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

MASSIVELY PARALLEL SIMD SIMULATION OF DISCRETE TIME STOCHASTIC PETRI NETS

Subhas C. Roy

Department of Computer Science
College of William and Mary
Williamsburg, Virginia 23185, U.S.A.

ABSTRACT

Performance modeling using Petri nets is becoming increasingly popular due to their versatility. Simulation remains the only feasible method of solving Petri nets with generally distributed firing times. Previous work on parallel simulation of Petri nets mostly involved some restricted classes of timed Petri nets that have been simulated on MIMD machines. This paper proposes a synchronous algorithm for SIMD simulation of a general class of timed Petri nets with discrete transition firing times. Implementation on a MasPar MP–2 shows that good speedups are achievable and a parallel implementation can significantly outperform a sequential implementation using a workstation.

1 INTRODUCTION

Petri nets (see [Murata 1989] for a survey on Petri net theory) provide a graphical and mathematical modeling formalism that is well-suited for describing and studying systems that exhibit concurrency, synchronizations and conflicts. When augmented with the notion of time, Petri nets can be used for performance modeling and evaluation. However, for large complex systems solving Petri net models using analytic methods is formidably difficult and may be possible only at the cost of generality. Analytic techniques need to make Markovian assumptions for tractability and even then the associated exponential time and space complexities limit their usefulness beyond modest net sizes. Therefore, discrete event simulation of Petri nets is an attractive alternative since it imposes little restriction on the modeler. However, the computational resource requirement for simulation of large Petri nets can be enormous. Therefore, parallel execution of such simulations is desirable.

Discrete-time stochastic Petri nets are an important class that is suitable for modeling "clocked systems" i.e. systems where activity durations are multiple of a common time unit. Most computer hardware systems operate on a basic clock and so such systems may be modeled using discrete-time Petri nets. Also, many practical continuous-time systems can be approximately modeled with discrete-time Petri nets. This work deals with techniques of efficient parallel discrete event simulation (PDES) of a general class of discrete firing time Petri nets running on a SIMD machine.

Petri net (PN) is a directed weighted bipartite graph containing two types of nodes called *places* (represented by circles) and *transitions* (represented by bars) which are connected by directed *arcs*. Places may contain *tokens* (shown as dots) which determine the dynamic behavior of the net. The *marking* of a PN is the vector of token counts for all the places. A transition is *enabled* if each of its input places contains at least as many tokens as there are arcs from the place to the transition. An enabled transition can *fire* (in one atomic operation) by removing all of its enabling tokens from its input places and adding to each of its output places one token for each output arc. Many variations and extensions to this basic model are used by the researchers. In Petri nets with transition firing delays, a transition must remain enabled for a certain duration of time before it can fire. For modeling system behavior, transition firing times can be random variates thus leading to stochastic firing time Petri nets (called SPNs). In case multiple transitions sharing a common input place are enabled, the one with the least remaining firing time (RFT) fires. This firing policy is called the "race policy". A set of enabled transitions may form a *conflict set* such that firing one disables another thereby requiring a conflict resolution policy. For Petri nets, a simulation run explores one sample path (one sequence of transition firings).

Although several researchers have explored the techniques of PDES of Petri nets, the context of their work is MIMD only. However, execution of PDES of Petri nets on SIMD machines offers an additional set of challenges. In either case, partitioning the net is an important issue that needs serious attention in order to achieve good performance. To expose maximum level of parallelism, [Thomas and Zahorjan 1991] allows each place node or transition node to become an LP at the cost of extra communications. However for real parallel machines, the cost of communication is too high to make such fine granularity useful. Therefore, [Nicol and Roy 1991], [Nicol and Mao 1993], and [Chiola and Ferscha 1993] suggest conflict-set based LP partitioning which reduces the number of LPs and aims at minimizing the communication overhead by carefully exploiting the net structure.

The rest of the paper is organized as follows. Section 2 discusses the discrete time Petri nets and the underlying formalism used in this work. Section 3 discusses the SIMD algorithm for the simulation. Section 4 reports on our empirical study.

2 DISCRETE TIME PETRI NET (DTPN)

The firing time for each of the transitions of a SPN of this type is taken from a discrete distribution where probability weights are assigned to a finite or countably infinite number of constants. Synchronous systems can be naturally modeled by DTPNs. An SPN formalism with discrete timings was first introduced in [Molloy 1985] which allowed only geometric distribution, the discrete analogue of exponential distribution. Although we allow non-geometric distributions, in the context of this paper, we assume that there is a basic time-step ω for a DTPN such that all events (transition firings) occur only at multiples of ω. Any discrete firing time distribution can be approximated arbitrarily well with a discrete distribution with a small enough basic time-step ω. SPN with deterministic firing times that are multiple of ω are just a special case of this model. If the distributions are geometric, the underlying stochastic process is a discrete time Markov chain (DTMC). Using state expansion, a DTMC can still be obtained [Ciardo, German, and Lindemann 1993] even if the distributions are not geometric as long as the firings occur only at some multiple of a basic step ω.

2.1 Conflict Sets

Since in a DTPN a firing can only occur at a time $\theta \in \{i\omega | i \in \mathbb{N}\}$, there is a nonzero probability that multiple transitions will attempt to fire at the same time. This gives rise to the possibility of conflict situations where two or more enabled transitions are trying to fire at the same instant and firing one disables some other.

We define **'conflicts with'** relation for a DTPN (with a set of transitions T) at time θ and with a marking μ such that the transitions in a set $T_c \subseteq T$ are said to conflict with each other if (a) all transitions $\in T_c$ are enabled in μ, (b) all transitions $\in T_c$ have equal RFTs at θ, and (c) the firing of any subset $S \subset T_c$ results in a marking in which some other transition $t \in T_c$ and $t \notin S$ is disabled.

A set with property $\mathcal{P}$ is a *maximal* set with property $\mathcal{P}$ if it is not a proper subset of any other set with property $\mathcal{P}$. A **conflict set** of a DTPN at time θ and with a marking μ is a maximal set of transitions in the DTPN with the property that its members conflict with each other at time θ and in marking μ. A **static conflict set** is a maximal set of transitions in the DTPN with the property that its members conflict with each other in a legal marking at some point of time ('conflict with each other' is not meant to be symmetric—it is possible in the presence of inhibitor arcs to have asymmetry in the 'conflicts with' relation).

If there is an *inhibitor* arc (the arrowhead of such an arc is pictorially shown by a small circle) from a place to a transition, the transition remains disabled as long as the place contains a token. The presence of *immediate* transitions (which have zero firing delay) and inhibitor arcs significantly complicates the conflict situations. In a marking many transitions together may form an extended conflict set due to the transitive 'conflicts with' relation. Figure 1(b)-(d) illustrate a few conflict situations assuming that in each case all transitions are enabled and they have the same RFTs. In Figure 1(b) and 1(c), the conflicts are due to the sharing of common input places (called decision places). In Figure 1(d), although the transitions t_1 and t_2 do not share any input place, they cannot be allowed to fire at the same time-step because the resulting marking would be an illegal one (i.e. the new marking cannot occur in the un-timed version of the PN).

The conflict situations must be resolved in a way that is meaningful to the modeler and avoids reaching a forbidden marking. This is an important and difficult problem in DTPN and no easy and satisfactory conflict resolution scheme is available. During a simulation execution, it is computationally expensive (exponential complexity in the worst case) to detect a conflict situation, compute the conflict set dynamically and fire the transitions of the conflict set using a resolution scheme. One way [Ciardo, German, and Lindemann 1993] to ensure unambiguous semantic, is to sequentialize the simultaneous firing attempts

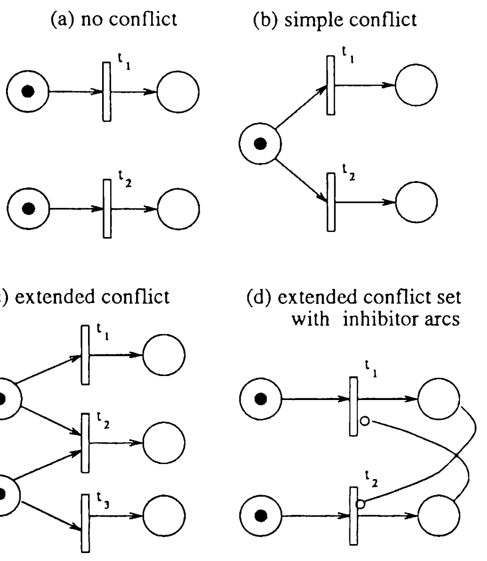

Figure 1: Conflict Sets

of the transitions of a conflict set. The next transition to be fired is stochastically chosen by using firing weights assigned to the conflicting transitions. We have adopted this approach due to its simplicity.

In [Holliday and Vernon 1987], another approach based on enumeration of Maximal sets is proposed (note that this term has its M as a block letter in order to differentiate it from the set theoretic term 'maximal set'). For a conflict set S, a Maximal set $U \subset S$ has a property such that all transitions in U can successfully fire simultaneously and no other transition in S can also fire at the same time-step. In Figure 1(c) the only Maximal sets are $\{t_1, t_3\}$ and $\{t_2\}$. In this scheme, the idea is to enumerate all the Maximal sets pertaining to the conflict set, assign a probability weight to each of the Maximal sets, stochastically choose a Maximal set using the probability weights, and then fire all the transitions of that Maximal set (in any order). A problem with this scheme is the exponential time and space complexity (in the worst case) of enumerating the Maximal sets. Also, we expect this approach to be not well-suited for execution on the SIMD architecture.

2.2 DTPN Formalism

We have adopted a DTPN formalism that is based (with some modifications) on [Ciardo, German, and Lindemann 1993]. Formally, we define a DTPN as a tuple $\mathcal{A} = \{P, T, I, O, H, \mu_0, \tau^0, F, w\}$ where

- $P = \{p_1, p_2, \ldots, p_{|P|}\}$ is a finite set of places. A marking $\mu \in \mathbb{N}^{|P|}$ is a vector of token counts of the places and $\#(p, \mu)$ denotes the token count of place p in marking μ.
- $T = \{t_1, t_2, \ldots, t_{|T|}\}$ is a finite set of transitions.
- $\forall p \in P, \forall t \in T, I_{p,t} \in \mathbb{N}, O_{p,t} \in \mathbb{N}$ and $H_{p,t} \in \mathbb{N}$ denote the multiplicities of the input arc from p to t, the output arc from t to p and the inhibitor arc from p to t respectively.
- $\mu_0 \in \mathbb{N}^{|P|}$ is the initial marking. Since the state at time 0 may reflect transitions which have been enabled for some time, $\tau_t^0 \in \{i\omega | i \in \mathbb{N}\}$ is the RFT at time $\theta = 0$ for $t \in T$.
- $\forall t \in T, F_t$ is the CDF (cumulative distribution function) for the firing time distribution for $t \in T$ such that $\text{rnd}(F_t) \in \{i\omega | i \in \mathbb{N}\}$ where $\text{rnd}(D)$ is a random variate from the distribution D.
- $\forall t \in T, \forall S \in 2^T, w_{t|S} \in \mathbb{R}^+$ is the firing weight for transition t when the transitions in S are all attempting to fire simultaneously.

A transition $t \in T$ is enabled in marking μ iff

$$\forall p \in P, I_{p,t} \leq \#(p,\mu) \wedge (H_{p,t} > \#(p,\mu) \vee H_{p,t} = 0)$$

After a transition t enabled in marking μ fires, the marking changes to a new marking $f(t,\mu)$ such that

$$\forall p \in P, \#(p, f(t,\mu)) = \#(p,\mu) - I_{p,t} + O_{p,t}$$

A DTPN changes state only at a time $\theta \in \{i\omega | i \in \mathbb{N}\}$. Since there can be non-geometric distributions too, the state s of a DTPN needs to specify the RFT vector as well (as does the underlying Markov chain in such a case). Thus as a vector of markings and residual firing times the state is expressed as $s = (\mu, \tau)$, where τ_t denotes the RFT of $t \in T$. Assume that, at time θ, DTPN is in state $s = (\mu, \tau)$. Define $E(\mu)$ to be the set of enabled transitions in marking μ. Also let $\tau^* = \min_{t \in E(\mu)}\{\tau_t\}$ and $S = \{t \in E(\mu) | \tau_t = \tau^*\}$. Then, at time $\theta + \tau^*$, the probability that $t \in S$ will fire next is $w_{t|S}/(\sum_{u \in S} w_{u|S})$. After firing t, the new state of the DTPN will be $s' = (\mu', \tau')$, where the new marking is $\mu' = f(t,\mu)$. The new RFT vector $\tau' = g(t, \mu, \tau)$ is given as follows. $\forall u \in T$,

$$\tau'_u = \begin{cases} \text{rnd}(F_u) & \text{if } (u = t \vee u \notin E(\mu)) \wedge u \in E(\mu') \\ \tau_u - \tau^* & \text{if } u \neq t \wedge u \in E(\mu) \wedge u \in E(\mu') \\ \infty & \text{if } u \notin E(\mu') \end{cases}$$

2.3 DTPN Simulation : Formal Description

Consider the transient simulation of a DTPN $\mathcal{A}$ from the time 0 to a finite time θ_F. Algorithm 1 outlines a formal way how to simulate one sample path of $\mathcal{A}$ (based on our adopted formalism).

```
$\theta \leftarrow 0$; $\mu \leftarrow \mu_0$; $\tau \leftarrow \tau^0$;
repeat forever
    $\tau^* \leftarrow \min_{t \in E(\mu)}\{\tau_t\}$;
    if $\theta + \tau^* > \theta_F$ then stop simulation.
    $S \leftarrow \{t \in E(\mu) | \tau_t = \tau^*\}$;
    select $t \in S$ with probability $w_{t|S} / \sum_{u \in S} w_{u|S}$;
    $\mu' \leftarrow f(t, \mu)$; $\tau' \leftarrow g(t, \mu, \tau)$;
    $\mu \leftarrow \mu'$; $\tau \leftarrow \tau'$;
    $\theta \leftarrow \theta + \tau^*$;
```

Algorithm 1: Sequential Simulation of DTPN

Note that in Algorithm 1, all firings are strictly sequentialized. Although this eliminates any ambiguity, strict sequentialization is not always essential—from a given marking there may be many transitions enabled to fire which are independent in the sense that the order in which they fire does not matter. Also note that the algorithm is not practical for direct implementation since it theoretically requires all $|T| \times 2^{|T|}$ firing weights for specifying $w_{t|S} : T \times 2^T \rightarrow \mathbb{R}^+$. In practice, however, most of this values are not necessary and we can avoid specifying them.

3 SIMD SIMULATION OF DTPN

A DTPN simulation jumps through the time-steps $\in \{i\omega | i \in \mathbb{N}\}$. For a large sized DTPN consisting of many thousands of places and transitions, it is expected that a large number of firing events occur at each of these time-steps. Such abundant parallelism can be naturally exploited in a SIMD (Single Instruction Multiple Data) machine where the processors execute in a locked-step manner. The SIMD architecture is characterized by numerous synchronized PEs which are broadcast a single instruction stream by a control processor. The PEs process the local data in their respective local memories using the common instruction thread. Conditional instructions are executed by the *active set* of PEs that are enabled by the data dependent condition code. Efficient support for global reduction operations and near neighbor communications are provided.

However, ensuring correct parallel execution (which avoids reaching an illegal marking) of DTPN simulation needs some careful considerations regarding the decision on choosing the transitions for simultaneous firing. Although strict sequentialization similar to Algorithm 1 is unnecessary, sometimes it is needed as in the case of Figure 1(d). For efficiency, we want to make the execution model model fit the SIMD architecture. This may call for breaking away from the traditional future-event-list oriented simulation approach.

3.1 Partitioning the Net into LPs

For simulation of a DTPN on a parallel machine, we need to partition the net into LPs such that correctness of the DTPN execution semantics is ensured as well as communication among LPs is kept to a minimum. Like [Nicol and Roy 1991] and [Chiola and Ferscha 1993], our partitioning scheme is essentially based on static conflict sets. However we make additional restrictions in order to deal with the immediate transitions. Our partitioning scheme makes it possible to keep all firing decisions (in particular conflict resolutions) at any time-step completely local to an LP. To allow the LPs to determine the enabling status of its transitions, we allocate each place to the same LP as that of its output transitions. Thus all inter-LP arcs originate from transitions only.

Using the previous notations, suppose at time $\theta \in \{i\omega | i \in \mathbb{N}\}$, the state of the DTPN to be simulated is (μ, τ). Suppose, based on the firing weights, we have selected for firing the transition $t \in E(\mu), \tau_t = \tau^*$ where $\tau^* = \min_{t \in E(\mu)}\{\tau_t\}$. Also, after the firing at time $\theta + \tau^*$, the resulting state becomes (μ', τ') where $\mu' = f(t, \mu)$ and $\tau' = g(t, \mu, \tau)$.

To form n LPs, we carefully partition T into n disjoint subsets $T_1, T_2, \ldots, T_n$. Define $FS(k, \beta)$ to be the set of *firable* transitions in LP k at time β such that all transitions in $FS(k, \beta)$ have the same RFTs and they are scheduled to fire at β.

Let the transition that is selected for firing at time $\theta' = \theta + \tau^*$ belongs to LP i which means $t \in T_i, 1 \leq i \leq n$. We do the partitioning such that there will not exist any transition $u \in T_j, 1 \leq j \leq n, i \neq j$ for which any of the following assertions holds:

(a) $u \in E(\mu) \wedge \tau_u = \tau^* \wedge u \notin E(\mu')$

(b) $u \notin E(\mu) \wedge u \in E(\mu') \wedge \tau'_u = 0$

'The assertion (a) cannot hold true' implies that firing $t \in FS(i, \theta')$ cannot disable $u \in FS(j, \theta')$ if $i \neq j$. This condition holds if the static conflict sets are not split across LPs.

'The assertion (b) cannot hold true' implies that firing $t \in FS(i, \theta')$ cannot add a member to the set of firable (at time θ') transitions in LP j if $i \neq j$. We need to satisfy this condition in order to deal with the immediate transitions. Consider Figure 2 where the transition t_2 is an immediate transition (firing distribution is Const(0)). If we assign p_1 and t_1 to LP i and the rest of the transitions and places to LP j where $i \neq j$, then condition (b) is violated, because if t_1 fires, t_2 will become enabled and will try to fire at the same time-step as t_1. Therefore t_2 must be assigned to the same LP as t_1.

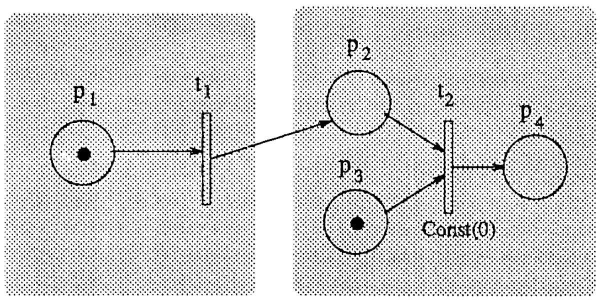

Figure 2: Partitioning and Immediate Transitions

This way of partitioning forces all transitions of a conflict set to be completely included in an LP. This scheme allows all conflict resolution decisions to be taken completely locally within an LP without any need for communication with other LPs. The implementation of the "race policy" too is done locally.

So, we partition the net into LPs as follows. First partition the set of transitions T into the static conflict sets $T_1, T_2, \ldots, T_n$. Now, for $t_1 \in T_i$ and an immediate transition $t_2 \in T_j$ where $i \neq j$, if the set of output places of t_1 and the set of input places of t_2 are not disjoint, then merge T_i and T_j. Finally, allocate each place to the unique LP which contains its output transitions. Multiple LPs each containing small number of nodes may be merged together for better load balancing.

For conflict resolution, we specify the firing weights on a per-LP basis. For LP i, $w_{t|S}$ values need to be specified for each $t \in T_i$ and each $S \subseteq T_i$ where T_i is the set of transitions assigned to the LP i. In an actual implementation this can be further simplified by the knowledge of the problem. For example, in some restricted classes of Petri nets such as Free Choice nets, the number of subsets $\in \{S \subseteq T_i\}$ that actually are required to be specified is very small.

3.2 SIMD Algorithm for DTPN Simulation

The basic idea is that if we do the partitioning as described, then at a given time-step, we can correctly simulate two firing events belonging to two different LPs in parallel. There is a common global clock. At a given time θ, within an LP i we compute $FS(i, \theta)$ and we pick a transition from that firable set using the discrete probabilities distribution involving the firing weights. As a result of firing tokens may be deposited to other LPs which may involve inter-PE communication. Due to the firing of t, some other transitions in LP i may get disabled or enabled. New RFTs are computed. Now the clock is incremented by the global minimum RFT (among all LPs).

The SIMD algorithm for PDES of DTPN $\mathcal{A} = \{P, T, I, O, H, \mu_0, \tau^0, F, w\}$ is described as Algorithm 2. The set of transitions is partitioned into $T_1, T_2, \ldots, T_n$ thus forming n LPs. For simplicity of exposition, we assume here that each PE contains only one LP. However, this restriction is not necessary.

```
1. (a) Initialize $(\mu, \tau)$;
   (b) Assign the LPs to the PEs;
   (c) $\theta \leftarrow 0$;
2. $\tau^* = \min_{t \in E(\mu)}\{\tau_t\}$; $\theta \leftarrow \theta + \tau^*$;
3. If $\theta > T_F$, stop simulation.
4. For each LP $i$, $S_i \leftarrow FS(i, \theta)$;
5. For PE $i$ do in parallel:
     if $S_i \neq \emptyset$
   (a) Choose a transition $t \in S_i$ to fire with
       probability $w_{t|S_i} / \sum_{u \in S_i} w_{u|S_i}$;
   (b) Fire the transition $t$: move the tokens
       to obtain the new marking and update
       RFTs of the local transitions;
6. Go to step 2;
```

Algorithm 2: SIMD Algorithm for Simulating DTPN

Our DTPN model allows a border transition t to be an immediate transition or to be a part of a conflict set. If t is an immediate transition, it may fire instantly as a consequence of the firing of another transition which feeds t's input place(s). On the other hand, if $t \in S$ belongs to a conflict set S where $|S| > 1$, then a previously enabled t may get disabled due to either the "race policy" or an inhibitor arc or a conflict resolution. Immediate transitions, inhibitor arcs, conflicts, and the "race policy" are handled by Algorithm 2 quite naturally. However a standard simulation based on event-lists needs to do explicit event preemption to handle these features. More importantly, computation of good lookaheads for such a general case is difficult. An enabled stochastic border transition can get disabled and later re-enabled with possibly smaller firing duration time. Thus the final firing instant of a transition may be hard to predict. A common way of computing the lookahead in PDES of Petri nets is to estimate the lower bound on the firing time of a border transition. In presence of such event preemption, obtaining good lookaheads is difficult. A general algorithm that can efficiently compute the lookaheads by extensively analyzing the net pertaining to the LP is not available. For restricted class of DTPNs with only non-immediate border transitions which are also non-preemptive (i.e. singleton conflict sets), the computation of lookahead is simpler.

3.3 PDES on the SIMD Architecture

A SIMD machine has a single thread of control. There are thousands of PEs each with a small amount of memory. Due to these distinctions, we need to design different PDES algorithms for SIMD machines which may be quite different from the corresponding MIMD algorithms. Not all problems are good candidates for parallel execution on SIMD machines. The problems which have some regular structure, finely partitionable and without bottleneck communication pattern are suitable for SIMD machines. Since these machines support very efficient near neighbor communications, effective mapping of the problem to the PE interconnection network topology can greatly improve the performance. Also, problems with underlying synchronous systems naturally fit the SIMD execution style. Load balancing problems are more severe in the SIMD context since there is no control parallelism and the PEs are completely synchronized. For every operation, the time taken is the worst case time for all PEs. For example, in a PDES the time taken by all PEs to do single insertions to their respective local linked-lists is the same as the time taken by the PE requiring the maximum number of node traversals. Therefore a conventional MIMD algorithm for PDES will perform poorly on the SIMD architecture.

For efficient execution of DTPN simulation on a SIMD machine, we have deviated from the traditional PDES approaches. Algorithm 2 is essentially synchronous time-stepping (with all LPs synchronized by a common global clock) and can be considered to be a direct parallel extension of Algorithm 1. Note however that the usual overhead of a time-stepped simulation—scanning for work during idle time periods—is absent, since the clock is incremented by the global minimum RFT among all transitions in all PEs. Global minimums are computed quickly on a SIMD machine. Worst-case cost for linked-list oriented simulation is avoided since no list for future events is maintained. Although the synchronization approach is conservative, it does not involve lookahead computations. Event preemptions (which are needed for the implementation of 'race policy' which means firing the transition with the smallest RFT among the competing enabled transitions) which complicates a standard PDES protocol, is automatically taken care of. Since the local memory of a PE is very limited in a SIMD machine, the number of nodes (i.e. places or transitions) assigned per LP should not be large. Instead of using linked-lists, our SIMD implementation scans through the arrays (with a small upper bound on the size) of places and transitions.

4 EMPIRICAL STUDY

We have implemented Algorithm 2 on a MasPar MP-2 with 1024 processors. MasPar's MP series SIMD architecture comprises a two-dimensional matrix of PE array (size ranges from 1K to 16K) where two separate communication mechanisms called *x-net* and *router* are provided. The x-net communication is supported by a two-dimensional toroidal mesh interconnect which allows efficient near neighbor communications along the rows, columns and the diagonals. The global router communication mechanism (which is for general any-to-any inter-PE communications) is supported by a butterfly-like multistage crossbar interconnect. Each PE is a load/store arithmetic processor with register space and 64K bytes of local data memory. The instructions are broadcast to the PE array from the *array control unit* (ACU). There is efficient support for global reductions and the scan operations.

As noted earlier, our program implementing Algorithm 2 has a common global clock for all PEs. Since all LP data arrays are bounded from above by relatively small constants (due to the limited amount of PE memory), instead of linked-lists, operations are performed on fixed-sized arrays thus avoiding the overhead of dynamic linked-list management on a SIMD machine. The maximum number of communication partners for each LP is assumed to be relatively small. In the cases where the Petri net to be simulated involves only regular near neighbor communications, the program uses the x-net mechanism. Any sophisticated resolution policy for the conflicting transitions has not yet been implemented—the firing weights are assigned by the user in an application dependent way using complete knowledge of the nature of the conflicts involved.

The implementation of a SIMD algorithm for PDES of DTPN needs some special care. For example, we need to correctly handle the memory conflicts during which two or more PEs simultaneously attempt to write to the same memory location belonging to another PE. The result of such a simultaneous operation may be incorrect or undefined—for example the machine might automatically serialize those write attempts and thus the PE which writes last prevails. Such situations can frequently occur in the SIMD simulation of a Petri net as different transitions belonging to different PEs attempt to add tokens to their common output place contained in another PE. To solve this problem, we do the following in order to sequentialize all simultaneous write attempts. Each PE which attempts to write to another PE flags itself recording (i) that it intends to write, (ii) the destination PE's id, (iii) the number of tokens to be added, and (iv) the id of the place (in the destination PE) where the tokens will be added. Once all such competing PEs have recorded these informations in their respective local variables, the destination PE starts its action—it reads and checks those variables from all of the PEs which might send tokens to it. If it finds any such PE that intends to send tokens and the destination PE id matches with its own processor id, the token addition operation is performed.

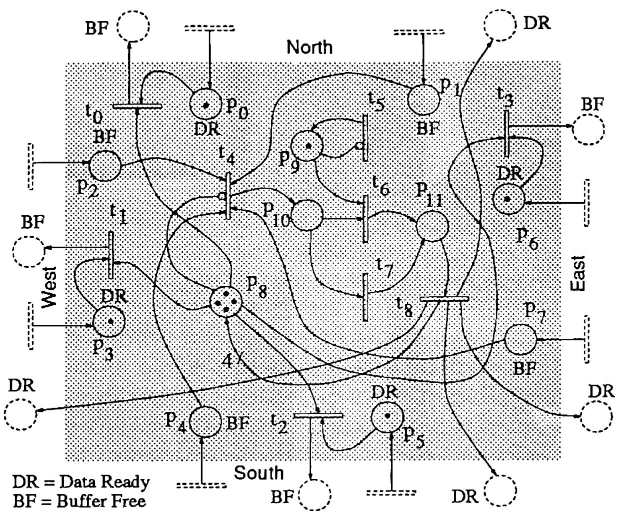

Figure 3: DTPN Model Used in the Empirical Study

4.1 Petri Net Model

We have run experiments using a discrete timed Petri net which contains a mixture of deterministic and geometric distributions. There are also inhibitor arcs and immediate transitions. This DTPN models computation, communication and buffering synchronizations with the neighbors in a regular two-dimensional torus network of processing elements. The net is built using a replicated module (shown in Figure 3) that models the logic for each processing element. The PN model fits the MP-2 architecture nicely thus making x-net communication attractive. Each LP (the module) has 12 places, 9 transitions and 4 communication partners (NEWS neighbors). The maximum size of any conflict set is 2. The communication among PEs is needed only when a transition adds tokens to a place contained in a different PE.

Refer to Figure 3. BF (buffer-free) type places p_1, p_2, p_4 and p_7 are there to indicate that the data supplied to the corresponding neighbor has been consumed and the corresponding output buffer is now free. The transition t_4 cannot fire until all such buffers are free. The place p_8 has an inhibitor arc to t_4 so as to ensure that it cannot fire until the input data from all the four neighbors have been fetched. DR (data-ready) type places p_0, p_3, p_5 and p_6 are for indicating that data from the corresponding neighbors are available. The firings of transitions t_0, t_1, t_2 and t_3 model the fetching actions of those data. Transition t_4 fires after all the input data from the neighbors are fetched and the output buffers are freed. The stochastic firing time transitions t_5, t_6 and t_7 jointly model the variable (data-dependent) time needed to do the computations on the input data. As soon the results of the computations are available (as indicated by the place p_{11}), the neighbors are notified. The notification action is modeled by the immediate transition t_8.

4.2 Execution Performance

For a sample run, with approximately one million firing events, of the simulation of this Petri net with 1024 LPs, the timings (ignoring the I/O) observed are shown in Table 1. An optimized sequential code for the same simulation was run on a SPARC IPC workstation for comparison. The parallel timings in Table 1 were obtained using a 1024 processor MP-2 with one LP per PE.

Table 1: Comparison of Execution Timings

machine	*time*	*relative*
using x-net on MP-2	13.9 sec	1.0
using router on MP-2	16.9 sec	1.2
sequential (workstation)	380 sec	27.3

Execution profile data of the runs on MP-2 show that when we use the global router mechanism, about 20% of the execution time is spent in communications whereas in the runs using x-net, only 2-3% of the execution time is spent in communications. Computation of the global minimum firing time (which is done at every iteration) took less than 2% of the total execution time. As we expected, the routine where maximum amount (20-30%) of the execution time was spent is the one that checks if a transition is enabled or not.

The efficiency of the parallel program improves as we increase the number of LPs per each PE. This is apparent from Figure 4 which shows that the average execution time per firing decreases with the higher number of LPs per PE. Since within each PE there are multiple LPs, as soon as the current LP being attended to runs out of firing events to be processed, the PE can switch to another LP which has firable transitions. Thus idle time is reduced.

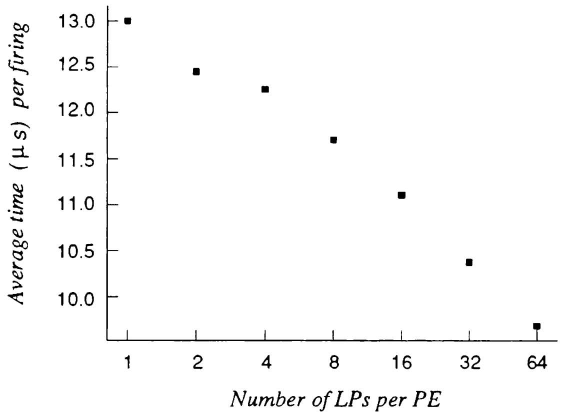

Figure 4: Effect of LP Aggregation

5 CONCLUSION

This paper presents techniques of efficient parallel simulation of a general class of discrete time Petri nets (DTPNs) on SIMD machines. Such Petri nets are useful in modeling many synchronous systems that operates on a basic clock. In this paper DTPNs have been discussed in detail and we elucidated on the problem involving conflict sets which poses challenge to the parallelization particularly in presence of inhibitor arcs and immediate transitions. We have proposed a synchronous SIMD algorithm that we believe naturally suits the simulation of DTPNs with an underlying basic time-step. In the context of SIMD simulation of generalized discrete timed Petri nets, we consider this simple algorithm to be more suitable than the standard PDES synchronization protocols. The partitioning technique we have used is essentially static conflict based. Experiments with an implementation of the algorithm on a 1024 processor SIMD machine MP-2 demonstrates the viability of the synchronous approach of the algorithm which does not use any linked-list of future events. We are presently working towards a more efficient SIMD algorithm as well as studying the partitioning and load balancing problems that arise in this context.

ACKNOWLEDGEMENT

We are indebted to Gianfranco Ciardo and David Nicol for many useful discussions. This work is partially supported by the NASA under Grant No. NAG 1-1132.

REFERENCES

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

Ciardo, G., R. German, and C. Lindemann. 1993. A characterization of the stochastic process underlying a stochastic Petri net. In *Proceedings of the Fifth International Workshop on Petri Nets and Performance Models (PNPM93).* Toulouse, France.

Holliday, M. A., and M. K. Vernon. 1987. A Generalized Timed Petri Net Model for Performance Analysis. *IEEE Transactions on Software Engineering* SE-13(12):1297–1310.

Molloy, M. K. 1985. Discrete Time Stochastic Petri Nets. *IEEE Transactions on Software Engineering* SE-11(4):417–423.

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

Nicol, D. M., and S. C. Roy. 1991. Parallel Simulation of Timed Petri Nets. In *Proceedings of the 1991 Winter Simulation Conference*, 574–583.

Nicol, D. M., and W. Mao. 1993. Automatic Parallelization of Timed Petri-Net Simulations. Technical Report 93-91, ICASE, NASA Langley Research Center, Hampton, Virginia.

Thomas, G. S., and J. Zahorjan. 1991. Parallel Simulation of Performance Petri Nets: Extending the Domain of Parallel Simulation. In *Proceedings of the 1991 Winter Simulation Conference*, 564–573.

AUTHOR BIOGRAPHY

SUBHAS C. ROY is a PhD candidate in the department of Computer Science at the College of William and Mary, Williamsburg, Virginia. He received a B.E. degree in Computer Science and Engineering from Jadavpur University, Calcutta, India in 1987 and M.S. degree in Computer Science from College of William and Mary in 1991 respectively. His research areas of interest include distributed simulation, parallel computation and modeling.