A Distributed Discrete Event Simulation Framework for Timed Petri Net Models

G. Chiola*, A. Ferscha†

* Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, I-10149 Torino, Italy

† Institut für Statistik und Informatik, Universität Wien
Lenaugasse 2/8, A-1080 Vienna, Austria

Abstract

Parallel and distributed simulation techniques are consolidating as a potentially effective way of improving elapsed time for discrete event simulation experiments of large system models. The common aim to all the approaches is to divide a single simulation program into logical processes (LPs) to be executed concurrently on individual processing nodes of a parallel computer. Two issues are critical in this concern: First there is the problem of defining protocols that allow an efficient, distributed handling of local event simulations in the LPs while at the same time maintaining the proper causality among events, and secondly the problem of partitioning of large simulation models into a number of cooperating simulation LPs an mapping them onto processing nodes. Most of the literature so far has concentrated on queueing networks as simulation models, trying to tackle these problems.

In this work timed Petri nets will be studied as specifications of simulation models, and both the development (or adaptation) of distributed simulation protocols, as well as the partitioning of large Petri net models will be studied in order to optimize the elapsed (parallel) simulation time. Distributed simulation mechanisms based on the two classical approaches (conservative, optimistic) for timed transition Petri nets (TTPNs) are systematically developed. The proposed partitioning is based on characteristics and mathematical properties of the structure of the Petri net model that can be effectively computed off-line before the start of the simulation experiment. These properties statically identify portions of models that potentially yield concurrent events and portions that have no chance to yield useful parallelism. Based on this information an automated partitioning into minimum grain size LPs is possible, and rules for packing grains can be given in order to find an optimum balance of communication and computation requirements of the distributed simulation to fit to dedicated multiprocessor systems.

The distributed simulation framework has been implemented on three multiprocessor hardware platforms (Intel iPSC/860, Transputer and Sequent Balance). Speedup characteristics of sample TTPNs on these machines are compared, and the performance influences are categorized according to empirical observations. This work demonstrates, that distributed simulation on a real multiprocessor system can in fact gain speedup over the sequential simulation – this can be achieved even for very small scale simulation models.

1 Introduction

The popularity of the timed Petri net modeling and analysis framework [Ramc 74, Merl 76, Sifa 77, Rama 80, Moll 82, Razo 84, Duga 84, Ajmo 84, Holl 87, Ajmo 87] for the quantitative (and simultaneously qualitative) study of asynchronous concurrent systems (e.g. [Fers 90]) is mainly due to the availability of software packages [Feld 92] automating the analysis algorithms and often providing a powerful, graphic user interface [Balb 89, Chio 91].

Although the use of Petri net tools shows that complex simulation models are hard and costly to evaluate, simulation is often the only practical analysis means. Parallel and distributed simulation techniques have in some practical applications, and at least in theory, improved the performance of large systems simulation, but are still far from a general acceptance in science and engineering. The main reasons for this pessimism are the lack of automated tools and simulation languages suited to parallel and distributed simulation, as well as performance limitations inherent to the simulation strategies.

This paper points out the special suitability of the Petri net formalism for parallel and distributed simulation by automating the exploitation of useful model parallelism by structural Petri net analysis, and by designing the simulation strategy in order to optimize overall execution time. Causal relations among events are explicitly described by the Petri net structure so that a proper exploitation of this information may potentially improve the performance of general purpose, distributed simulation engines. A TTPN is decomposed into a set of spatial regions that bear potentially concurrent events, based on a preanalysis of the net topology and initial marking.

The decomposed TTPN is assigned to multiple processors with distributed memory, working asynchronously in parallel to perform the global simulation task. Every processor hence simulates events according to a local (virtual) time and synchronizes with other processors by rollbacks of its local time caused by messages with time stamp in the local past — thus no synchronous communication is required at the software level (neither the send nor the receive message operations are blocking to the processors). Input buffers are assumed to be available for every processor to keep arriving messages until they are withdrawn by an input process between the simulation of two consecutive events.

1.1 Timed Transition Petri Nets

In the sequel of this work we consider the class of timed Petri net models where timing information is associated to transitions formally summarized as [Mura 89]:

Definition 1.1 *A timed transition Petri net is a tuple* $TTPN = (P, T, F, W, \Pi, \tau, M_0)$ *where:*

(i) $P = \{p_1, p_2, \ldots, p_{n_P}\}$ *is a finite set of P-elements (places),*
$T = \{t_1, t_2, \ldots, t_{n_T}\}$ *is a finite set of T-elements (transitions)*
with $P \cap T = \emptyset$ *and a nonempty set of nodes* $(P \cup T \neq \emptyset)$*.*

(ii) $F \subseteq (P \times T) \cup (T \times P)$ *is a finite set of arcs between P-elements and T-elements denoting input flows* $(P \times T)$ *to and output flows* $(T \times P)$ *from transitions.*

(iii) $W : F \mapsto \mathbb{N}$ *assigns weights* $w(f)$ *to elements of* $f \in F$ *denoting the multiplicity of unary arcs between the connected nodes.*

(iv) $\Pi : T \mapsto I\!N$ *assigns priorities* π_i *to* T*-elements* $t_i \in T$.

(v) $\tau : T \mapsto I\!R$ *assigns firing delays* τ_i *to* T*-elements* $t_i \in T$.

(vi) $M_0 : P \mapsto I\!N$ *is the marking* $m_i = m(p_i)$ *of* P*-elements* $p_i \in P$ *with tokens in the inital state of* $TTPN$ *(initial marking).*

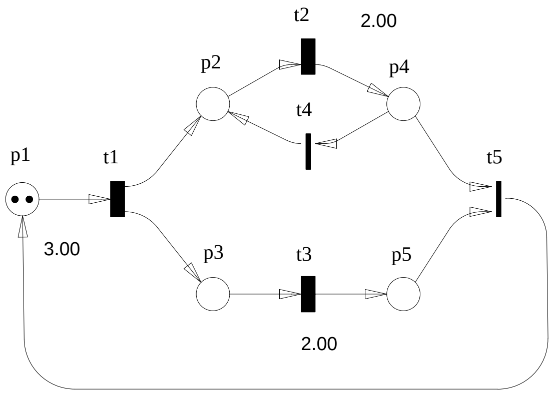
A transition t_i with $\pi_i = \Pi(t_i)$ is said to be *enabled with degree* $k > 0$ *at priority level* π_i in the marking M, iff $k > 0$ is the maximum integer such that $\forall p_j \in {}^\bullet t_i$, $m(p_j) \geq k\ w(p_j, t_i)$ in M (${}^\bullet t_i = \{p_j \mid (p_j, t_i) \in F\}$ and $t_i^\bullet = \{p_j \mid (t_i, p_j) \in F\}$). t_i is said to be *enabled with degree* $k > 0$, iff there are no transitions enabled with positive degree on a priority level higher than π_i. The multiset of all transitions enabled in M (with positive enabling degree representing the multiplicity of the transition) is denoted $E(M)$. A transition instance t_i that has been continuously enabled during the time τ_i must *fire*, in that it removes $w(p_j, t_i)$ tokens from every $p_j \in {}^\bullet t_i$ by at the same time placing $w(t_i, p_k)$ tokens into every $p_k \in t_i^\bullet$. This firing of a transition takes zero time and is denoted by $M[t_i\rangle M'$.

1.2 Discrete Event Simulation of TTPNs

Discrete event simulation (DES) when executed sequentially repeatedly processes the occurrence of events in simulated time by maintaining a time ordered *event list* (EVL) holding time stamped events already scheduled to occur in the future, a (global) *clock* indicating the current time and *state variables* defining the the current state of the system. A *simulation engine* (SE) drives the simulation by continuously taking the first event out of the event list (i.e. the one with the lowest time stamp), simulating the effect of the event by changing the state variables and/or scheduling new events in the event list – possibly also remove obsolete events. This is being performed until some pre-defined endtime is reached, or there are no further events to occur.

In discrete event simulations of TTPNs a natural correspondence between events and transition firings is exploited [Duga 84, Ciar 89, Balb 89]. The occurrence of an event in the simulation system relates to the firing of a transition in the TTPN model. The event list hence carries transitions and the time instant at which they will fire, given that the firing does not become obsolete in the meantime. The state of the system is represented by the current marking (M), which is being changed by the processing of an event, i.e. the firing of a transition: the transition with the smallest time stamp is withdrawn from the event list, tokens are removed from its inputplaces and deposited in its outputplaces. The new marking however can enable new transitions or disable enabled transitions, such that the event list has to be corrected accordingly: new enabled transitions are scheduled with their firing time to occur in the future by inserting them into the EVL, while (new) disabled transitions are removed. Finally the simulated (or virtual) time (VT) is set to the timestamp of the transition just simulated. It is obvious that always the transition with smallest timestamp is simulated first, otherwise the future of the simulation could have an impact onto the past – which would yield *causality errors*. A simple TTPN example is given in figure 1 demonstrating a sample simulation.

Note that in this example there are situations where several transitions have identical smallest time stamps, e.g. in step 5 all scheduled transitions have identical end firing time instants. This is not an exceptional situation but appears whenever two or more events *(potentially) can* occur or *(actually) do* occur simultaneously. The latter distinction is very important for all our further considerations,



	VT	M	EVL
1		(2,0,0,0,0)	<t1 @ > <t1 @ >
2		(1,1,1,0,0)	<t1 @ > <t2 @ > <t3 @ >
3		(0,2,2,0,0)	<t2 @ > <t2 @ > <t3 @ > <t3 @ >
4		(0,2,1,0,1)	<t2 @ > <t2 @ > <t3 @ >
5		(0,1,1,1,1)	<t2 @ > <t3 @ > <t4 @ > <t5 @ >
6		(0,0,1,2,1)	<t3 @ > <t4 @ > <t4 @ > <t5 @ >
7		(1,0,1,1,0)	<t3 @ > <t4 @ > <t1 @ >
8		(1,1,1,0,0)	<t3 @ > <t2 @ > <t1 @ >
9		(1,1,0,0,1)	<t2 @ > <t1 @ >
10		(1,0,0,1,1)	<t4 @ > <t5 @ > <t1 @ >
11		(2,0,0,0,0)	<t1 @ > <t1 @ >
12		(1,1,1,0,0)	<t1 @ > <t2 @ > <t3 @ >
13		(1,0,1,1,0)	<t1 @ > <t3 @ > <t4 @ >
14		(0,1,2,1,1)	<t3 @ > <t4 @ > <t2 @ > <t3 @ >
15		(0,1,1,1,1)	<t4 @ > <t5 @ > <t2 @ > <t3 @ >
16		(1,1,1,0,0)	<t2 @ > <t3 @ > <t1 @ >
17		(1,1,0,0,1)	<t2 @ > <t1 @ >
18		(1,0,0,1,1)	<t4 @ > <t5 @ > <t1 @ >
19		(2,0,0,0,0)	<t1 @ > <t1 @ >
20		(1,1,1,0,0)	<t1 @ > <t2 @ > <t3 @ >

Figure 1: Discrete event simulation of a sample Petri net

and will have severe impact on the construction of parallel or distributed simulation engines: $t2$ and $t3$ are scheduled to fire at time 5 (their enabling lasted for the whole period of their firing time $\tau(t2) = \tau(t3) = 2$), where firing of one of them will not interfere the firing of the other one. $t2$ and $t3$ are said to be *concurrent* since their firings are not interrelated. Obviously $t2$ and $t3$ could be simulated in parallel, say $t2$ by some processor P1 and $t3$ by another processor P2, or, as an improvement of sequential simulation, could both be removed from the EVL in a single simulation step. The situation is somewhat different with $t4$ and $t5$, since firing of one of them will disable the other one – $t4$ and $t5$ are said to be in *structural conflict*. The effect of simulating one of them would (besides changing the marking) also be to remove the other one from the EVL, since it has lost enabling. $t4$ and $t5$ preclude parallel simulation.

Before following the idea of simulating TTPNs in parallel, we can first systematically utilize the information within the net structure to describe a SE for sequential TTPN simulation. In the example we have seen that some relations among transitions can help to establish necessary conditions for a transition firing to disable other transitions (i.e. conflict). Of course it is also possible to derive conditions for transition firings enabling others. The exploitation of these relations represent a source of potential improvement of the simulation procedure, since the *range of effects* of transition firings (upon the EVL) can be determined statically in a preanalysis of the TTPN net structure, thus reducing the search space of potential effects. The enabling and disabling effects of a single transition $t_i \in T$ are obtained as follows:

enabling Let $t_i, t_j \in T$ be *causally connected*, i.e. given $t_i \in E(M)$ and $t_j \notin E(M)$, then the firing $M[t_i\rangle M'$ might cause that $t_j \in E(M')$. Let further be $CC(t_i)$ the set of all transitions causally connected to t_i. Then the firing of t_i has a potential enabling effect only on transitions in $SPE(t_i) = CC(t_i)$. In our example we have $SPE(t1) = CC(t1) = \{t2, t3\}$, $SPE(t2) = \{t4, t5\}$,

$SPE(t3) = \{t5\}$, $SPE(t4) = \{t2\}$, $SPE(t5) = \{t1\}$.

disabling Analogously a *set of potential disablings* $SPD(t_i)$ can be defined: Let $t_i, t_j \in T$ be *mutually exclusive*, i.e. they cannot be simultaneously enabled in any reachable marking. Let $ME(t_i)$ be the set of all transitions mutually exclusive to t_i. Then the firing of t_i has a potential disabling effect only onto transitions in $SPD(t_i) = SC(t_i) - ME(t_i)$, where $SC(t_i)$ is the set of transitions in structural conflict to t_i. For the example: $SPD(t1) = SPD(t2) = SPD(t3) = \{\}$, $SPD(t4) = \{t5\}$, $SPD(t5) = \{t4\}$.

The gain of this TTPN structure exploitation is that after every transition firing only $SPE(t)$ and $SPD(t)$ have to be checked for enabling (disabling) by the SE, rather than the whole T. This is a significant improvement to the management of the EVL if the set of transitions T is (very) much larger than $\mid SPE(t) \mid$ or $\mid SPD(t) \mid$.

More than one event occurring at the same instant of time, as e.g. the two transitions $t2$ and $t3$ in the example net in figure 1, is not the only kind of model parallelism exploitable for distributed simulation, in fact the degree of attainable parallelism and the grain size of pieces of work would be too small to give a basis for a useful implementation on a multiprocessor. Obviously it is more desirable to also simulate events with different occurrence times concurrently in order to enlarge the pieces of work done in parallel. A parallel/distributed simulation of that type although has to take care of the causality of events. Two simulation strategies guaranteeing causal safety will be described in the sequel.

1.3 Approaches to DDES in the Literature

Distributed simulation strategies generally aim at dividing a global simulation task into a set of communicating *logical processes* (LPs) trying and exploiting the parallelism inherent in the concurrent execution of these processes. With respect to the target architecture we distinguish among *parallel discrete event simulation* (PDES) if the implementation is for a tightly coupled multiprocessor allowing simultaneous access to shared data (shared memory machines) (e.g. [Kona 91]), and *distributed discrete event simulation* (DDES) if loosely coupled multiprocessors with communication based on message passing are addressed (e.g. [Fuji 88a] [Fuji 88b]). In this work we only restrict to DDES. Notice however that the simulation algorithms developed for DDES trivially adapt to the execution on shared memory multiprocessors without performance loss.

Indeed event processing must respect causality constraints even in parallel and distributed simulation: classical DDES approaches preserve causality by simply enforcing timestamp ordering in event processing. Two main protocols have been proposed for the implementation of DDES: *conservative* that enforces time stamp order processing by restricting the parallelism among LPs and *optimistic* that instead allows out of timestamp order to occur and than correct the situation by "undoing" part of the simulation. Both, the *conservative* [Chan 79] and the *optimistic* strategy [Jeff 85a] have led to implementations of general purpose discrete event simulators on parallel and distributed computing facilities (for a survey see [Kaud 87], [Fuji 90]). The main potential contribution of the use of Petri net model descriptions for DDES is to define causality relations explicitly through the graph structure, thus allowing one to relax constraints on timestamp ordering for the processing of events that are not causally related. From this consideration we can thus expect to be able to increase the parallelism of a DDES engine with respect to the standard approaches by just taking the structure of the PN model into proper account.

In conservative DDES strategies all interactions among LPs are based on timestamped event messages sent between the corresponding LPs. Out-of-time-order processing of events in LPs is prevented by forcing LPs to block as long as there is the possibility for receiving messages with lower timestamp, which in turn has a severe impact on simulation speed. Optimistic DDES strategies weaken this strict synchronization rule by allowing a preliminary causality violation, which is corrected by a (costly) detection and recovery mechanism usually called rollback (in simulated virtual time). In [Misr 86] a DDES communication protocol (deadlock avoidance) is proven to be correct up to the time revealed by the smallest message timestamp in the system, and free of deadlocks caused by cyclic waiting for messages. The protocol introduces null messages to announce the absence of messages and promise that no message will be sent with a timestamp less than that of the null message. Especially in DDES this approach encounters a dramatic burden on the simulation [De V 90], and additionally does not cope with cycles of zero timestamp increment. A radical elimination of the use of null messages [Chan 81] (deadlock detection and recovery) allows the occurrence of deadlocks, and proposes a global controller process to detect deadlocks, which is broken by allowing the LP with the smallest timestamp to proceed (which is always safe). Naturally a global controller violates the principles of distributed computing. Current activities with the conservative approach concentrate on the detection of optimal lookahead information to speed up simulation [Lin 90, Yu 91, Nico 91].

In order to guarantee a proper synchronization among LPs in optimistic simulation, two rollback mechanisms have been studied. In the original Time Warp [Jeff 85a] [Jeff 85b] an LP receiving a message with timestamp smaller than the LPs locally simulated time (*straggler message*) starts a *rollback* procedure right away. Rollback is initiated by revoking all effects of messages that have been processed prematurely by restoring to the nearest safe state, and by neutralizing, i.e. sending "antimessages" for messages sent since that state (*aggressive cancellation*, *ac*). The impact of the erroneously sent messages is that also succeeding LPs might be forced to rollback, thus generating a rollback chain that eventually terminates. Reducing the size of the rollback chain is attempted by "filtering" messages with preempted effects [Prak 91], by postponing erroneous message annihilation until it turns out that they are not reproduced in the repeated simulation (*lazy cancellation*, *lc*) [Gafn 88], or by maintaining a causality record for each event to support *direct cancellation* of messages that will definitely not be reproduced [Kona 91]. The performance of rollback mechanisms is investigated in [Lin 91] and [Luba 91].

1.4 Paper organization

The balance of the rest of this report is the following.

Section 2 introduces classical concepts of DDES and adapts them to the particular case of TTPN models. Different kinds of customizations of DDES strategies and partitioning in concurrent LPs are discussed in a general framework.

Section 3 presents empirical results derived from the implementation of several variations of DDES strategies for TTPN models on 3 different multiprocessor architectures. Results are analyzed to identify weaknesses and potentialities of the different alternatives and to validate some of the proposed net specific optimizations.

Section 4 outlines additional optimizations that may be introduced in the most promising DDES strategy to take into account particularly favorable net topologies.

Section 5 contains concluding remarks and perspectives for future developments of the research.

2 A Distributed Simulation Framework for TTPNs

2.1 Logical Processes

In the most general case of distributed simulation of TTPNs we consider a decomposition of the "sequential" discrete event simulation task into a set of communicating LPs to be assigned to individual processing elements working autonomously and communicating with each other for synchronization. Each and only one LP is assigned to a dedicated processor and is residing there for the whole real simulation time. A message transportation system is assumed to be available (either implemented fully in software or by hardware routing facilities) that implements directed, reliable, FIFO communication channels among LPs. Every LP applies one and the same discrete event strategy to the simulation of the local subnet.

We identify the work partition assigned to LPs, the LPs' communication behavior and the LPs' simulation strategy as the constituent parts of a distributed simulation framework, and will in further treat these three components in a very abstract form.

Definition 2.1 *A* Logical Process *is a tuple* $LP_k = (TTPNR_k, I_k, SE)$ *where*

(i) **region**
$TTPNR_k = (P_k, T_k, F_k \subseteq (P_k \times T_k) \cup (T_k \times P_k), W, \Pi, \tau, M_0)$ *is a (spatial)* region *of some* $TTPN$ *such that* $P = \bigcup_{k=1}^{N_{LP}} P_k$, $T = \bigcup_{k=1}^{N_{LP}} T_k$, $F = \bigcup_{k=1}^{N_{LP}} F_k \bigcup_{i \in neighborhood(TTPNR_k)} ((P_k \times T_i) \cup (T_k \times P_i))$, *which are again TTPNs in the sense of Definition 1.1,*

(ii) **communication interface**
$I_k = (CH, m)$, *the* communication interface *of* LP_k *is a set of communication channels* $CH = \bigcup_{k,i} ch_{k,i}$ *where* $ch_{k,i} = (LP_k, LP_i)$ *is a directed channel from* LP_k *to* LP_i, *corresponding to an arc* $f \in F$ *and carrying messages* $m = \langle w(f), D, TT \rangle$ *where* $w(f)$ *(*w_f *for short) is the number of tokens transferred,* D *is an identifier of the destination (place or transition) and* TT *(token time) is the timestamp of the local virtual time of the sending* LP_k *at send time.*

(iii) **simulation engine**
SE *is a* simulation engine *implementing the simulation strategy.*

The set of all LPs $LP = \bigcup_k LP_k$ *together with directed communication channels* $CH = \bigcup_{k,i} ch_{k,i} = \bigcup_{k,i}(LP_k, LP_i)$ *constitute the* Graph of Logical Processes $GLP = (LP, CH)$.

Within the frame of this general definition of a logical simulation process a variety of different distributed simulations are possible. Not all combinations of decompositions into regions, communication interfaces and simulation engines make sense in practical situations: most of the combinations will not conform to the expectations of increasing simulation speed over sequential simulation. Moreover they are highly interrelated in that e.g. the simulation engine highly determines the communication interface and the appropriateness of decompositions.

To the best of our knowledge three approaches have been developed so far in the literature ([Thom 91b], [Nico 91], [Amma 91]), which would be categorized according to our framework as follows.

In [Thom 91a] [Thom 91b] an LP is created for every $p \in P$ and every $t \in T$, resulting in a very large number of LPs ($| LP |=| P | + | T |$) of smallest possible grain size, where the maximum number of arcs between any pair of LPs is 1. The idea behind this partition choice is to maximize the "potential parallelism" of the DDES in terms of number of LPs. For every arc an unidirectional channel with the same orientation as the arc is introduced. For every input arc from a place's LP and a transition's LP another unidirectional channels with the opposite orientation is required (control channel) in order to solve conflicts. The I_k proposed is based on a protocol invoking four messages ("activate," "request," "grant," and "confirm") among a transition's LP and all its input places' LPs in order to fire the transition. Obviously the (possibly tremendous) amount of messages inherent to this approach prevents efficiency if simulated in a distributed memory environment (their protocol has been simulated on a shared memory multiprocessor). The simulation engine used follows the *conservative* strategy.

Another conservative approach has been introduced by [Nico 91], allowing *"completely general"* decompositions into regions and employing a simulation engine that handles three kinds of events: the arrival of a token, the start and the end of transition firing in order to exploit lookahead. The generality of the partition is limited in such a way that conflicting transitions together with all their input places are always put together in the same LP, in order to avoid distributed conflict resolution.

Recently [Amma 91] a Time Warp simulation (i.e., a simulation engine following the optimistic approach) of SPNs has been proposed also allowing any partitioning in the $TTPNR_k$ sense, but with a redundant representation of places (and the corresponding arcs) in adjacent LPs. Their I_k maintains time consistency of markings among LPs by exchanging five different types of messages. The freedom of allowing any partitioning that stems from arbitrary arc cutting makes the message exchange protocol and the simulation engine unnecessarily complex.

In the following we will systematically work out *useful*, and in further *improved* combinations of TTPN regions, communication interfaces and simulation engines.

2.2 TTPN Regions

The most natural decomposition of work is the spatial partitioning of a TTPN into disjoint *region*s TTPNR$_k$ representing small(er) sized TTPNs. This also supports the application of one and the same simulation engine to all regions. (In the next section we will follow the argument, that such a simulation engine must be a full TTPN discrete event simulator, i.e. be able to also simulate the whole nonpartitioned TTPN).

Apparently the decomposition of a TTPN has strong impact onto the DDES performance. Small sized regions naturally raise high communication demands, whereas large scale regions can help to clamp local TTPN behavior inside one LP. The common drawback of the intuitive partitionings in [Thom 91b] and [Amma 91] is the necessity of developing a proper communication protocol among LPs in order to implement a distributed conflict resolution policy for transitions sharing input places. In a message passing environment for inter process communication, such a conflict resolution protocol may induce substantial overhead in the distributed simulation, thus nullifying the advantages of the potential model parallelism on the simulation time. "Arbitrary arc cutting" for partitioning can burden performance since arcs $f \in (P \times T)$ have a (severe) consequence on the efficient computation of enabling and firing, whereas arcs out of $(T \times P)$ have not such consequence. The latter is the reason why the partitioning has to be related to the TTPN firing rule [Ferr 91, Nico 91, Chio 93]: an LP is a minimum set of transitions T_k along with its input set (${}^\bullet(T_k) = \bigcup_{i|t_i \in T_k} {}^\bullet t_i$) such that local information is sufficient to decide upon the enabling and firing of any $t_i \in T_k$. This is in order to minimize conflict resolution overhead, since in this way conflict resolution always occurs internally to an LP, thus involving no communication overhead.

2.2.1 Minimum Regions

Let $CS(t_i)$ be the set of all transitions $t_j \in T$ in structural conflict with t_i ($t_i, t_j \in T$ are said to be in *structural conflict* denoted by (t_i SC t_j), iff exists M such that $t_i, t_j \in E(M)$, and $M[t_i\rangle M'$ might cause that $t_j \notin E(M')$). In the case of GSPNs with priority structures [Ajmo 91] *extended conflict set* $ECS(t_i)$ have been defined as the sets of all transitions $t_j \in T$ in indirect structural conflict with t_i. To relate the region partitioning to the firing semantics definition of TTPNs, all transitions within the same CS (or ECS) have to be included in the same TTPNR$_k$, since a distribution over several regions would require distributed conflict resolution (which would in any case degrade performance). This consideration leads directly to a partitioning into *minimum regions*, i.e. TTPNR$_k$s that should not be subdivided further in order to be practically *useful*:

Definition 2.2 *TTPNR$_k$ = $(P_k, T_k, F_k, W, \Pi, \tau, M_0)$ of some TTPN, $T = \bigcup_{k=1}^{N_{LP}} T_k$ where $T_k = \{t_i\}$, $t_i \in T$ is a single transition iff $t_i \in T$ does not participate in any ECS or $T_k = ECS(t_i) \subseteq T$, $t_i \in T$ is the whole ECS iff $t_i \in T$ participates in some ECS, $P_k = \bigcup_{t_i \in T_k} {}^\bullet(t_i)$ and $F_k \subseteq (P_k \times T_k)$ is called a* minimum region *of TTPN.*

Any general TTPN can always (and uniquely) be partitioned into *minimum regions* according to Definition 2.2.

2.3 The Communication Interface

Of course, the decomposition of TTPNs requires an interface among TTPNRs preserving the behavioral semantics of the whole TTPN. Such an interface has to be implemented by an appropriate communication protocol among the regions, according to the strategy chosen for the simulation engines.

The most general case of a communication interface as in definition 2.1 would cause — as seen e.g. in [Thom 91b] and [Amma 91] — a nonjustifyable complication of the simulation engine (handling of different types of messages) and an enormous amount of messages in the distributed memory multiprocessor. A more *useful* communication interface is obtain by restricting its generality, also as a logical consequence of the arguments used for the minimum region decomposition.

Since we have forced the regions not to distribute $p \in P_i$ and $t \in T_k$ over different regions TTPNR$_i$ and TTPNR$_k$ if there exists an arc $f = (p, t) \in (P_i \times T_k)$ (remember that for each $TTPNR_k$, $t_i \in T_k \Rightarrow {}^\bullet(t_i) \in P_k$), we can map the sets of arcs $(T_k \times P_j) \subseteq F$ interconnecting different TTPN regions to the channels of the communication interface. In other words we define $I_k = (CH, m)$ of TTPNR$_k$ to be the communication interface with channels $CH = \bigcup_{k,i} ch_{k,i}$, where $ch_{k,i} = (LP_k, LP_i)$

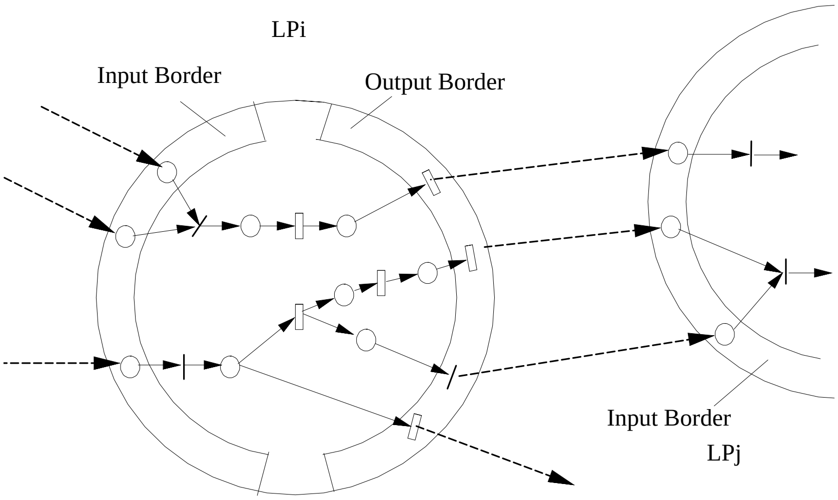


Figure 2: TTPN Region for a Logical Process.

is a set of directed channel from LP_k to LP_i, corresponding to all the arcs $(t_l, p_m) \in (T_k \times P_i) \subseteq F$, $t_l \in T_k$, $p_m \in P_i$, carrying messages m of identical type. A message m represents tokens to be deposited in places of adjacent LPs due to the firing of local transitions. Since all tokens to be transferred from LP_k to LP_i are to be sent over one and the same channel (which will be mapped onto a physical communication device of the multiprocessor system), a considerable reduction of communication complexity is expected from this restriction.

Moreover the description (and implementation) of the interaction among LPs is also simplified in the following way. Consider two TTPN regions TTPNR_i and TTPNR_j, and let $T_i^{\triangleright j} = \bigcup_l t_l \mid (t_l, p) \in (T_i \times P_j)$ be the set of transitions in TTPNR_i incident to arcs pointing to places in TTPNR_j, and analogously $P_j^{\triangleleft i} = \bigcup_l p_l \mid (t, p_l) \in (T_i \times P_j)$ the set of places in TTPNR_j incident to arcs originating from transitions in TTPNR_j, then the interaction of LP_i and LP_j only concerns $T_i^{\triangleright j}$ and $P_j^{\triangleleft i}$. We call $T_i^{\triangleright j}$ ($P_j^{\triangleleft i}$) the *output border* (*input border*) of TTPNR_i *towards* TTPNR_j (of TTPNR_j *from* TTPNR_i). $T_k^{\triangleright} = \bigcup_i T_k^{\triangleright i}$ is the *output border* of TTPNR_k, $P_k^{\triangleleft} = \bigcup_i P_k^{\triangleleft i}$ is the *input border*. (Naturally, $P_k^{\triangleleft} = P_k$ and $T_k^{\triangleright} = T_k$ for a minimum region.)

As a static measure of the potential communication induced by LP_i we can define its *arc-degree* (*of connectivity*) as the number of arcs $f \in F$ originating in TTPNR_i and pointing to places in adjacent TTPNRs and vice-versa:

$$\mathrm{AD}(TTPNR_i) = \Big| \bigcup_{k \in \mathrm{neighborhood}(TTPNR_i)} (P_i \times T_k) \cup (T_k \times P_i) \Big|$$

Similarly we define the *channel-degree* as the number of channels connected to a region:

$$\mathrm{CD}(TTPNR_i) = \Big| \bigcup_k ch_{i,k} \Big| + \Big| \bigcup_k ch_{k,i} \Big|$$

Note that these measures can be evaluated in a static preanalysis of the decomposed TTPN.

Figure 2 shows a TTPN region to become the "local simulation task" of some LP. The set of dashed TTPN arcs from LP_i to LP_j represent the directed communication channel among them. The

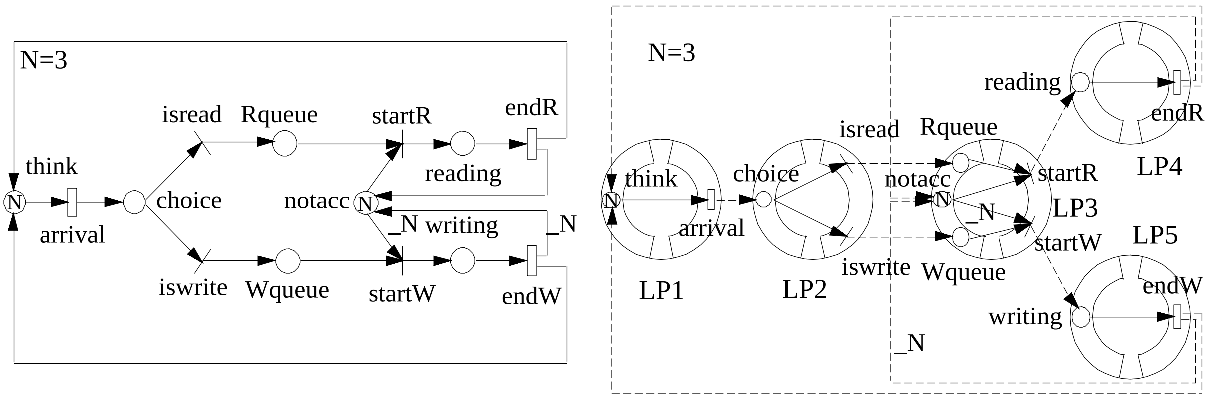


Figure 3: Reader/Writer Example: Minimum Region Partitioning.

arc-degree of connectivity of LP_i in figure 2 is 7, while a single channel is needed to connect LP_i to LP_j, thus inducing a channel-degree ≤ 5 for LP_i. Figure 3 gives the minimum region partitioning discussed in the previous chapter of an example GSPN for the reader-writer problem [Ajmo 91] (same figure).

2.4 The Simulation Engine

A general *simulation engine* (SE) implements the simulation of the occurrence of events in virtual time according to their causality, while collecting an event occurrence trace over the whole simulation period. Data structures for the TTPN representation, an event list (EVL) with entries $e_k = \langle t_i @FT \rangle$ (fire transition t_i at time FT), the virtual time (VT), and the event stack (ES) with entries of the form $\langle t_i, VT, M \rangle$ (t_i is the transition that has fired at virtual time VT, yielding a new marking M) have to be maintained.

An optimized discrete event SE [Balb 89] [Chio 93] exploits structural properties of the underlying TTPN in order to speed up simulation time: Let $t_i, t_j \in T$ be *causally connected* denoted by $(t_i\ CC\ t_j)$ (i.e. given $t_i \in E(M)$ and $t_j \notin E(M)$, then $M[t_i\rangle M'$ might cause that $t_j \in E(M')$) and $CC(t_i)$ the set of all transitions causally connected to t_i, then the firing of t_i has a potential enabling effect only on transitions in $SPE(t_i) = CC(t_i)$. Call $SPE(t_i)$ the *set of potential enablings* of t_i. Analogously a *set of potential disablings* $SPD(t_i)$ can be defined: Let $t_i, t_j \in T$ be *mutually exclusive* (denoted by $(t_i\ ME\ t_j)$), i.e. they cannot be simultaneously enabled in any reachable marking, and $ME(t_i)$ the set of all transitions mutually exclusive to t_i, then the firing of t_i has a potential disabling effect only on transitions in $SPD(t_i) = SC(t_i) - ME(t_i)$ ($SC(t_i)$ is the set of transitions in structural conflict to t_i). The gain of this TTPN structure exploitation is that after every transition firing only $SPE(t)$ and $SPD(t)$ have to be investigated rather than the whole T, which is significant for $\mid SPE(t_i) \mid$, $\mid SPD(t_i) \mid \ll \mid T \mid$.

The (optimized) SE behaves as follows. After an initialization and a preliminary scheduling of events caused by the inital marking the algorithm performs — if there are events to be simulated — the following steps until some end time is reached. First all simultaneously enabled transitions are generated and conflict is resolved identifying one transition per actual conflict set to be fired; the first scheduled transition is fired in that it is removed from EVL, the marking is changed accordingly, possible new (obsolete) transition instances are scheduled (descheduled) by insertion (deletion) of new (old) events into (from) EVL (investigating only SPE and SPD !); the occurrence of the event at its virtual simulated time and the new marking are logged into ES, and the VT is updated to the occurrence time of the event processed.

This (sequential) general SE is adapted to DDES strategies as described in the following sections.

2.5 The Conservative Strategy

The SE following the conservative approach allows only the processing of *safe* events, i.e. the firing of transitions up to a *local* virtual time LVT for which the LP has been guaranteed not to receive token messages with timestamp "in the past." The causality of events is preserved over all LPs by sending timestamped token messages of type $m = \langle w, D, TT \rangle$ (with $w > 0$) in non decreasing order, or at least a promise $m = \langle 0, D, TT \rangle$ (*null message*) not to send a new message time-stamped earlier than TT, and by processing the corresponding events in nondecreasing time stamp order.

One basic practical problem is the determination of when it is safe to process an event, since the degree to which LPs can *look ahead* and predict future events plays a critical role in the performance of the DDES. For conservative DDES so called "lookahead" coming directly from the TTPN structure can be exploited. Given some transition t_i located in the output border of some LP_k ($t_i \in T_k^{\triangleright}$). t_i is said to be *persistent* if there is no $t_j \in T_k$ such that if $t_i, t_j \in E(M)$, and $M[t_j\rangle M'$ causes that $t_i \notin E(M')$ for all reachable markings M. A sufficient condition for persistence of t_i is that $\forall t_j$, ${}^\bullet t_i \cap {}^\bullet t_j = \emptyset$. Call t_j the persistent predecessor of t_i if ${}^\bullet t_i = t_j^\bullet$ and $\forall k \neq j$, ${}^\bullet t_i \cap t_k^\bullet = \emptyset$;

let $T_k^{pers}(t_i)$ be the set of all persistent predecessors ahead of t_i, i.e. the transitive closure of the persistent predecessor relation. Then a lower bound for the degree of lookahead exposed by LP_k via t_i is $\sum_{j | t_j \in T_k^{pers}(t_i)} \tau_j$, i.e., the sum over all firing times of transitions in $T_k^{pers}(t_i)$.

As an example consider the upper net part in LP_i in figure 2. Until the firing of the immediate transition connected to the input border a null message with a time increment of the firing times of the two succeeding timed transitions can be sent to LP_j, promising that it is safe to process events up to the token time of the null message in LP_j (given that safety of processing is not restricted by other causalities).

Upon firing of the immediate transition connected to the input border the token message corresponding to the firing of the timed transition on the output border of LP_i may be sent right away, since the transition will certainly be enabled in the future and its firing won't be preempted: the timestamp of the token message must be in this case LVT increment of the firing times of the two succeeding timed transitions (thus ahead of the current LVT). [1]

Definition 2.3 *Let $t_i \in T_k^{\triangleright}$ and $t_j \in T_k$ be persistent $(({}^\bullet t_i)^\bullet = t_i, ({}^\bullet t_j)^\bullet = t_j)$. t_i and t_j are said to be in the same persistence chain denoted by the set $(t_j \rightsquigarrow t_i)$, iff there exists a sequence $S = \{t_r, t_s \ldots t_t\}$ of persistent predecessors $(t_r, t_s \ldots t_t \in T_k)$ such that $t_j \in E(M) \Rightarrow t_i \in E(M')$ $M[S\rangle M'$.*

Since every $t \in S$ is persistent, we can state the following:

Corollary 2.1 *Given that $t_j \in T_k \rightsquigarrow_i \in T_k^{\triangleright}$ and M with $t_j \in E(M)$ at time τ, then M' with $t_i \in E(M')$ is reached at time $\tau + \sum_{t \in S} \tau(t)$ at the earliest.*

[1] Notice that the utilization of transition firing times ahead of the LVT is trivial in case of deterministic timings; it is less trivial to implement efficiently in case of random timing where in general requires a precomputation of the next instances of random variables associated with the random process that defines the time of each transition. Such a precomputation is practically feasible in the case of bounded TTPNs where a maximum value for the enabling degree of each persistent predecessor transition can be statically computed before running the DDES processes.

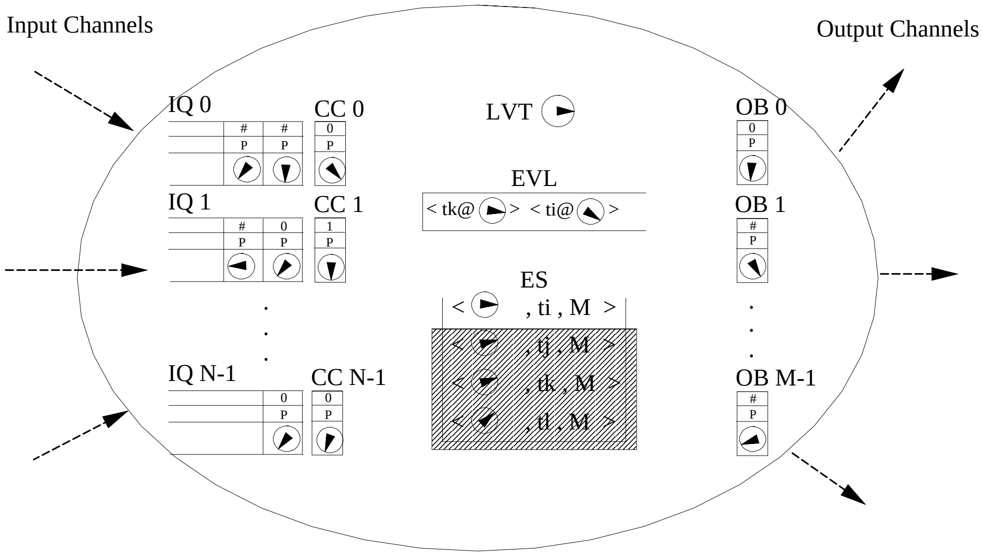


Figure 4: Logical Process for the Conservative Strategy

For two transitions $t_j \in T_k \rightsquigarrow t_i \in T_k^{\triangleright}$ in the same persistence chain we can define the amount of lookahead of t_j onto t_i by $la((t_j \rightsquigarrow t_i)) = \sum_{t\in(t_j\rightsquigarrow t_i)} \tau(t)$ (with the particular case $la((t_j \rightsquigarrow t_i)) = 0$ if $(t_j \rightsquigarrow t_i) = \emptyset$). The value of la can be established for all pairs of transitions in the TTPN region (one in the output border, the other not in the output border) by a static preanalysis of the region structure. The impact on the improvement of simulation performance is, that upon firing of any transition within some TTPNR$_k$ the timestamp of output messages caused by transitions in the output border which are in the same persistence chain can be increased (and thus improved) by la, thus relaxing the synchronization constraints on the adjacent LPs.

Assuming that the communication requirements of $GLP = (LP, CH)$ are supported by the multiprocessor hardware (i.e. there exist communication media for all $ch_{k,i} = (LP_k, LP_i)$), then we derive the following conservative simulation engine SE^{co} to be applied in every $LP_i \in LP$. Two types of messages are necessary to implement the communication interface: *token messages* $m = \langle w_f, D, TT\rangle$ carrying a specific number of tokens (w_f or $\#$) to some destination place D (which uniquely defines the destination LP), and *null messages* $m = \langle 0, D, TT\rangle$. SE^{co} holds the net data of TTPNR$_k$ and simulates the local behavior of the region by holding transitions to be fired in a local EVL, recording event occurrences in a local ES and incrementing a local time LVT. For every input channel an input queue collects incoming messages, the first element of which (that defines the *channel clock*, CC) is used for synchronization. Output buffers OB keep messages to be sent to other LPs, one per output channel. The behavior of SE^{co} is to process the first event of EVL if there is no token message in one of the CC_is with smaller timestamp (*process first event*), or to process the token message with the minimum token time (*process first message*). Processing the first event (i.e., firing transition t) is similar to the general SE (change marking, schedule/deschedule events, increment LVT), but also invokes the sending of messages: If $t \in T_k^{\triangleright}$ then a message $\langle \#, D, LVT\rangle$ is generated and deposited in the corresponding OB; if $t \notin T_k^{\triangleright}$ then a null message $\langle 0, D, LVT + la((t \rightsquigarrow t_i))\rangle$ is deposited for every $t_i \in T_k^{\triangleright}$ in the corresponding OBs – thus giving maximum lookahead to all the following LPs. After processing the first event the contents of all OBs is transmitted, except for null messages with token-times that have already been distributed in a previous step (to reduce the number of null messages).

```
program SE^co(TTPN_k)
S1      LVT := 0; EVL := {}; M := M_0;
S2      for all CC_i do (CC_i := ⟨0, dummyP, 0⟩) od;
S3      for all t_i ∈ E(M_0) do schedule(⟨t_i@τ_i⟩) od;
S4      while LVT ≤ endtime do
S4.1        for all t_i ∈ E(M) with τ_i = 0 do
S4.1.1          t := selected_for_firing(E(M));
S4.1.2          process(t);
S4.1.3          delete(event(t), EVL) od;
S4.2        m_min := CC | min_j(tokentime(CC_j));
S4.3        if tokentime(CC_j)) < tokentime(first(EVL)) and tokenmessage(m_min)
                then do
                    /* process first message */
                    M(place(m_min)) := M(place(m_min)) + tokencount(m_min)
                    for all t_i ∈ (place(m_min))• do
                        if t_i ∈ E(M)
                          then schedule(t_i@LVT + τ(t_i))
                          else  deschedule(t_i) od,
                    tokencount(m_min) := 0 od;
S4.4        if not empty (EVL) and tokentime(first(EVL)) < tokentime(m_min)
              then do
                  /* process first event */
                  t := select_for_firing(E(M));
                  process(t);
                  delete(event(t),EVL) od;
S4.5        for all OB_i do sendout(OB_i);
        od while
```

```
program process(t)
S1      LVT := LVT +τ(t);
S2      change_marking(M, t);
S3      for all t_i ∈ SPE(t) do
                if t_i ∈ E(M) then schedule(⟨t_i@LVT + τ(t_i)⟩) od;
S4      for all t_i ∈ SPD(t) do
                if t_i ∉ E(M) then deschedule(t_i) od;
S5      if t ∈ T_k^▷
          then do
               OB(t) := ⟨w(t, outplace(t)), outplace(t), LVT⟩
               for all t_i ∈ T_k^▷ and t_i <> t do
                      OB(t_i) := ⟨0, outplace(t_i), LVT⟩ od od
          else do
               for all t_i ∈ T_k^▷ do
                      if t ⇝ t_i
                        then OB(t_i) := ⟨0, outplace(t_i), LVT + la((t ⇝ t_i))⟩
                        else OB(t_i) := ⟨0, outplace(t_i), LVT⟩ od od;
S6      push(ES, ⟨LVT, t, M⟩)
```

Figure 5: Conservative Simulation Engine for TTPN_k.

The processing of the first message invokes removing the message with minimum timestamp over all CC_is from CC_i (the head of IQ_i), while leaving a "null message copy" (change # to 0 in the message head) of it in CC_i if it has been the last message in IQ_i. M and LVT are changed accordingly, also scheduling/descheduling of events might become necessary. The SE^{co} is given in figure 5, the data structure of an LP with conservative SE is depicted in figure 4. The performance of SE^{co} will be seen in comparison studies in the next section.

The algorithm blocks as soon as the minimum timestamp of messages in the CCs is not larger than the occurrence time of the first event in EVL, and avoids deadlock as long as one does not have any cycles in which the collective time stamp increment of messages traversing that cycle *could* be zero [Misr 86]. A sufficient condition not to have deadlock is thus $\forall\ t_i \in T_k^{\triangleright},\ \tau(t_i) > 0$.

2.6 Time Warp Strategies

In order to simulate a $TTPNR_k$ according to the Time Warp (optimistic) strategy [Jeff 85c] one *input queue* IQ and one *output queue* OQ with time sequenced message entries are maintained. Either *positive (token) messages* ($m = \langle w_f, D, TT, \text{`}+\text{'}\rangle$) or *negative (annihilation) messages* ($m = \langle w_f, D, TT, \text{`}-\text{'}\rangle$) are received from other LPs out of $GLP = (LP, CH)$ not necessarily in time stamp order, indicating either the transfer of tokens or the request to annihilate a previously received positive message. Messages are assumed to be buffered in some input buffer IB upon their arrival, to be taken over into IQ eventually. Messages generated during a simulation step are intermediately held in an *output buffer* OB to be sent upon completion of the step itself, all at once. The data structure of an LP with an optimistic SE (SE^{opt}) is depicted in figure 6; a simulation engine is shown in figure 7.

SE^{opt} behaves as follows: first all $t_i \in T_k$ enabled in M_0 are scheduled. Messages received are processed according to their timestamp and sign; messages with timestamp in the local future (*tokentime*$(m) >$ LVT) are inserted into IQ (if the sign of m is '+' then it is inserted in timestamp order, a message with sign(m) = '$-$' annihilates its positive counterpart in IQ. *Straggler messages* (*tokentime*$(m) <$ LVT) forces the LP to *roll back* in local simulated time by restoring the most recent *valid* state. Processing the first event (as in SE^{co}) simulates the firing of a transition and the generation of output messages, while processing the first message changes M and possibly schedules/deschedules new/old events. Whenever an event is processed the *event stack* ES additionally records all state variables such that a past state can be reconstructed on occasion. The algorithm requires the knowledge of the global virtual time (GVT), a lower bound for the timestamp of any unprocessed message in $GLP = (LP, CH)$, in order to reduce bookkeeping efforts for past states (ES).

Rollback can be applied in two different ways. An optimistic simulation engine with *aggressive cancellation* (SE^{ac}, see figure 8) first inserts the straggler message into IQ, and sets LVT to *toktime*(m). Then the state at time LVT is restored, which is the marking at time of the already processed event closest but not exceeding LVT in SQ. All messages in OQ with token time larger than the rolled back LVT are annihilated by removing them from OQ and sending corresponding antimessages. Finally, all records prematurely pushed in the STST are popped out (line *S5*). A *lazy cancellation* optimistic simulation engine (SE^{lc}) would keep the negative messages generated in an intermediate list (*S4.1:* insert(IL, $\langle tokencount(m), outplace(m), tokentime(m), \text{`}-\text{'}\rangle$)), and move that (negative) m to OB only in the case that resimulation has increased LVT over *tokentime*(m). In the case that reevaluation yields exactly the same positive message as already sent before, the new positive message is not

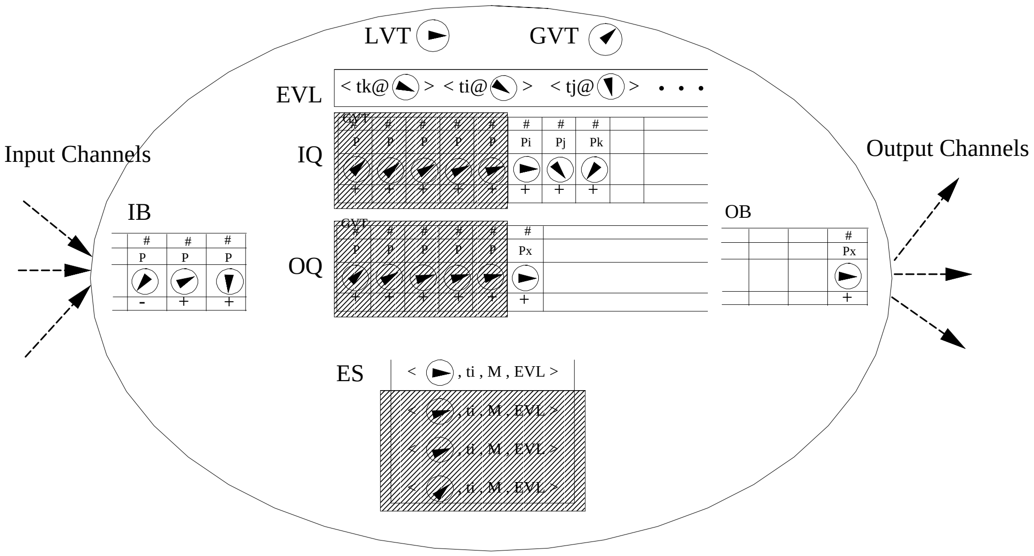


Figure 6: Logical Process for Optimistic Strategies

---

program $SE^{opt}(\mathrm{TTPN}_k)$

S1 GVT:=0; LVT := 0; EVL := {}; M := M_0; stop := false;
S2 **for all** $t_i \in E(M_0)$ **do** schedule($\langle t_i @ \tau_i \rangle$) **od**
S3 **while** not stop **do**
S3.1 **for all** $m \in$ IB **do**
S3.1.1 **if** $tokentime(m) < LVT$
S3.1.1.1 **then** rollback($tokentime(m)$, m) **else** insert(IQ, m) **od**
S3.2 **if** EVL = {} $\wedge$ IQ = {} **then** goto *S3.1*
S3.3 **if** (GVT $\leq$ endtime) $\wedge$ (not empty(EVL)) **do**
S3.3.1 **if** $tokentime$(first(EVL)) < $tokentime$(first(IQ))
S3.3.1.1 **then** process(first(EVL)) **else** process(first(IQ)) **od**
S3.4 **for all** $m \in$ OB **do** send(m) **od**
S3.5 **if** (GVT $\geq$ endtime) **then** stop := true;
S3.6 **od while**

Figure 7: Aggressive Cancellation Simulation Engine for TTPN_k.

---

resent, but used to compensate the corresponding negative message from IL — thus annihilating the (obsolete) causality correction localy, and preventing from unnecessary message transfers as well as possibly new rollbacks in other LPs.

3 Performance Influences

Due to the variety of degrees of freedom in composing a DDES for TTPNs as offered by the framework developed above, the question for optimizing the performance of a distributed simulation on a physical multiprocessor naturally arises. We have implemented the SEs and communication interfaces for a 30 node Sequent Balance, a 16 node Intel iPSC/860 and a 16 node T800 based Transputer multiprocessor. Case studies showed that there are influences onto the performance of the distributed simulation which are inherent to a proper combination of the *partitioning in regions* (TTPNR_ks), the *communication*

```
procedure rollback(time, m)
         /* aggressive cancellation */
S1       LVT := time
S2       insert(IQ, m)
S3       restore-state(LVT)
S4       for all m ∈ OQ with tokentime(m) ≥ LVT do
S4.1           insert(OB, ⟨tokencount(m), outplace(m), tokentime(m), '−'⟩);
S4.2           delete(OQ, m) od
S5       pop(ES, LVT)
```

Figure 8: *Aggressive Cancellation* Rollback Mechanism.

---

interface (I_ks) and the *simulation engine*, but also caused by the physical hardware as such. In the following we will evaluate the main observations and use the empirical results for an improved distributed simulator for TTPNs.

3.1 Partitioning

We already proposed a partitioning such that several transitions together with all their input places are simulated by a single logical process. Subnets are constructed from the topologic description of the TTPN so that conflict resolution always occurs internally to a logical process. Depending on the particular multiprocessor architecture on which the DDES is run, such minimal region partitioning may however turn out to be too fine grain to efficiently adapt to the interprocessor communication overhead. One should thus be willing to partition the DDES in a lower number of LPs in order to reduce the communication overhead and attain better performance. In practice a trade-off must be sought for different target architectures based on empirical cases in order to achieve speedup over sequential simulation.

Figure 9 compares the minimum region partitioning with a SE^{lc} of the reader writer net in figure 3 in a balanced parametrization: $\lambda_{arrival} = 1.0$, $\lambda_{endR} = 2.0$, $\lambda_{endW} = 0.5$, $prob(isread) := 0.8$, $prob(iswrite) := 0.2$ and $N = 4$ processes. The results show that the Sequent Balance takes physically more communication time, although the work profile is the same as for the iPSC. Actually the iPSC uses a faster processor which has also impact onto the simulation behavior: the higher processor speed allows faster event simulation wich is the cause for invoking also more rollbacks (see e.g. the absolute number of rollback steps in LP2 on the iPSC in figure 10.

From the communication overheads encountered for the minimum region partitioning it is obvious to see that the communication/computation ratio has to be increased to use this kind of multiprocessor hardware more efficiently. In order for that additional aggregation of conflict sets into larger logical simulation processes may increase the balancing of the distributed simulation without decreasing its inherent parallelism if some conditions are verified on the TTPN model structure (grain packing). For example the transitions *endR* and *endW* are never simultaneously enabled, thus LP_4 and LP_5 in figure 3 can be aggregated to a single LP without loss of potential model parallelism.

The following rules can be applied to pack grains starting from minimum $TTPNR_k$s:

Rule 1 mutually exclusive (ME) transitions go into one LP since they bear no potential parallelism Denote two mutually exclusive transitions $t_i, t_j \in T$ by $(t_i\ ME\ t_j)$, then a sufficient condition

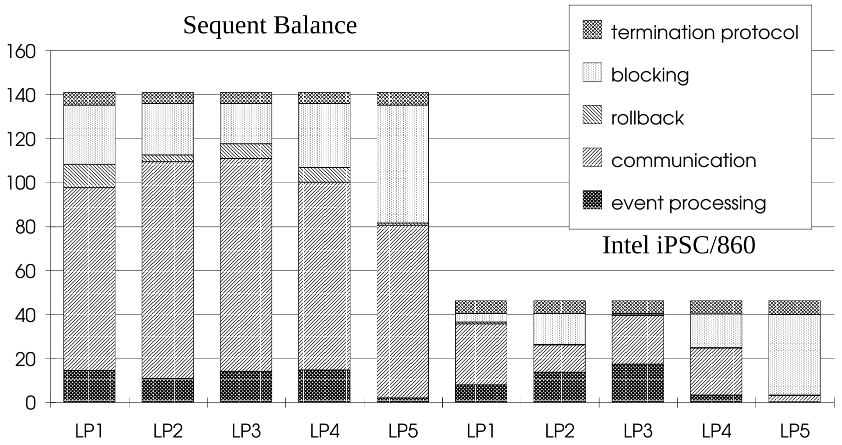


Figure 9: Execution Profiles for Minimum Region Partitioning

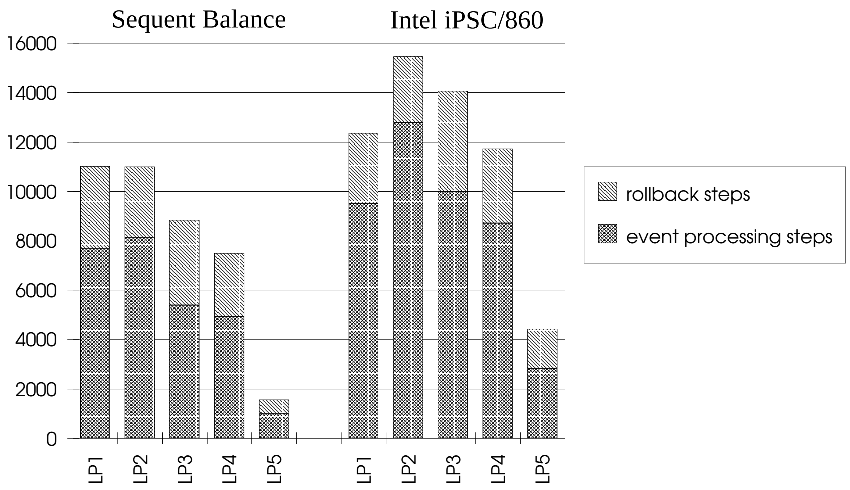


Figure 10: Simulation Steps in Minimum Region Partitioning

for $(t_i\ ME\ t_j)$ is that the number of tokens in a P-invariant out of which t_i and t_j share places prohibits a simultaneous enabling. Another sufficient conditions for $(t_i\ ME\ t_j)$ is that $\exists t_k\ :\ \pi_k > \pi_j$ such that $\forall p \in {}^\bullet t_k\ p \in {}^\bullet t_i \cup {}^\bullet t_j \wedge w(t_k, p) \leq \max(w(t_i, p), w(t_j, p))$.)

Rule 2 endogenous simulation speed is balanced to prevent from rollbacks, i.e. the probability of receiving straggler messages is reduced by balanced virtual time increases in all LPs

Rule 3 LPs with high message traffic intensity are clamped to save message transfer costs

Rule 4 persistent net parts and free choice conflicts are always placed to the output border to allow sending out messages ahead the LVT (lookahead) without possibility of rollback, (i.e. sending ahead messages that will be inevitably generated by future events — unless local rollback occurs)

Rule 5 transitions having only a single input place can also be connected to the input border, since the enabling test can be avoided for these transitions (firing can be scheduled immediately upon receipt of the positive token message without additional overhead)

One may realize by looking at the connectivity of LP_3 (figure 3) that this logical process needs to treat 3 events out of 4 per access cycle (the marking of the input border of LP_3 is affected by

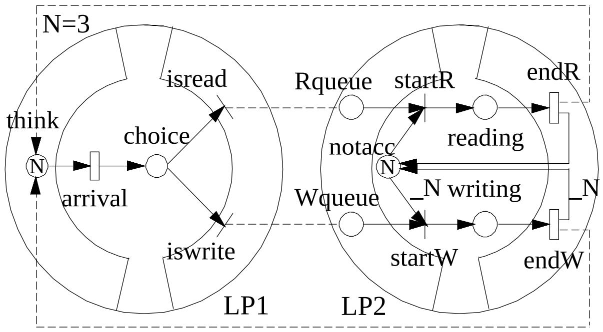


Figure 11: Reader/Writer Example: Optimum Partitioning.

the firing of transitions "isread,", "iswrite," "startR," "startW," "endR," and "endW," and is not affected only by the firing of "arrival"). This structural characteristics limits the maximum speedup of a distributed simulation of the particular model to the value 4/3 (since each marking update accounts for one simulation step of LP_3). As LP_4 and LP_5 are persistent supporting the exploitation of lookahead both of them will be aggregated to LP_3 to form the output border of a new LP_{3+4+5} (Rule 4). (Moreover LP_3 and LP_5 are mutually exclusive (Rule 1), and the aggregation reduces the external communications (*endW*, *notacc*) and (*endR*, *notacc*) (Rule 3).)

Since we observe an overlap of real simulation work only among LP_1, LP_2 and LP_3 in the minimum region decomposition (LP_1 simulates the arrival of some customer, while LP_2 simulates the choice of another one and at the same time LP_3 simulates the start of a read access), we can use the following arguments to merge LP_1 and LP_2 to a new LP_{1+2}: LP_{1+2} contains an TTPNR with single input transition connected to the input border and free-choice conflicting transitions in the output border, so that 1) the enabling test for transition *arrival* can be avoided on receipt of a token message (Rule 5), and 2) the output message can be sent ahead of simulation due to the free choice conflict in the output border (Rule 4). (It is natural to see that we preserve (and exploit) the maximum speedup offered by the bottleneck LP_3 since LP_{1+2} can send outmessages upon receipt of inmessages ahead of simulation.) We finally end up with a two LPs (LP_{1+2}, LP_{3+4+5}) partitioning which is optimum in the particular model.

The figures 11, 12 show the optimum partitioning and the performance (total simulation time) of its Transputer implementation (Part. 1 refers to the partitioning (LP_1, $LP_{2+3+4+5}$), Part. 2 to (LP_{1+2}, LP_{3+4+5}) and Part. 3 to (LP_{1+2+3}, LP_{4+5}). Figure 13 compares the Balance and iPSC performance for the minimum region partitioning, a decomposition into 3 regions (LP_{1+2}, LP_3, LP_{4+5}), the optimum partitioning and the simulation engine simulating the whole TTPN (sequential simulation).

3.2 Communication

Since communication latency is rather high compared to the raw processing power for the multiprocessor systems under investigation, communication is the dominating performance influence factor (see e.g. figure 9) for all simulation engines. The most promising tuning of a DDES of TTPNs is by making arc-degree and channel-degree reduction the main principles of the partitioning process. Additionally to the grain packing rules we can state:

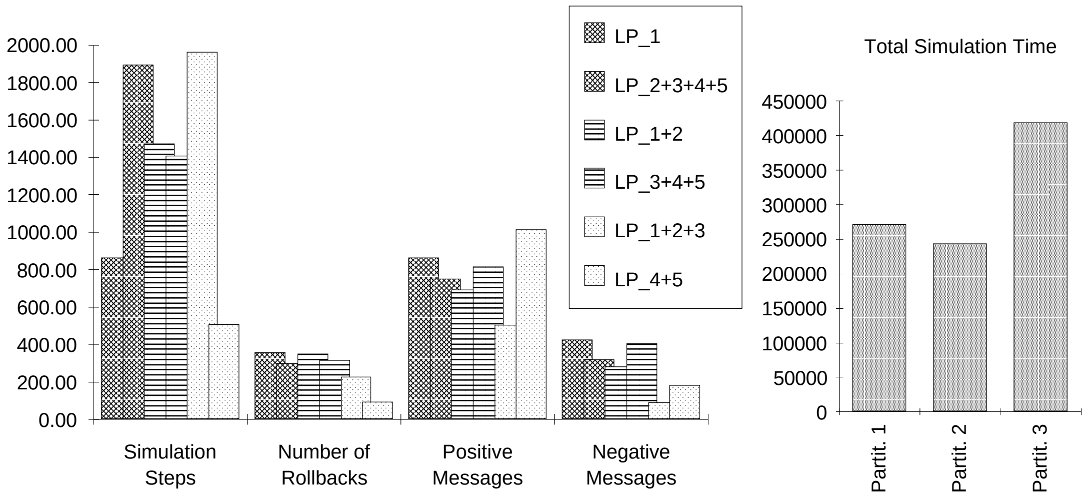


Figure 12: Reader/Writer Example: Optimum Partitioning Performance (Transputer)

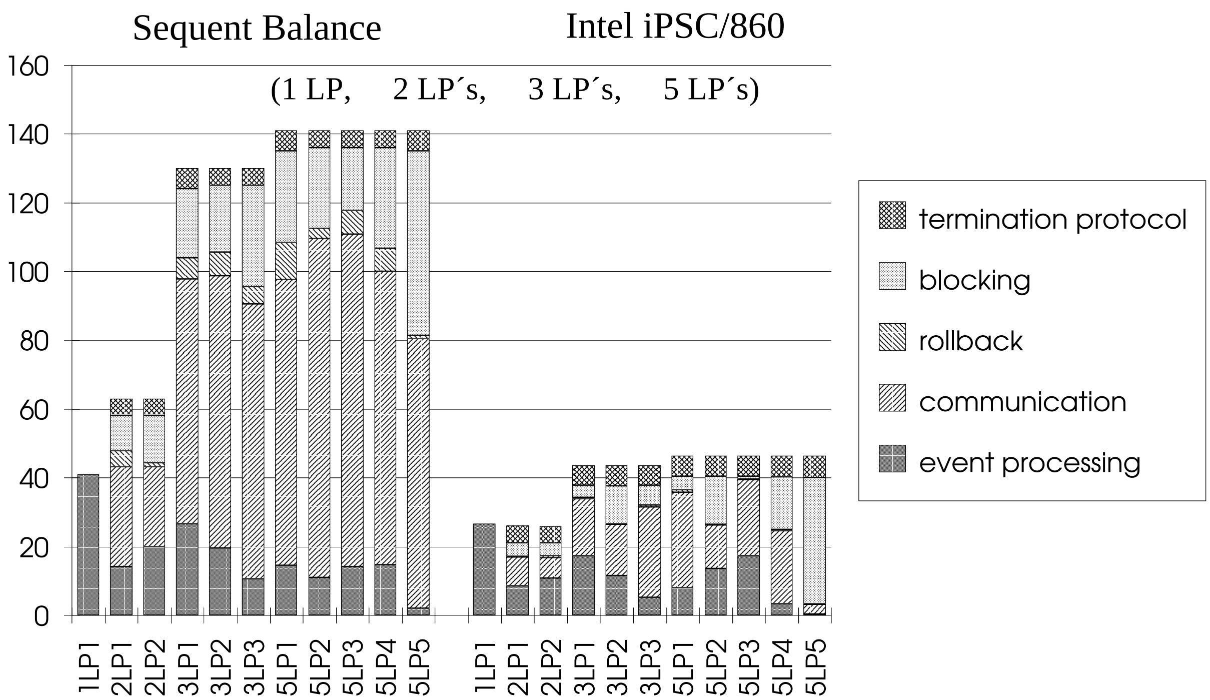


Figure 13: Empirical Comparison of Partitionings on Sequent Balance and Intel iPSC/860

Rule 7 Among all the possible region partitionings of TTPN into a graph of logical processes $GLP = (LP, CH)$ employing a constant number of LPs ($| LP |$) where $LP_i \in LP$ simulates TTPNR_i, choose the one with minimum average channel degree $(\sum_i CD(\mathrm{TTPNR}_i)/ \mid LP \mid \; \rightarrow \; min$. Should there be more than one GLP with equivalent (minimum) average-channel degree, then use the one with the minimum average arc-degree $AD(\mathrm{TTPNR}_i)$ out of these.

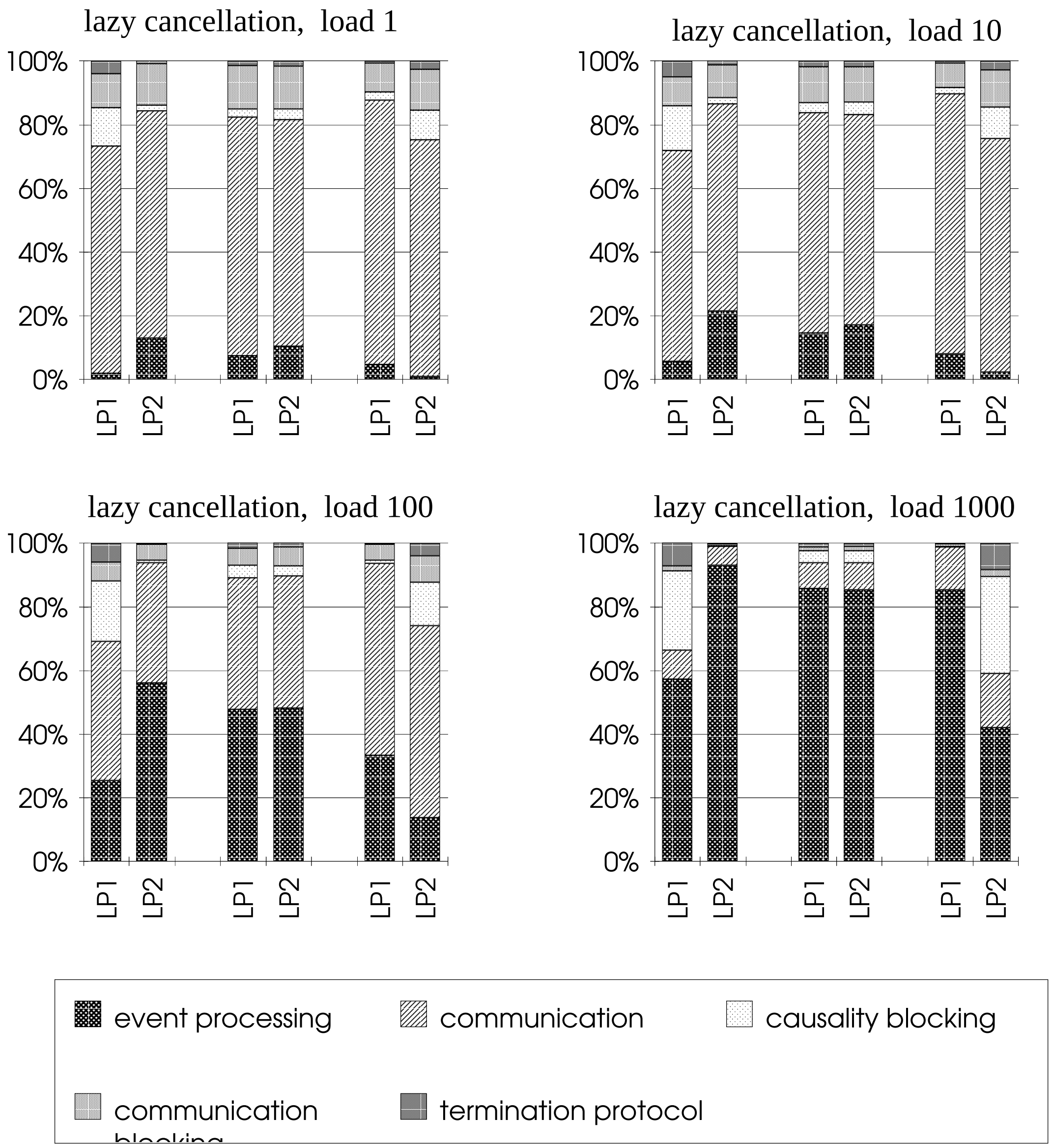


Figure 14: The Impact of LP-Load onto the Execution Profile (Transputer)

3.3 Load

With respect to the empirical observations concerning the communication latency the grain packing rules have also to be extended in order to achieve *actual* speedup when simulating the TTPN regions in parallel. Naturally this can happen as soon as the efforts for real (local) simulation work exceed the communication efforts:

Rule 8 Cluster TTPNR_ks such that the local simulation work in terms of physical processor cycles exceeds a certain (hardware specific) computation/communication threshold in order to observe real speedup.

Figure 14 shows execution profiles of lazy cancellation simulations of the balanced reader/writer GSPN in three different partitionings (Part. 1 = (LP_1, $LP_{2+3+4+5}$), Part. 2 = (LP_{1+2}, LP_{3+4+5})

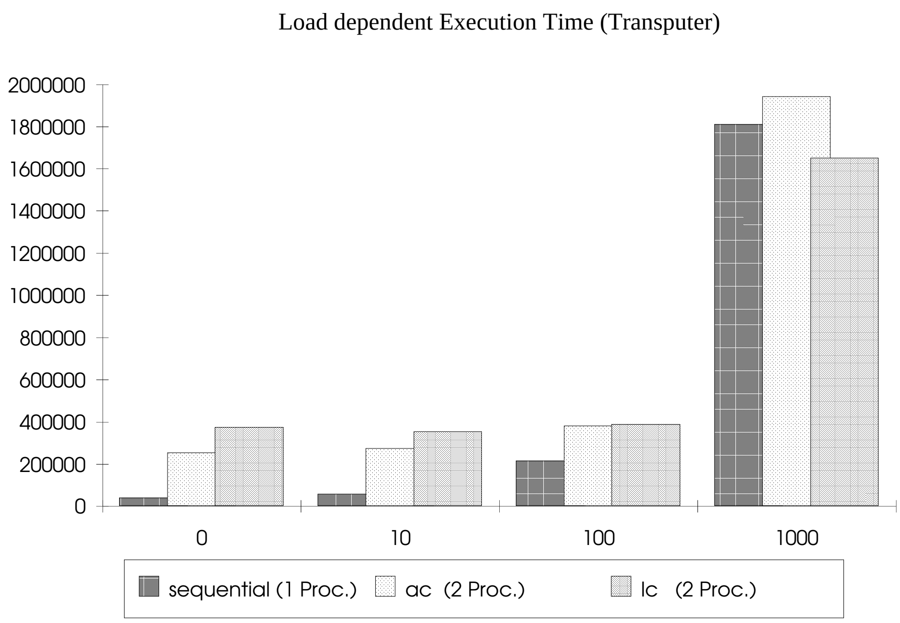


Figure 15: The Impact of LP-Load and Sim. Strategy onto the Execution Time (Transputer)

and Part. 3 = (LP_{1+2+3}, LP_{4+5}) for increasing (hypothetical) loads on the two processors. The executions have been generated by inserting additional transitions in the various LPs in order to increase the amount of local simulation work, and show how the computation/communication ration can be increased by clamping transitions with 'local behavior' to LPs. Figure 15 shows the threshold of local work to be reached in order to outperform the sequential (single processor) implementation of the simulation.

3.4 The Strategy

Obviously the simulation strategy has a strong influence on the performance of a DDES. Since SE^{co} strictly adheres the local causality constraint by processing events only in nondecreasing timestamp order, the SE^{co} cannot fully exploit the parallelism available in the simulation application. In the case where one transition firing *might* affect (directly or indirectly) the firing of another transition SE^{co} must execute the firings sequentially – hence it forces sequential execution even if it is not necessary. So in the case where causality affects among transitions is possible but seldom, SE^{co} is overly pessimistic. SE^{ac} and SE^{lc} gain from a proper partitioning and the placement of net parts in the input- or outputborder of the LP (as described), but suffer from tremendous memory requirements and memory access requirements. Although empirical observations give raise for best speedup attainable by the use of SE^{lc} and large $TTPNR_k$s with minimum average channel degree, a general rule cannot be stated on the SE to be applied.

4 Optimizing the SE

With SE^{co} we have seen how to exploit the $TTPNR_k$ structure to improve the standard simulation engine by introducing persistence chains for transitions in the output border. Based on particular model structures the optimistic simulation engines can also be optimized in different respects. One of these optimizations have already been undertaken with the local annihilation of negative messages in SE^{lc}. Others are sketched as follows:

4.1 Message Send Ahead

Output messages generally sent at the end of one simulation step (*S3.4* in figure 7) in SE^{lc} and SE^{ac} can be sent for every $t_i \in T_k^{\triangleright}$ and $t_j \in T_k$ with $t_i \rightsquigarrow t_j$ upon scheduling of t_i, thus reducing substantially the latency of the information propagation. The potential lookahead $la((t_i \rightsquigarrow t_j))$ can hence also help to improve optimistic SEs. Empirical observations show that the gain from SE^{lc} with message send ahead is substantial for the reader/writer net: In the LP$_{1+2}$, LP$_{3+4+5}$ partitioning messages for "endR" and "endW" can be sent one simulation step ahead upon scheduling of "startR" and "startW" giving LP$_{1+2}$ the chance to schedule (and fire) new arrivals at the same time (in parallel) the accesses are simulated by LP$_{3+4+5}$.

4.2 Elimination of Enabling Test

Transitions with a single input place in the input border allow the firing of the transition in the same simulation step when the enabling message is received (if there were no previously scheduled events), thus also contributing to a speedup of the corresponding LP. Empirically we have observed minor improvement by the eliminated enabling test since the maintenance of IQ, OQ and ES is dominating the execution time within one simulation iteration (*S3*).

4.3 Lazy Rollback

In cases where a straggler positive message changes the marking but does not cause the enabling of any new event in the past (but e.g. in the future), a simplified rollback mechanism is sufficient to recover from the causality error. This is best explained by a ceteris paribus analysis of a simple example. The two LPs in figure 16) simulate (approx.) at the same load, whereas LP$_2$ ($\lambda(T4) = 0.5$) increments LVT twice as fast as LP$_1$ ($\lambda(T1) = 0.25$) does. After every 6^{th} step in LP$_1$ $T2$ generates a straggler message for LP$_2$ (i.e. time stamped in the past of LP$_2$, and with effect possibly ($\lambda(T3) = 0.1$) in the future of LP$_2$) which potentially does not violate any causality in LP$_2$. In this case no rollback would be invoked. Should the effect of the message received however be in the past of LP$_2$, then only an appropriate insertion of the firing of $T3$ is made on ES, and the top of ES (entries with time stamps inbetween the occurrence time of $T3$ and LVT) is copied considering a potential change in the marking (not necessary in the example). The effect of lazy rollback is also shown in figure 16 (Transputer implementation of SE^{lc} with lazy rollback).

5 Conclusions

In this paper we have described the implementation of various adaptations of classical DDES strategies to the simulation of TTPN models. Several prototypes have been developed and run on three different distributed architectures, allowing the collection of many empirical data from the measurement of the performance of such prototypes. These results have been used to validate the different variations of the techniques on some case study in order to identify problems and potentialities of the approach.

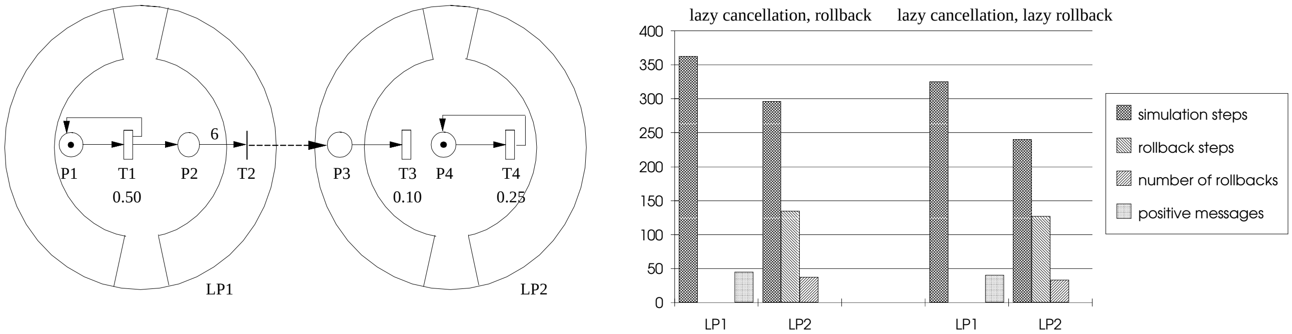


Figure 16: Performance improvement by Lazy Rollback

The main performance problem that we found (not surprisingly) was related to the interprocessor communication latency inherent to distributed architectures. The conclusion that can be drawn from our preliminary results is that DDES has no hope to attain real speedup over sequential simulation unless the intrinsic properties of parallelism and causality of the simulated model are properly identified and exploited to optimize the LPs. Moreover, only large TTPN descriptions have a chance to produce a sufficiently large number of LPs each one of sufficiently large grain so as to overcome the communication overhead. Experimental results show that increasing the number of LPs by fine grain partitioning is a naive and ineffective way of identifying massive "potential parallelism." Each model may be characterized by its inherent parallelism independently of the number of places and transitions of the TTPN description, and it is this inherent parallelism that we should try and capture in order to achieve speedup over sequential simulation.

In this sense, the use of a TTPN formalism may provide a substantial contribution to the implementation of efficient, general purpose DDES engines. Indeed most of the relevant characteristics that have to be taken into account to produce efficient LPs are determined by the Petri net structure. Appropriate phases of structural analysis may be implemented in order to capture such relevant characteristics automatically from the model structure, thus releaving the modeler from the burden of identifying them by himself. Proper software tools exploiting this idea may thus bring efficient utilization of DDES techniques to users not having expert knowledge on this research field.

The work presented in this paper should be considered only as a first step in the direction of exploiting Petri net structural analysis for the efficient implementation of DDES techniques. We already identified some net patterns that yield particularly efficient simulation strategies. We believe however that several other net dependent optimizations may be studied and implemented in order to obtain practical advantages from the application of DDES techniques to real models.

References

[Ajmo 84] M. Ajmone Marsan, G. Conte, and G. Balbo. “A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems”. *ACM Transactions on Computer Systems*, Vol. 2, No. 2, pp. 93 – 122, May 1984.

[Ajmo 87] M. Ajmone Marsan and G. Chiola. “On Petri Nets with Deterministic and Exponentially Distributed Firing Times”. In: G. Rozenberg, Ed., *Advances in Petri Nets 1987, Lecture Notes in Computer Science Vol. 266*, pp. 132 – 145, Springer Verlag, 1987.

[Ajmo 91] M. Ajmone Marsan, G. Balbo, G. Chiola, G. Conte, S. Donatelli, and G. Franceschinis. “An Introduction to Generalized Stochastic Petri Nets”. *Microelectronics and Reliability*, 1991.

[Amma 91] H. H. Ammar and S. Deng. “Time Warp Simulation of Stochastic Petri Nets”. *submitted for publication*, 1991.

[Balb 89] G. Balbo and G. Chiola. “Stochastic Petri Net Simulation”. In: *Proc. 1989 Winter Simulation Conference*, Washington D.C., December 1989.

[Chan 79] K. M. Chandy and J. Misra. “Distributed Simulation: A Case Study in Design and Verfification of Distributed Programs”. *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 5, pp. 440–452, Sep. 1979.

[Chan 81] K. M. Chandy and J. Misra. “Asynchronous Distributed Simulation via a Sequence of Parallel Computations”. *Communications ACM*, Vol. 24, No. 11, pp. 198–206, Apr. 1981.

[Chio 91] G. Chiola. “*GreatSPN* 1.5 Software Architecture”. In: *Proc. 5^{th} Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, Feb. 1991.

[Chio 93] G. Chiola and A. Ferscha. “Exploiting Timed Petri Net Properties for Distributed Simulation Partitioning”. In: *Proceedings of the 26^{th} Hawaii Int. Conf. on Systems Sciences*, pp. 194 – 203, IEEE Computer Society Press, 1993.

[Ciar 89] G. Ciardo, J. Muppala, and K. S. Trivedi. “SPNP: Stochastic Petri Net Package”. In: *Proc. of the Third International Workshop on Petri Nets and Performance Models, December 11-13, 1989, Kyoto, Japan.*, p. , IEEE Computer Society Press, 1989.

[De V 90] R. C. De Vries. “Reducing Null Messages in Misra’s Distributed Discrete Event Simulation”. *IEEE Transactions on Software Engineering*, Vol. 16, No. 1, pp. 82–91, January 1990.

[Duga 84] J. B. Dugan, K. S. Trivedi, R. M. Geist, and V. F. Nicola. “Extended Stochastic Petri Nets: Applications and Analysis”. In: *Proc. of the 10th Int. Symp. on Computer Performance (Performance 84), Paris, France, Dec 19-21, 1984*, pp. 507 – 519, 1984.

[Feld 92] F. Feldbrugge. “Special Volume: Petri Net Tools Overview 92”. *Petri Net Newsletter*, No. 41, pp. 2 – 42, 1992.

[Ferr 91] P. Ferrara, A. Ferscha, I. Graf, and W. M"ullner. "Distributed Timed Petri Net Simulation". Tech. Rep. Research Report, Dept. of Statistics and Computer Science, Univ. of Vienna, Austria, 1991.

[Fers 90] A. Ferscha. "Modelling Mappings of Parallel Computations onto Parallel Architectures with the PRM-Net Model". In: C. Girault and M. Cosnard, Eds., *Proc. of the IFIP WG 10.3 Working Conf. on Decentralized Systems*, pp. 349 – 362, North Holland, 1990.

[Fuji 88a] R. M. Fujimoto. "Lookahead in Parallel Discrete Event Simulation.". In: F. A. Briggs, Ed., *Proceedings of the Int. Conf. on Parallel Processing*, pp. 31–41, IEEE Computer Society Press, 1988.

[Fuji 88b] R. M. Fujimoto. "Performance Measurement of Distributed Simulation Strategies.". In: *Proceedings of the Distributed Simulation Conference*, pp. 14–20, 1988.

[Fuji 90] R. M. Fujimoto. "Parallel Discrete Event Simulation". *Communications of the ACM*, Vol. 33, No. 10, pp. 30–53, October 1990.

[Gafn 88] A. Gafni. "Rollback Mechanisms for Optimistic Distributed Simulation Systems". In: *Proc. of the SCS Multiconference on Distributed Simulation 19*, pp. 61–67, 1988.

[Holl 87] M. A. Hollyday and M. K. Vernon. "A Generalized Timed Petri Net Model for Performance Analysis". *IEEE Transactions on Software Engineering*, Vol. 13, No. 12, pp. 1297 – 1987, Dec. 1987.

[Jeff 85a] D. A. Jefferson. "Virtual Time". *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, pp. 404–425, July 1985.

[Jeff 85b] D. R. Jefferson and H. Sowizral. "Fast Concurrent Simulation Using the Time Warp Mechanism.". In: *Proceedings of the Conf. on Distibuted Simulation*, pp. 63 – 69, 1985.

[Jeff 85c] D. Jefferson and H. Sowizral. "Fast Concurrent Simulation Using the Time Warp Mechanism". In: P. Reynolds, Ed., *Distributed Simulation 1985*, pp. 63–69, SCS-The Society for Computer Simulation, Simulation Councils, Inc., La Jolla, California, 1985.

[Kaud 87] F. J. Kaudel. "A Literature Survey on Distributed Discrete Event Simulation". *Simuletter*, Vol. 18, No. 2, pp. 11–21, June 1987.

[Kona 91] P. Konas and P.-C. Yew. "Parallel Discrete Event Simulation on Shared-Memory Multiprocessors". In: A. H. Rutan, Ed., *Proceedings of the 24^{th} Annual Simulation Symposium, New Orleans, Louisiana, USA, April 1-5, 1991.*, pp. 134–148, IEEE Computer Society Press, 1991.

[Lin 90] Y.-B. Lin and E. D. Lazowska. "Exploiting Lookahead in Parallel Simulation". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 4, pp. 457–469, October 1990.

[Lin 91] Y.-B. Lin and E. D. Lazowska. "A Study of Time Warp Rollback Mechanism". *ACM Transactions on Modeling and Computer Simulation*, Vol. 1, No. 1, pp. 51–72, January 1991.

[Luba 91] B. Lubachevsky, A. Weiss, and A. Shwartz. "An Analysis of Rollback-Based Simulation". *ACM Transactions on Modeling and Computer Simulation*, Vol. 1, No. 2, pp. 154–193, April 1991.

[Merl 76] P. Merlin and D. Farber. "A Methodology for the Design and Implementation of Communication Protocols". *IEEE Transactions on Communications*, Vol. COM-24, No. 6, June 1976.

[Misr 86] J. Misra. "Distibuted Discrete-Event Simulation". *ACM Computing Surveys*, Vol. 18, No. 1, pp. 39–65, March 1986.

[Moll 82] M. K. Molloy. "Performance Analysis Using Stochastic Petri Nets". *IEEE Transactions on Computers*, Vol. C-31, No. 9, pp. 913 – 917, Sep. 1982.

[Mura 89] T. Murata. "Petri Nets: Properties, Analysis and Applications". *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 541–580, Apr. 1989.

[Nico 91] D. M. Nicol and S. Roy. "Parallel Simulation of Timed Petri-Nets". In: B. Nelson, D. Kelton, and G. Clark, Eds., *Proceedings of the 1991 Winter Simulation Conference*, pp. 574 – 583, 1991.

[Prak 91] A. Prakash and R. Subramanian. "Filter: An Algorithm for Reducing Cascaded Rollbacks in Optimistic Distributed Simulation". In: A. H. Rutan, Ed., *Proceedings of the 24^{th} Annual Simulation Symposium, New Orleans, Louisiana, USA, April 1-5, 1991.*, pp. 123–132, IEEE Computer Society Press, 1991.

[Rama 80] C. V. Ramamoorthy and G. S. Ho. "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets". *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 5, pp. 440 –449, Sep. 1980.

[Ramc 74] C. Ramchandani. "Analysis of Asynchronous Concurrent Systems by Petri Nets". Tech. Rep., MIT, Laboratory of Computer Science, Cambridge, Massachusetts, Feb. 1974.

[Razo 84] R. R. Razouk and C. V. Phelps. "Performance Analysis using Timed Petri Nets". In: *Proc. 1984 Int. Conf. on Parallel Processing*, pp. 126 – 128, IEEE Comp. Soc. Press, 1984.

[Sifa 77] J. Sifakis. *Use of Petri Nets for Performance Evaluation*, pp. 75 – 93. North-Holland, 1977.

[Thom 91a] G. S. Thomas. "Parallel Simulation of Petri Nets". Tech. Rep. TR 91-05-05, Dep. of Computer Science, University of Washington, May 1991.

[Thom 91b] G. S. Thomas and J. Zahorjan. "Parallel Simulation of Performance Petri Nets: Extending the Domain of Parallel Simulation". In: *Proc. of the 1991 Winter Simulation Conference*, 1991. to appear.

[Yu 91] M.-L. Yu, S. Ghosh, and E. DeBenedicts. "A Provably Correct, Non-Deadlocking Parallel Event Simulation Algorithm". In: A. H. Rutan, Ed., *Proc. of the 24th Annual Simulation Symposium*, pp. 100–111, New Orleans, Louisiana, April 1991.