Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

CONCURRENT EXECUTION OF TIMED PETRI NETS *

Alois Ferscha

Institut für Angewandte Informatik und Informationssysteme
Universität Wien
Lenaugasse 2/8, A-1080 Vienna, AUSTRIA
Email: ferscha@ani.univie.ac.at

ABSTRACT

The dynamics of discrete event systems can be effectively described and analyized using the timed Petri net formalism. The aim of this paper is to comprehensively present the achievements attained in accelerating Petri net executions by using parallel or distributed multiprocessing environments. The basic problem is to generate concurrent Petri net executions insuring correctness in the sense that the partial ordering of transition firings produced is consistent with the total event ordering that would be produced by a (hypothetical) sequential execution. Two lines of thought have been followed: in *parallel* simulations transition firings evolve as governed by a SIMD iteration mechanism. *Distributed* simulations aim at a proper synchronization of firings in spatially different net parts to avoid timing inconsistencies and alterations of the execution behavior. In both cases, structural properties of the underlying Petri net can be efficiently used to simplify and/or accelerate concurrent execution implementations.

1 INTRODUCTION

General PNs can be classified with respect to the complexity of structure they employ. With decreasing restrictiveness in allowing structural constructs, the modeling power of the resulting PN class increases, and so does its analysis complexity. While analytical quantitative evaluation methods are feasible and tractable for less expressive PN classes and mainly exponential timing, simulation tends to be the only applicable method for analyzing general PN classes and versatile timing. Since simulation can become quite aggressive in the use of computational resources, parallel and distributed simulation techniques used with multiprocessors or distributed computing environments can still be a promising approach for the execution of complex simulation models. In this work we shall give a thorough background on PNs and how they describe time dynamic discrete event systems. We show how qualitative properties of PNs are obtained, and how they can be used to realize their parallel or distributed execution. Since realizations of these concurrent executions are always implementation (software) and technology (hardware) dependent, we shall avoid presenting absolute execution performance here, and will study just the sources of parallelism which make speedup possible.

2 BACKGROUND

Petri Nets A Petri net (PN) (as reviewed by Murata (1989)) is usually denoted by a tuple $(P, T, F, W, \mu^{(0)})$ where P is the set $(p_1, p_2, \ldots p_\mathcal{P})$ of *places*, T is $(t_1, t_2, \ldots t_\mathcal{T})$, the set of *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ defines an input-output relation to and from transitions, represented by a set of directed *arcs*. (The introduction of *inhibitory arcs* or transition priorities raising the expressive power of PNs to that of Turing machines is straightforward and will not be presented here.) The set of input places to $t \in T$ is denoted by $I(t)$, analogously $O(t)$ is its set of output places. I, O are used in a similar way for places. $W : F \mapsto \mathbb{N}^+$ assigns weights $w((p,t))$ to arcs $(p,t) \in F$ to denote their multiplicity. $\mu^{(0)}$ is a *marking* (vector) generated by the marking function $\mathcal{M} : P \mapsto \mathbb{N}^0$, expressing for every p the number of *tokens* $\mu^{(0)}(p)$ initially assigned to it. The dynamic behavior of a PN is described in terms of two rules:

(*i*) (*enabling rule*) A transition $t \in T$ is *enabled* in some marking μ iff each of its input places holds a "sufficient" amount of tokens, i.e. iff $\forall p \in I(t)$, $\mu(p) \geq w((p,t))$ in μ. $E(\mu)$ is the set of all transitions enabled in μ. Every $t \in E(\mu)$ may or may not fire.

(*ii*) (*firing rule*) When a transition $t \in T$ fires in μ it creates μ' by removing a certain amount of tokens from its input places and depositing a certain amount of tokens in its output places: $\forall p \in I(t) \cup O(t)$ $\mu'(p) = \mu(p) - w((p,t)) + w((t,p))$. The firing of t in the marking $\mu^{(i)}$ (reached after i other firings) is denoted by $\mu^{(i)} \xrightarrow{t} \mu^{(i+1)}$.

Let $w^+_{t,p} = w((t,p))$ be a shorthand for the multiplicity of the arc pointing from t to p, and $w^-_{t,p} =$

*This work was supported by a grant from the Academic Senate of the University of Vienna, Austria.

$w((p,t))$ respectively. Clearly, $a_{t,p} = w^+_{t,p} - w^-_{t,p}$ is the "amount of change" in the number of tokens in p caused by the firing of t. The set of all $a_{t,p}$ ($t \in T$, $p \in P$) defines the "amount of change" for PN in the *incidence matrix* $A = [a_{t,p}]$. Now the enabling of $t \in T$ can be verified by the condition $w^-_{t,p} \leq \mu(p)\ \forall p \in I(t)$, and consecutive firings can be described as

$$\mu^{(i)} = \mu^{(0)} + A^T \sum_{k=1}^{i} u_k \tag{1}$$

We read $\mu^{(i)}$ here as a $\mathcal{P} \times 1$ column vector showing the current number of tokens in p_j in its j-th component ($\mu^{(i)}_j = \mu^{(i)}(p_j)$). A^T is the transposed of the $\mathcal{T} \times \mathcal{P}$ matrix A, and u_k is $\mathcal{T} \times 1$ column vector having 1 in a single component and 0 in all others. Assume the *firing vector* u_k to have the 1 in row l. Then the operation $A^T u_k$ "selects" the l-th column of A^T (row of A) containing the "amount of change" caused by firing t_l, and the change in the marking can be simply determined by adding the vector $a_{l\bullet}$ to μ. Repeated transition firings can be represented by cumulating firing vectors u_k to a *firing count vector* $x = \sum_{k=1}^{i} u_k$, i.e. x counting in its l-th row the number of firings of $t_l \in T$ that must occur to generate $\mu^{(i)}$. Consequently, $\mu^{(j)} - \mu^{(i)} = \Delta\mu = A^T x$ is the marking difference imposed by firing transitions with respect to x starting from $\mu^{(i)}$ and ending up in $\mu^{(j)}$. A vector x with strictly positive integer components that solves $0 = A^T x$ does not impose *any* difference on *any* μ when executing the corresponding firings (since $\Delta\mu = 0$). x is called a *T-invariant* of PN, expressing that by firing $t_l \in T$ for x_l times starting in μ, the PN ends up in μ again, given that all the firings encoded in x are actually realizable.

A dual interpretation exists for a $\mathcal{P} \times 1$ vector y, which by its components y_l "weighs" the changes caused by firing $t_l \in T$ (encoded in row l of A) in such a way, that the overall change zeroes out ($0 = Ay$). An *integer* solution y to $Ay = 0$ is called a *P-invariant*, expressing that the *weighted* number of tokens $\overline{\mu} = \sum_{i=1}^{\mathcal{P}} \mu^{(j)}_i y_i$ in the respective places p_i is invariant over any reachable marking $\mu^{(j)}$. In other words, the firing of transitions does not change $\overline{\mu}$. The set of places p_i for which $\mu^{(j)}_i > 0$ is called the *support* of the invariant, and is said to be *minimal* if no subset of it is a support. A *minimal support P-invariant* can always be found if a P-invariant exists, and any linear combination of a P-invariant is still a P-invariant. Many properties of PNs (useful in the context of parallel and distributed simulation) can be verified using invariants, they may however not exist for the particular PN under study. Practically, vectors y can be found by rewriting $Ay = 0$ as

$$\begin{bmatrix} A_{11}^{|r \times \mathcal{P}-r|} & \overline{A}^{|r \times r|} \\ A_{21}^{|\mathcal{T}-r \times \mathcal{P}-r|} & A_{22}^{|\mathcal{T}-r \times r|} \end{bmatrix} \begin{bmatrix} Y_1^{|\mathcal{P}-r \times 1|} \\ Y_2^{|r \times 1|} \end{bmatrix} = 0 \tag{2}$$

where $\overline{A}$ is a submatrix of A having full rank r, and $[Y_1 Y_2]$ is the respective decomposition of Y. Then from $A_{11}Y_1 + A_{12}Y_2 = 0$ (the linearly independent part of (2)) we get $Y_2 = -\overline{A}^{-1} A_{11} Y_1$, which, by combining Y_1 to the left-hand and right-hand side, gives

$$\begin{bmatrix} Y_1^{|\mathcal{P}-r \times 1|} \\ Y_2^{|r \times 1|} \end{bmatrix} = \begin{bmatrix} I^{|\mathcal{P}-r \times \mathcal{P}-r|} \\ -\dfrac{A_{11}^{|r \times \mathcal{P}-r|}}{\overline{A}^{|r \times r|}} \end{bmatrix} Y_1^{|\mathcal{P}-r \times 1|}. \tag{3}$$

Here I is a $\mathcal{P} - r \times \mathcal{P} - r$ identity matrix ($Y_1 = IY_1$). Thus we get all possible P-invariants as vectors Y with integer coefficients. A marking $\mu^{(j)}$ is said to be *reachable* from marking $\mu^{(i)}$ if there exists a sequence of transitions $\sigma = (t_k, t_l, \ldots)$ such $\mu^{(i)} \xrightarrow{t_k} \mu^{(i+1)} \xrightarrow{t_l} \mu^{(i+2)} \ldots \xrightarrow{t_m} \mu^{(j)}$, or $\mu^{(i)} \xrightarrow{\sigma} \mu^{(j)}$ for short. The set of all markings reachable from $\mu^{(i)}$ is conveniently denoted by $RS(\mu^{(i)})$, and we must recognize that $|RS(\mu^{(i)})|$ can be exponential as a function of $\mathcal{P}$ and the number of tokens in $\mu^{(0)}$. A PN is *conservative* over a set of places $P' \subseteq P$ if the number of tokens is "conserved" over P' despite transition firings. As we can conclude from (2), a necessary and sufficient condition for conservativeness is the existence of a P-invariant y, $Ay = 0$, with positive components refering to places $p \in P'$. A PN is *persistent* over a subset of transitions $T' \subseteq T$ if any $t \in T'$ can loose enabling only by its own firing, but not by the firing of some other transition $t' \in T$.

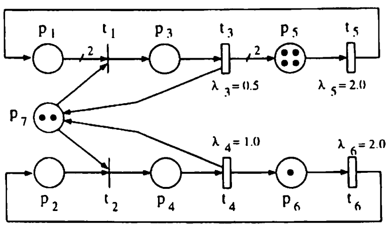

Figure 1: PN1: Comm. Processes with Contention

Example The net PN1 in Figure 1 models the behavior of a system of four computational (c-)processes (p_5), operating in parallel to each other and to one I/O process (p_6). Two c-processes after having done local computations (t_5) compete with another c-process pair and the I/O process for two shared communication devices (e.g. channels, p_7) to be able to communicate (t_3). We obtain the following properties: PN1 is not conservative over P, but over (p_2, p_4, p_6) and (p_3, p_4, p_7). It is persistent over (t_3, t_4, t_5, t_6), while t_1 and t_2 are not persistent: firing t_2 in $\mu = (2,1,1,0,0,0,1)^T$ will disable t_1 and vice versa. We find the minimum support P-invariants $y_1 = [1,0,2,0,1,0,0]^T$, $y_2 = [0,0,1,1,0,0,1]^T$ and $y_3 = [0,1,0,1,0,1,0]^T$, saying that irrespective of any transition firing $\mu^{(i)}_1 + 2\mu^{(i)}_3 + \mu^{(i)}_5 = 4$, $\mu^{(i)}_3 + \mu^{(i)}_4 +$

$\mu_7^{(i)} = 2$, and $\mu_2^{(i)} + \mu_4^{(i)} + \mu_6^{(i)} = 1$ in $\mu^{(i)} \in RS(\mu^{(0)})$. Two T-invariants exist: $x_1 = [1,0,1,0,2,0]$ and $x_2 = [0,1,0,1,0,1]$, expressing that if in $\mu^{(i)}$ there exists a sequence to (exclusively) fire t_1, t_3 and two times t_5 yielding $\mu^{(i+4)}$, or to fire t_2, t_4 and t_6 yielding $\mu^{(i+3)}$, then $\mu^{(i)} = \mu^{(i+4)}$ or $\mu^{(i)} = \mu^{(i+3)}$.

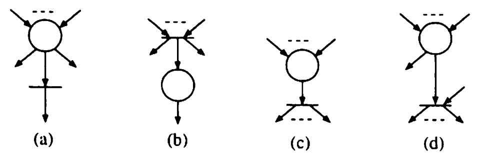

Figure 2: Structural Restricitions that classify PNs

PN classes Restricting the input-output relation F in a general PN to allow for transitions constructs only like those in Figure 2 (*a*) ($|I(t)| = |O(t)| = 1\ \forall t \in T$) yields a class of nets known as *state machines* (SMs). Since transitions can neither create nor destroy tokens, transtitions can be removed from a SM yielding the classical representation of a finite state automaton, which it is. SMs can model *concurrency* and *conflict* (*choice*), but not *synchronization*. If F is restricted for places to structures like in Figure 2 (*b*) ($|I(p)| = |O(p)| = 1\ \forall p \in P$), then PN is called a *marked graph* (MG). Places are redundant in MGs, and their removal yields what is well known as series parallel (task-)graphs. MGs can model *concurrency* and *synchronization*, but no *conflict*. The extension to MGs that allows multiple input to places but only single output from places as in Figure 2 (*c*) ($|O(p)| = 1\ \forall p \in P$) is commonly called *conflict free* nets (CFNs). A PN that allows both constructs in Figure 2 (*a*) and (*b*), but not the construct in (*d*) ($O(p_i) \cap O(p_j) \neq \emptyset \Rightarrow |O(p_i)| = |O(p_j)| = 1\ \forall p_i, p_j \in P, i \neq j$) is called a *free choice* net (FCN), which unifies (and extends) the power of SMs and MGs, but does not allow situations called *confusion*. In a FCN if two transitions have an input place in common, there is always a "free choice" of which one to fire, i.e. no marking can exist in which one is enabled and the other is not. (PN1 exhibits confusion around p_7). Finally we refer to a PN that does not impose any of the restrictions in Figure 2 as a *general net* GN.

Timing A PN executes transition firings instantaneously, i.e. no time is consumed, which is certainly sufficient for reasoning about the *quality* of system behavior (causalities, synchronization, etc.). To make the PN formalism adequate also for *quantitative* (i.e. performance) analysis, finite timing of activities can be expressed by associating a time concept to places, transitions, arcs, tokens, or any combination of them. Ramchandani (1974) assigned *firing times* to transitions representing the time interval during which a corresponding activity is going on, whereas Sifakis (1977) associated *holding times* with places modeling the time expired between the occurrence of events (an arriving token has to reside in the place for the holding time before it can enable a transition). Several equivalences among timing notions have been proven, see e.g. David and Alla (1992). Probably the most popular time extension to PNs are *Stochastic Petri Nets* (SPNs), assigning exponentially distributed firing rates to transitions for *continuous time* (CT) systems, or geometrically distributed firing rates in case of *discrete time* (DT) systems. In both cases an isomorphism among the markings $RS(\mu^{(0)})$ and the states in a Markov chain (MC) can be proven, which allows the evaluation of models in terms of CTMC or DTMC *steady state* or *transient* analysis. PN1 is a *Generalized SPN* (GSPN) (see Ajmone Marsan et.al. (1987)), allowing a combination of nontimed (bars) and stochastically timed transitions (empty boxes), while still supporting embedded MC analysis. We shall consider mostly transition timing when talking about *timed PNs* (TPNs).

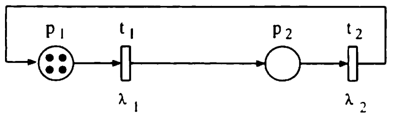

Figure 3: PN2: A TPN with Model Parallelism

Timed Enabling and Firing Semantics Important for modeling, analysis and simulation is how the timing concept attached to PNs modifies the enabling and firing rule of PNs: If $\forall p \in I(t)$, $\mu(p) \geq c\ w((p,t))$ $c > 1$ in μ then t is said to be *multiply enabled* at degree c. If t can only "fire" (serve) one enabling at a time, we talk about *single server* (SS) (enabling) semantics, and *infinite server* (IS) semantics if *any* amount of enablings can be served at a time. If t_1 in the SPN in Figure 3 followed SS, then the *occurrence time* $ot(t_1(\bullet_i))$ of t_1 with the i-th token out of the four in p_1 would be $ot(t_1(\bullet_i)) = ot(t_1(\bullet_{i-1})) + X_i$, where $X_i \sim exp(\lambda_1)$ is an exponential variate. If t_1 applied IS, then $ot(t_1(\bullet_i)) = X_i\ \ \forall \bullet_i$, expressing a notion of parallelism among the tokens. The latter is useful in our interpretation of PN2 where tokens model machines subject to failure (t_1) and repair (t_2). Note that for memoryless distributions, a t with IS can be modeled as an appropriately "faster" SS transition by choosing a marking dependent rate, e.g. $\mu_1\lambda_1$ for t_1. In a PN with IS, SS can always be modeled by assigning to transitions a loop-back place marked with a single token.

Both the enabling and the firing can be timed in a TPN. Timing the firing is straightforward. If the enabling is timed, then for SMs, FCNs and GNs (but not for MGs and CFNs) two questions arise related to timing: (*i*) should it be allowed to preempt the enabling, and if so, (*ii*) how should intermediate "work" be memorized. In a policy with non-preemptive enabling semantics, once t becomes enabled it is decided whether it should fire. If t fires, it removes tokens from $I(t)$ (*start-firing event*), "hides" tokens during the firing period, and after that releases tokens to $O(t)$ (*end-firing event*). This policy is know as *pres-*

election (PS), since in cases of conflict among t and t' one of them is preselected for firing. *Atomic firing* (AF) refers to a policy where t, once it becomes enabled, its firing is delayed to the instant when the enabling time $\tau(t)$ has expired. If t is still enabled at that time, it fires exactly like in a PN. For the case where t looses enabling during that period, either a new, *resampled* $\tau(t)$, or the non expired part of $\tau(t)$ (*age memory*) is used for the eventual new enabling of t. While with PS conflicts (among timed transitions) are resolved by predefined mechanisms like random switches or priorities (from "*outside*"), AF resolves (timed) conflicts always and entirely by the race condition (from "*inside*"). Real systems with race conditions, like timers controling activities, *cannot* be modeled by TPNs with PS.

Discrete Event Systems The execution of a TPN can now be interpreted as a time dynamic discrete event system in various different ways. If oriented along the places $p \in P$, the token *arrival* and *departure* events characterize the dynamic behavior of a system. More natural is the transition oriented view, where transition firings are related to event occurrences, while places represent the conditions that need to hold for the event to occur; e.g. in Figure 3 the firing of t_1 represents the occurrence of a "*machine failure*" event, etc.

3 PARALLEL SIMULATION OF TPNs

Parallel discrete event simulation (PDES) of TPNs aims to exploit the specific features offered by SIMD operated environments, especially the hardware support for fast communication in a static, regular interconnection network of processors controlled by a centralized control unit. Not only the one-to-all broadcast operation is possible in $O(logN)$ time, N being the number of processors, but also the reduction of data values d_i from the N processors with respect to any binary associative operator $\circ$ (e.g. $+$, min, etc.), and, most importantly, the N partial products $\bigcirc_{j=1}^{i} d_j \quad i = 1 \ldots N$ (*parallel prefix* operation).

Time Stepping Obviously, TPNs with DT and PS find a straightforward *parallel* execution implementation, since start-firing and end-firing events can only occur at instants $k\Delta_t,\ k = 0, 1, \ldots$ of simulated time. Denote the state of such a TPN by (μ, ϱ), where μ is the marking and ϱ is a $T \times 1$ vector holding for every $t \in T$ the *remaining time to next event* (start-firing or end firing), then the algorithm at time $k\Delta_t$ executes events related to $t_i \in T$ for which $\varrho_i = 0$ in parallel (as long as there are $t_i \in T$ with $\varrho_i = 0$). After that, for all $\varrho_i > 0$ it sets $\varrho_i = \varrho_i - \Delta_t$ and steps to time $(k+1)\Delta_t$. Clearly, time gaps containing no event occurrences can be "overjumped" employing a *min*-reduction over ϱ to determine the earliest next event time. In principle, the same execution strategy applies to TPNs with AF and/or CT, but is likely to be less effective due to handling the descheduling with AF, and the far smaller probability of two or more events occuring at exactly the same instant of time (parallelism) with CT. Time stepping will perform best for TPNs with a very high concentration of events at certain points in virtual time, and finds an efficient implementation for SMs, MGs, and CFNs. A distributed conflict resolution scheme necessary for FCNs and GNs can severely complicate the implementation and degrade performance.

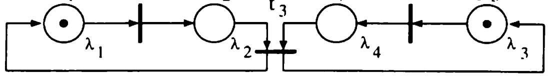

Figure 4: PN3: MG with Stochast. Holding Times

Recurrence Equations A more general way to execute certain classes of TPNs concurrently was developed by Baccelli and Canales (1993), describing in terms of recurrence equations how the TPN dynamically evolves. Consider exponential holding times associated with places p_i as $h_i(1), h_i(2), \ldots$, and (for simplicity) zero enabling/firing times in the MG in Figure 4. Then the occurrence of the k-th firing of $t_i \in T$, $ot_i(k) = ot(t_i(\bullet_k))$ can be written as:

$$ot_1(k) = h_1(k) \otimes ot_3(k-1),$$
$$ot_2(k) = h_3(k) \otimes ot_3(k-1),$$
$$ot_3(k) = h_2(k) \otimes ot_1(k) \oplus h_4(k) \otimes ot_2(k),$$

where $\oplus$ denotes *max* as an associative infix operator, and $\otimes$ denotes the $+$ operator. This set of equations together with $ot_1(0) = ot_2(0) = ot_3(0) = 0$ defines the evolution of PN3, and can, with $ot_3(k) = h_2(k) \otimes h_1(k) \otimes ot_3(k-1) \oplus h_4(k) \odot h_3(k) \otimes ot_3(k-1)$, be written in matrix form as:

$$ot(k) = ot(k-1) \otimes A_1(k) \otimes (E \oplus A_0(k)), \qquad (4)$$

$$A_0(k) = \begin{pmatrix} \epsilon & \epsilon & h_2(k) \\ \epsilon & \epsilon & h_4(k) \\ \epsilon & \epsilon & \epsilon \end{pmatrix} \quad A_1(k) = \begin{pmatrix} \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon \\ h_1(k) & h_3(k) & \epsilon \end{pmatrix}$$

where $\otimes$ and $\oplus$ are operations in the semiring $(\mathcal{R}, \oplus, \otimes)$ with null elements $\epsilon = -\infty$ for $\oplus$ and $e = 0$ for $\odot$; E is the identity matrix in $(\mathcal{R}, \oplus, \otimes)$. Indeed, (4) is a special form of

$$\begin{aligned} ot(k) &= (ot(k-1) \otimes A_1(k) \oplus \cdots \oplus ot(k-M) \otimes A_M(k)) \\ &\quad \otimes (E \oplus A_0(k) \oplus A_0(k)^2 \oplus \cdots \oplus A_0(k)^L) \end{aligned} \qquad (5)$$

where $A_n(k)$ are matrices representing with their elements ij the holding time of the k-th token in $p \in P$ with $O(t_i) = I(t_j) = p$ having n tokens in $\mu^{(0)}$, L is the maximum number of places on a directed cycle in MG not having a token in $\mu^{(0)}$, and M is the maximum number of tokens in any $p \in P$ in $\mu^{(0)}$. By appropriately rewriting (5) and using the associativity of $\otimes$ in $(\mathcal{R}, \oplus, \otimes)$, the underlying MG can be executed either in a *spatially decomposed* way (parallelism from the MG size), or in a *temporally decomposed* way by executing matrix multiplications that simulate the evolving of MG in distinct *epochs of time*, the results of which are *parallel prefixed* with $\otimes$ eventually. Bachelli and Canales (1993)

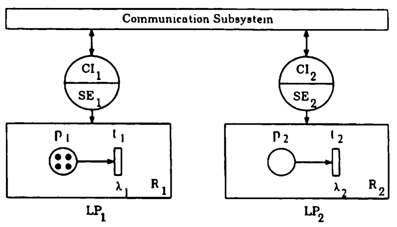

Figure 5: Logical Process simulation of TPNs

achieved excellent performance for MGs with DT or CT for both transitions and places, and FIFO token arrival/consumption. The extension of the approach to higher net classes is possible in principle, but not within $(\mathcal{R}, \oplus, \otimes)$.

4 DISTRIBUTED SIMULATION OF TPNs

Opposed to PDES, distributed discrete event simulations (DDES) execute TPNs as a spatially decomposed event system dispersed over processors that operate *asynchronously* in parallel (but not necessarily; note the implementation of a DDES scheme for MGs on a SIMD platform by Sellami et.al. (1994)). As such, DDES of TPNs is more general than the time stepping approach, since also transition firings occuring at different instants of virtual time are considered for parallel execution. However, DDES potentially induces higher overheads for event list management and complex synchronization protocols not present in PDES.

The basic idea of a DDES is to separate topological TPN parts to be simulated by *logical processes* (LPs). A synchronization protocol provides for a proper synchronization of intra and inter LP event relations with respect to simulated time and causal interdependencies. A DDES of a TPN is organized as in Figure 5 (see also Chiola and Ferscha (1993a)), where LPs comprise

R a spatial *region* of TPN: $R_i = (P_i \subseteq P,\ T_i \subseteq T,\ F_i \subseteq F)$ (all event occurrences in R_i will be simulated by LP_i),

SE an event driven *simulation engine* maintaining an event list (EVL), a *local* virtual time (LVT), and a static subset of the state variables $S_i \subseteq S$ defining the state of R_i at any time LVT (we will study cases with $S_i = \cup_j \mu_j \ \ \forall p_j \in P_i$), and

CI a *communication interface* triggering the local SE and synchronizing with other LPs. CI implements a synchronization protocol either based on the Chandy-Misra-Bryant scheme described by Misra (1986) (conservative), or on Time Warp described by Jefferson (1985) (optimistic).

Event Driven Simulation Engines EVL in SE_i holds pending future event occurrences, organized in increasing timestamp order. In case of e.g. AF semantics, these are tuples $\langle t_j@ot(t_j)\rangle$ denoting the scheduled firing of t_j at time $ot(t_j)$. Once SE_i gets permission from CI_i to fire the first (lowest occurrence time) transition t_j in state S_i, it (i) removes $\langle t_j@ot(t_j)\rangle$ from EVL, (ii) modifies the state variables according to $S \xrightarrow{t_j} S_i'$, (iii) schedules new tuples $\langle t_k@ot(t_k)\rangle$ for all $t_k \in E(S_i')$, and (iv), removes tuples $\langle t_l@ot(t_l)\rangle$ which have lost enabling in S_i' (only in the AF policy). Note that in target environments lacking a shared address space, for performance reasons it is very important to be able to verify $t_k \in E(S_i)$ without having to consult data residing in remote LPs (implicitly assumed here with the notation), thus restricting the freedom in partitioning.

Partitioning Thomas (1991) has proposed $R_i = (p \in P)$ (P-LPs) or $R_i = (t \in T)$ (T-LPs), setting up $\mathcal{P} + \mathcal{T}$ LPs. His "Transition Firing Protocol" (TFP) operates in two phases: (i) verify $t_k \in E(\mu)$, t_k residing in LP_{t_k}, and (ii) fire t_k. A fairly communication intensive double-handshake protocol is used for (i), involving the announcement of available tokens by P-LPs, the requesting of a certain amount of tokens necessary for enabling by T-LPs, the granting of the requested amount of tokens by P-LPs, and the confirmation of absorbation of that amount of tokens again by T-LPs. Basically TFP implements a (distributed) token competition resolution policy, but not a conflict resolution mechanism in the sense of the GSPN definition, where user defined *random switches* for competing transition selection can be specified. (ii) implements the removal (deposit) of tokens from (to) P-LPs. P-LPs and T-LPs employ different SEs, which in combination behave *conservatively* (P-LPs do only serve competing *earliest* requests).

Ammar and Deng (1991) allow a totally general decomposition of TPNs into regions, but with redundant proxy representations of places that are cut away from their output transitions. A Time Warp based communication interface managing five different types of messages is used to maintain consistency in the "overlapped" state representation (note that here S_i is not a pairwise disjoint partition of μ).

It has been seen by Thomas (1991), Nicol and Roy (1991) and Chiola and Ferscha (1993a), that for performance reasons the generality of R_i should be limited in such a way that conflicting transitions together with all their input places should always reside in the same LP, i.e. for $t_k \in T_i$, $t_k \in E(S_i)$ can always be verified without communication. A *minimum region partitioning* and *grain packing* strategy was proposed by Chiola and Ferscha (1993b), that uses two sources of partitioning information: (i) the TPN topology (P, T, F), and (ii) structural properties of the TPN in combination with $\mu^{(0)}$ (if available). Consider the following relations among transitions:

SC $t_i, t_j \in T$ are in *structural conflict* $(t_i\ SC\ t_j)$, iff $I(t_i) \cap I(t_j) \neq \emptyset$

CC $t_i, t_j \in T$ are *causally connected* $(t_i\ CC\ t_j)$, iff $t_i \in E(\mu)$ and $t_j \notin E(\mu)$, $\mu \xrightarrow{t_i} \mu'$ might cause that $t_j \in E(\mu')$.

ME $t_i, t_j \in T$ are *mutually exclusive*, $(t_i\ ME\ t_j)$, iff $\nexists \mu$ s.t. $t_i, t_j \in E(\mu)$.

CN $t_i, t_j \in T$ are *concurrent* (t_i *CN* t_j), if they are neither SC, nor CC, nor ME.

$t_i, t_j \in T$ are in *symmetric* SC (t_i *SSC* t_j) iff $((t_i\ SC\ t_j) \vee (t_j\ SC\ t_i)) \wedge \neg(t_i\ ME\ t_j)$. Note that $(t_i\ SC\ t_i)$, $(t_i\ SSC\ t_i)$. The transitivity and reflexivity of *SSC* allows to partition T into equivalence classes of potentially conflicting transitions by computing the transitive closures for $t \in T$ with respect to *SSC*, (denoted by $ECS(t)$, the extended conflict set of t), which gives the *minimun region partitioning* (MRP) of TPN. (We neglect the possibility of indirect conflict for simplicity here). $ECS(t)$ can be computed in polynomial time in a pre-partitioning analysis for GNs, and is trivially obtained for SMs, MGs, CFNs and FCNs. The MRP for PN1 is $R_1 = ((p_1, p_7, p_2), (t_1, t_2), F_1)$, $R_2 = ((p_3), (t_3), F_2)$, $R_3 = ((p_4), (t_4), F_3)$, $R_4 = ((p_5), (t_5), F_4)$, $R_5 = ((p_6), (t_6), F_5)$; for PN2 it is depicted in Figure 5.

Starting from the automatically generated MRP, regions of larger "grain size" can be obtained by collapsing R_is using "grain packing" heuristics. A few arguments are: (*i*) *ME* transitions do not bear any parallelism, s.t. it might not make sense to separate them into different regions: consider PN1 having just a single shared communication device, i.e. $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 1]^T$. Then together with the P-invariant $y_2 = [0, 0, 1, 1, 0, 0, 1]^T$ we have $\mu_3^{(i)} + \mu_4^{(i)} + \mu_7^{(i)} = 1$, which is a sufficient condition for (t_3 *ME* t_4), and we would "pack" R_2 and R_3 together. Moreover this is sufficient for (t_1 *ME* t_3) and (t_2 *ME* t_4) and we could collapse even R_1, R_2 and R_3. (*ii*) From $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 2]^T$ (two communication devices) together with $y_3 = [0, 1, 0, 1, 0, 1, 0]^T$ we have $\mu_2^{(i)} + \mu_4^{(i)} + \mu_6^{(i)} = 1$ which is a sufficient condition for (t_4 *CC* t_6) (even (t_4 *ME* t_6)), s.t. the corresponding regions R_3 and R_5 could be collapsed. (*iii*) From $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 2]^T$ it is trivially seen that (t_5 *CN* t_6), i.e. executing firings of t_5 and t_6 can indeed exploit model parallelism. Note that also (t_4 *CN* t_5), (t_3 *CN* t_6) and (t_3 *CN* t_4) in $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 2]^T$. As a principle, grainpacking should be done so as to separate CN transitions.

Communication Interfaces The purpose of the CI is to synchronize the transition firings local to the LP's region with firings in remote regions (LPs). The main mechanism for this is to communicate time stamped *tokenmessages* $m = \langle \#, i, ts \rangle$ among the LPs, where # is the number of tokens to be deposited in place p_i in the receiver LP, respecting its timestamp ts, i.e. a copy of the senders LVT at the sending instant. Once a tokenmessage is received by the CI of an LP, it integrates the token in the local event structure at LVT = ts to preserve global causalities. Both classical synchronization protocols can be applied for this integration: While Chandy-Misra-Bryant (CMB) based CIs (CI^{CMB}) would prevent the local SE from simulating local transition firings surpassing a LVT horizon for which the LP has been guaranteed not to receive a tokenmessage with ts less than that horizon (by simply blocking SE), a Time Warp based CI (CI^{TW}) would relax this restriction allowing SE to progress local simulation as far into the simulated future as there are scheduled events in EVL. Once a straggler tokenmessage (i.e. one with timestamp less than the current LVT) is received, the CI^{TW} would recover from the causality violation by undoing both local and consequential remote "overoptimistic" simulations in the rollback procedure. Data and control structures for CI^{CMB} and CI^{TW} were worked out in detail by Chiola and Ferscha (1993a).

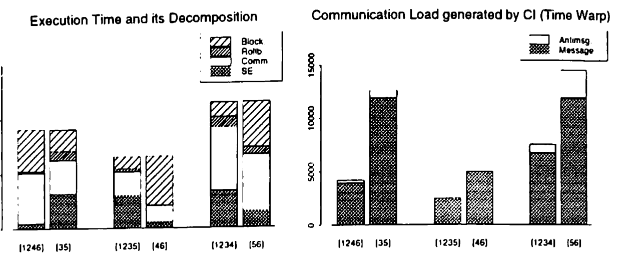

Figure 6: Performance of CI^{TW} on the CM-5

The performance of a DDES of PN1 as executing on the CM-5 with a Time Warp based, lazy cancellation CI is shown in Figure 6. Let $\Pi_1 = [1246][35]$ be a shorthand for the partitioning $R_1 = ((p_1, p_2, p_4, p_6, p_7), (t_1, t_2, t_4, t_6), F_1)$, $R_2 = ((p_3, p_5), (t_3, t_5), F_2)$, i.e. LP_1 is assigned t_1, t_2, t_4, t_6 with the respective inputplaces and LP_2 is assigned t_3 and t_5. Similarly let $\Pi_2 = [1235][46]$, $\Pi_3 = [1234][56]$. The potential gain from Π_1 and Π_2 is (t_3, t_5 *CN* t_4, t_6), i.e. t_3, t_5 and t_4, t_6 are pairwise *CN* in $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 2]^T$. The difference among Π_1 and Π_2 is that in Π_2 *ME* transitions on the same T-invariant $x_2 = [0, 1, 0, 1, 0, 1]$ are separated from each other: t_2 with (t_2 *ME* t_4, t_6) is separated from t_4, t_6 (transitions in $x_1 = [1, 0, 1, 0, 2, 0]$ (in case of Π_1) are not pairwise *ME*). A consequence of this is, that rollback *can never occur* in LP_2 with Π_2 (see middle columns in Figure 6). Moreover, no antimessage will ever be generated for (sent to) LP_2. The hope of Π_3 is (t_3 *CN* t_6) and (t_4 *CN* t_5). Note that Π_2, despite the violation of the grain packing argument (*i*), but due to avoiding communication induced by rollback (but suffering from blocking instead) outperforms Π_1 and Π_3 on the CM-5 for the particular $\mu^{(0)}$. The situation would change with a different $\mu^{(0)}$ (e.g. more than one I/O-processes in PN1), and might be different on a platform with a smaller communication-computation speed ratio.

As is seen from Figure 6, in CI^{TW} communication of token- and antimessages is dominating the performance – but the same is true for CI^{CMB} due to nullmessages. To avoid communication overhead as far as possible, Ferscha and Chiola (1994) proposed a CI that probabilistically controls the optimism in Time Warp: Let the event $e = \langle t@ot(t) \rangle$,

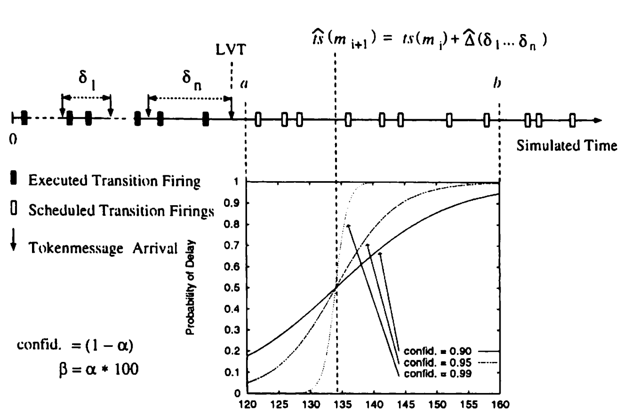

Figure 7: Probabilistic Optimism Control

$t \in T_i$ in some LP_i be *causal* ($\rightarrow$) for a future event $e' = \langle t'@ot(t) + \delta \rangle$, $t' \in T_i$ in LP_j, i.e. event e with some probability changes the state variables (marking) read by e'. Then the CMB "block until safe-to-process"-rule appears adequate for $P[e \rightarrow e'] = 1$, and is *overly pessimistic* for cases $P[e \rightarrow e'] < 1$, since awaiting the verification whether $(e \rightarrow e')$ or $(e \not\rightarrow e')$ hinders the (probably correct) concurrent execution of events e and e' in different LPs. Clearly, if $P[e \rightarrow e'] \ll 1$, an optimistic strategy could have executed e' concurrently to e most of the time (in repeated simulations). This argument mainly motivates the use of an optimistic CI for simulation models with nondeterministic event causalities like SMs, FCNs, and GNs. As a more specific argument assume the timestamp of the *next* (token-)message m_{i+1} to arrive at LP_j in the time interval $[a, b]$ (b can be arbitrarily large), i.e. $a \leq ts(m_{i+1}) \leq b$. Then LP_j could have safely simulated all the scheduled transition firings $\langle t_k@ot(t_k)\rangle$ with $ot(t_k) < ts(m_{i+1})$; CI^{CMB} however will block at some $LVT_j = a$ (Figure 7). As an example, let the probability of the actual timestamp of the forthcoming message be $P[ts(m_i) = x] = \frac{1}{b-a}$ $\forall x \in [b, a]$, then CMB, by blocking, fails to use a $(1-\alpha)$ chance to produce useful simulation work by progressing over a and executing scheduled transition firings in the time interval $[a, a + \alpha(b-a)]$. This is a clear *overpessimism* at least for small values of α. On the other hand, CI^{TW} would allow executing even scheduled firings $ot(t_k) > b$, which is a clear *overoptimism* because every firing $ot(t_k) > b$ definitely will have to be rolled back, incurring annihilation messages that could have been avoided.

The ***probabilistic*** CI (CI^{prob}) appears as a performance efficient compromise between CI^{CMB} and CI^{TW}, by executing scheduled firings with $a \leq ot(t_k) \leq b$ only with a certain probability, thus throtteling the optimism. After observing the past n message arrivals, CI^{prob} by statistically analyzing the arrival instants $ts(m_{i-n+1}), ts(m_{i-n+2}), \ldots ts(m_i)$ tries to estimate the timestamp of the next message as $\widehat{ts}(m_{i+1}) = ts(m_i) + \widehat{\Delta}(\delta_1, \delta_2, \ldots \delta_n)$, where $\delta_k = ts(m_{i-k+1}) - ts(m_{i-k})$ is the difference in timestamps

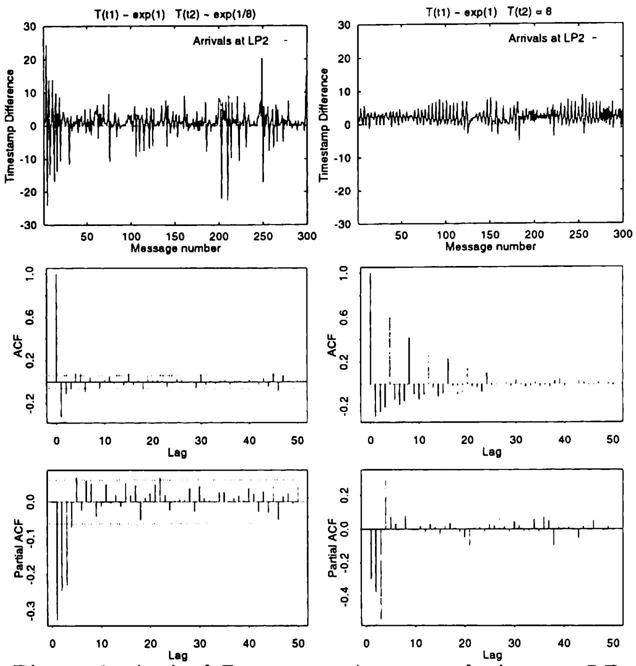

Figure 8: Arrival Processes, Autocorrelations at LP_2

of two consecutive messages. Let the level of confidence in the estimate be $(1-\alpha)$, then CI^{prob} gives SE permission to execute a scheduled firing of t_k, $a \leq ot(t_k) \leq b$ with probability

$$P[execute\ \langle t_k@ot(t_k)\rangle] = 1 - \frac{1}{1 + e^{-(\frac{LVT - \widehat{ts}}{\beta(1-\alpha)})}},$$

and "blocks" for the amount of the average CPU time used for executing one transition firing with $P[delay] = 1 - P[execute\ \langle t_k@ot(t_k)\rangle]$. Figure 7 explains that CI^{prob}, related to the confidence level (or forecast standard error), can be arbitrarily close to CI^{CMB} as well as to (a throtteled) CI^{TW} in behavior.

One approach to describe the arrivals is as an unknown stochastic process $\{X_i\} = (X_1, X_2, \ldots X_n)$ modeled by an autoregressive moving average process ARMA$[p, q]$ (see Brockwell (1991)), or an integrated ARIMA$[p, d, q]$ process if *stationarity* is obtained only after d-fold differencing the series. (Note that the more intuitive k-th order exponential smoothing and the special case ARIMA$[0, k, k]$ generate equivalent forecasts.) ARMA$[p, q]$ as defined by

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \qquad (6)$$

is the composition of a pure autoregressive process of order p (AR$[p]$) explaining X_t as a dependency $X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \epsilon_t$, ϵ_t being a white noise random error, and a pure moving average process of order q (MA$[q]$) that explains X_t as a series of i.i.d. white noise errors $X_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q}$ with $E(\epsilon_i) = 0$, $Var(\epsilon_i) = \sigma_\epsilon^2$ and $E(X_t) = 0$. Looking at the arrival process as obtained on the CM-5 for LP_2 of PN2 with $\tau(t_1) \sim exp(1)$ and $\tau(t_2) \sim exp(\frac{1}{8})$ (Figure 8, top left), and the corresponding autocorrelation (ACF) and partial ACF (PACF) leads us to hypothesize that the arrival process is MA, since ACF has a single spike at lag 1, and PACF dies down. With $\tau(t_1) \sim exp(1)$, $\tau(t_2) = 8$ we find ACF dying down in an oscillating damped exponential fashion (Figure 8 right) suggesting an AR model, etc. Indeed, we find best fittings with an ARIMA$[0,0,4]$ and an ARIMA$[3,0,0]$ for the two processes applying the classical Box Jenkins procedure: (i) model order identification (p,q,d), (ii) model parameter estimation (ϕ_i, θ_i), and (iii) model validation (Portmanteau test). The 1-step best linear forecast can now be used to predict the next message timestamp, and α, the confidence level of the goodness-of-fit test, can be directly used to control the optimism. By periodically rebuilding the model, CI^{prob} adapts the LP to a synchronization behavior directly reflecting the inherent model parallelism, and can also cope with *transient* arrival processes.

A CI based on *conservative time windows* was proposed by Nicol and Mao (1994), that lets SE execute all events e with $VT \leq ot(e) < VT + \omega$ without incurring any further communication among LPs, where VT is the (*global*) virtual time present in every LP after a barrier synchronization, and ω is the precomputed (global) time window. With respect to partitioning, their work avoids to make use of information other than the static net topology (P, T, F), but periodically attempts to redistribute LPs after monitoring the intensity and distribution of the workload. The initial mapping is based on a sophisticated heuristic for weighted directed graph partitioning.

ACKNOWLEDGMENTS

This paper was elaborated while the author was visiting the University of Maryland. The valuable support given by Satish K. Tripathi, the use of the resources at the Computer Science Dept. and the CM-5 at UMIACS are gratefully acknowledged.

REFERENCES

Ajmone Marsan, M., G. Balbo, G. Chiola, and G. Conte. 1987. Generalized Stochastic Petri Nets Revisited: Random Switches and Priorities. In *Proc. of the 2nd International Workshop on Petri Nets and Performance Models*, pages 44 - 53. IEEE-CS Press.

Ammar H. H. and S. Deng. 1991. Time Warp Simulation of Stochastic Petri Nets. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pages 186 - 195. IEEE-CS Press.

Baccelli, F. and M. Canales. 1993. Parallel Simulation of Stochastic Petri Nets using Recurrence Equations. *acm Transactions on Modeling and Computer Simulation*, 3(1):20 - 41.

Brockwell, P. J. and R. A. Davis. 1991. *Time Series: Theory and Methods.* Springer Verlag, New York.

Chiola, G. and A. Ferscha. 1993a. Distributed Simulation of Petri Nets. *IEEE Parallel and Distributed Technology*, 1(3):33 - 50.

Chiola, G. and A. Ferscha. 1993b. Distributed Simulation of Timed Petri Nets: Exploiting the Net Structure to Obtain Efficiency. In M. Ajmone Marsan, editor, *Proc. of the 14th Int. Conf. on Application and Theory of Petri Nets*, Lecture Notes in Computer Science 691, pages 146 - 165, Berlin. Springer Verlag.

David, R. and H. Alla. 1992. *Petri Nets & Grafcet. Tools for Modelling Discrete Event Systems.* Prentice Hall, NY.

Ferscha, A. and G. Chiola. 1994. Self Adaptive Logical Processes: The Probabilistic Distributed Simulation Protocol. In *Proc. of the 27th Annual Simulation Symposium*, pages 78–88, Los Alamitos, California. IEEE Computer Society Press.

Jefferson, D. A. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425.

Misra, J. 1986. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, 18(1):39–65.

Murata, T. 1989. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580.

Nicol, D. and W. Mao. 1994. Automated Parallelization of Timed Petri-Net Simulations. *submitted for publication.*

Nicol, D. M. and S. Roy. 1991. Parallel Simulation of Timed Petri-Nets. In B. Nelson, D. Kelton, and G. Clark, editors, *Proceedings of the 1991 Winter Simulation Conference*, pages 574 - 583.

Ramchandani, Ch. 1974. Analysis of Asynchronous Concurrent Systems by Petri Nets. Technical report, MIT, Laboratory of Computer Science, Cambridge, Massachusetts.

Sellami, H., J. D. Allen, D.E. Schimmel, and S. Yalamanchili. 1994. Simulation of Marked Graphs on SIMD Architectures using Efficient Memory Management. In *Proc. of MASCOTS'94*, pages 343 - 348. IEEE-CS Press.

Sifakis, J. 1977 Use of Petri Nets for Performance Evaluation. In: H. Beilner and E. Gelenbe, Editors, *Measuring, Modelling and Evaluating Computer Systems*, pages 75 - 93. North-Holland.

Thomas, G. S. 1991. Parallel Simulation of Petri Nets. Technical Report TR 91-05-05, Dep. of Computer Science, University of Washington.

AUTHOR BIOGRAPHY

ALOIS FERSCHA has been with the Department of Applied Computer Science and Information Systems at the University of Vienna, Austria, since 1986, where he received the Dr. degree in social and economic sciences in 1990. His current research interests include performance modeling and prediction of parallel programs, computer aided performance engineering of parallel software, distributed simulation and Petri nets.