decrease of the VPR).

Future work could also be oriented to the simulation of state dependent systems, for which the dynamic is a function of the present or past state. For such a system, the evolution equation is rewritten in the form $\tilde{x}(n) = \tilde{x}(n-1) \otimes \tilde{A}(n, \tilde{x}(n-1))$. The complexity of simulating this system could not be affected depending on the nature of the dependence. For instance, consider a graph where the place between transition t_i and t_j has an exponential holding time of constant

$$\mathbf{R} = \frac{\text{SPARC 2 Sequential Simulation Time}}{\text{CM 2 Parallel Simulation Time}}$$

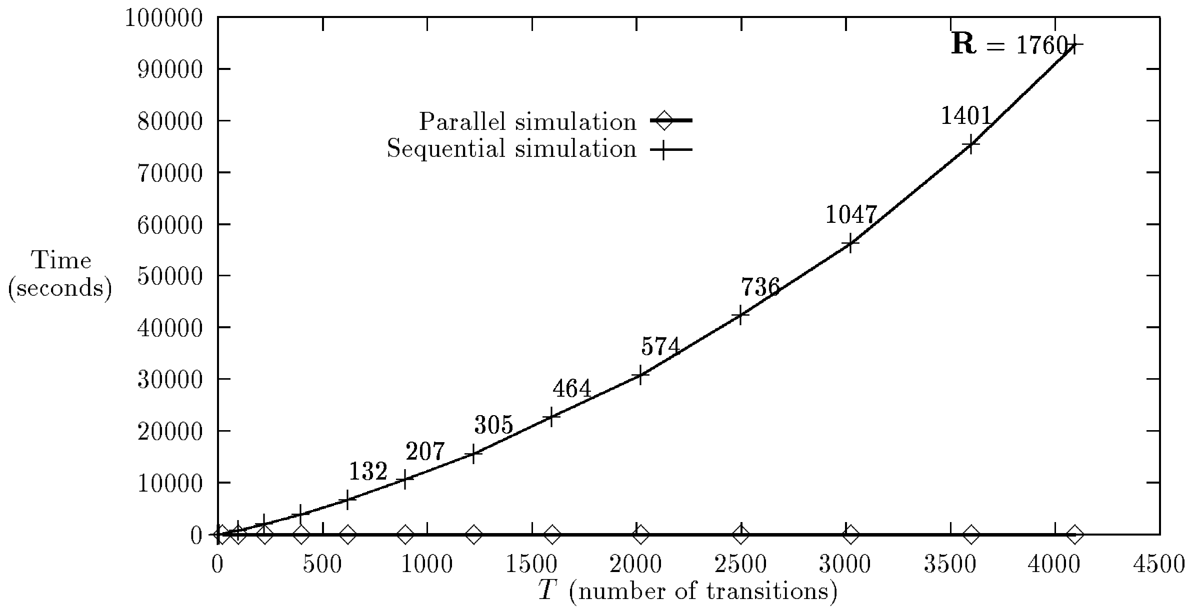

Figure 12: Performance comparison of sequential and parallel simulations of the Job Shop Model depicted in Figure 11 for different sizes of T. The best results are obtained for the parallel simulation, which introduces timing in the transitions and manipulates sparse matrices. The ratio parallel v/s sequential simulation time, is shown for each point.

---

parameter λ_{ij}. It could be changed to

$$\lambda_{ij}(n) = \lambda_{ij}(n-1)\frac{x_j(n-1) - x_i(n-1)}{x_j(n-2) - x_i(n-2)}$$

to reflect more precisely the behaviour of the real system, without affecting the complexity of the parallel algorithm.

A Vector-Matrix Multiplication in $(\mathbb{R}, \oplus, \otimes)$.

Let A be a $N \times M$ matrix and X a vector of size N. To compute $C = X \otimes A$ using a spatial approach, A is mapped directly on the CM processors, one matrix element on each processor. Vector X is replicated over all the columns of A. If c, x and a are the variables for the elements of C, X and A, each processor computes $b = x \otimes a$. Then, c is computed as the maximum b value over each column; this maximum is left in each element of the corresponding column. It is clear that the result is obtained as N row vectors of length M (Figure 14); in order to collect the result as column vectors, they must be transposed.

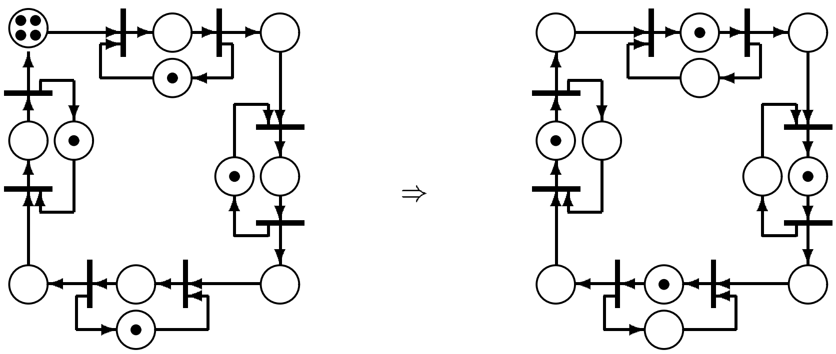

Figure 13: This example shows how the redistribution of tokens can decrease the length of the longest path without tokens (L).

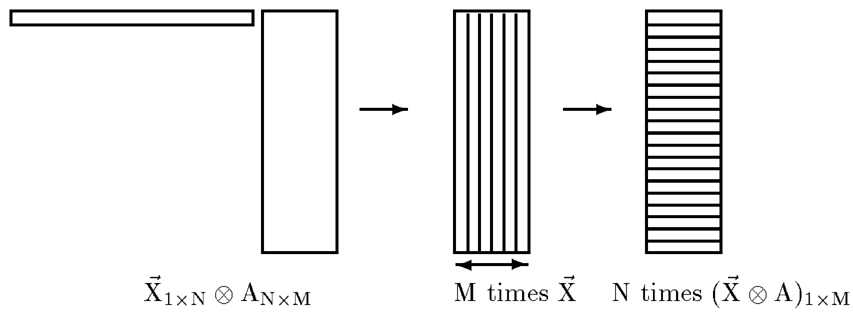

Figure 14: The vector is replicated M times over the columns of the matrix, the calculation $\otimes$ is done over each processor, the calculation $\oplus$ is done over the column, and N replicas of the result are retrived, as line vectors.

The algorithm is

```
b = x ⊗ a
c = Max_over_column(b)
send(c_ij → c_ji)
```

The computation of the maximum takes $O(\log N)$. It is supposed the `send` instruction, used to transform the row vectors in column ones, takes $O(\log M)$. Then, the algorithm takes $O(\log N + \log M)$ to do the vector-matrix multiplication.

B Acknowledgements.

We wish to thank Bruno Gaujal, Eitan Altman and particularly Alain Jean-Marie for their helpful comments during the development of the algorithms and the software package described here. We also thank Luz Echeverría for her help in the preparation of this document.

References

[1] M. Ajmone, G. Balbo, and G. Conte. *Performance Models of Multiprocessors Systems.* MIT press, 1986.

[2] F. Baccelli. Ergodic theory of stochastic petri networks. *Annals of Proba*, 20(1):375–396, 1992.

[3] F. Baccelli and M. Canales. Parallel simulation of stochastic petri nets using recursive equations. Technical Report 1520, INRIA, September 1991. Submitted to ACM TOMACS.

[4] F. Baccelli, G. Cohen, G.J. Oslder, and J.P. Quadrat. *Synchronization and Linearity.* Wiley, 1992.

[5] G. Cohen, P. Moller, J.P. Quadrat, and M. Viot. Une théorie linéaire des systèmes à événements discrets. Technical Report 362, INRIA, January 1985.

[6] A. Greenberg, I. Mitrani, and B. Lubachevsky. Algorithms for unboundedly parallel simulations. *ACM Transactions on Computer Systems*, 9(3), August 1991.

[7] W. Hillis. *The Connection Machine.* MIT, Cambridge, Mass., 1985.

[8] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), April 1989.

[9] R. Rubinstein. *Simulation and The Monte Carlo Method.* Wiley, 1981. Series in Probability and Mathematical Statistics.

[10] W. Tichy. Parallel matrix multiplication on the connection machine. In *Proceedings of the Conference on Scientific Applications of the CM*, 1988.