

# Timed Colored Petri Net Models of Distributed Memory Multithreaded Multiprocessors

W.M. Zuberek<sup>†</sup>, R. Govindarajan<sup>‡</sup> and F. Suciu<sup>†</sup>

<sup>†</sup>Department of Computer Science  
Memorial University of Newfoundland  
St. John's, Canada A1B 3X5

<sup>‡</sup>Supercomputer Education and Research Center  
Indian Institute of Science  
Bangalore 560 012, India

## Abstract

Distributed-memory multithreaded multiprocessors are composed of a number of (multithreaded) processors, each with its memory, and an interconnecting network. The long memory latencies and synchronization delays are tolerated by context switching, i.e., by suspending the current thread and switching the processor to another 'ready' thread provided such a thread is available. Because of very simple representation of concurrency and synchronization, timed Petri net models seem to be well suited for modeling and evaluation of such systems. Colors are used to represent the progress of remote memory access requests in the interconnecting network as well as to fold the models of individual processors.

This paper describes timed colored Petri net models of several multithreaded multiprocessor architectures, and presents some performance characteristics obtained by evaluation of these models.

## 1. Introduction

Multithreaded processors utilize the simple and efficient sequential execution technique of control-flow combined with data-flow like concurrency [14]. This supports the conceptually simple but quite powerful idea of rescheduling rather than blocking when waiting for data, e.g., from large and distributed memories, and thus can be used for tolerating long data transmission latencies. Multithreading makes multiprocessing far more efficient because the cost of moving data between distributed memories and processors can be hidden by other activities. The same hardware mechanisms can be used to synchronize interprocess communication and to alleviate operating system overheads.

Several multithreaded architectures have recently been proposed which differ in the implementation of multithreading [1, 3, 5, 7, 9]. Switching from one thread to another can be performed under different circumstances [4]:

- Switching on every instruction: one instruction is picked from each of runnable threads and is inserted into the processor's pipeline; if there are many threads, then each stage of the pipeline is executing an instruction from a different thread, and no instruction dependency problems exist [18].
- Switching on block of instructions: blocks of instructions from different threads are interleaved.
- Switching on every load: whenever a thread encounters a load instruction, the processor switches to another thread after that load instruction is issued; the context switch is irrespective of whether the data is local or remote [3].
- Switching on remote load: processor switches to another thread only when current thread encounters an access to remote memory [1].

The nodes of a multithreaded multiprocessor are linked by an interconnecting network. This network can be a two-dimensional torus-like network or a hypercube-type connection. It is assumed that all messages in the system are routed along the shortest paths, but in a non-deterministic manner. That is, whenever there are multiple (shortest) paths between the source and destination, any of the paths is equally likely to be taken. Consequently, the traffic on the links of the interconnecting networks is assumed to be uniformly distributed. The delay of each message is proportional to the number of hops between the source and destination nodes, and it also depends upon the traffic in the chosen path. A pair of network interfaces, one for outbound and the second for inbound traffic, connect the network switches with processor nodes.

Petri nets have been proposed as a simple and convenient formalism for modeling systems that exhibit parallel and concurrent activities [16, 15]. In order to take the durations of these activities into account, several types of Petri nets *with time* have been proposed by assigning *firing times* to the transitions or places of a net. In timed nets [20], deterministic or stochastic (exponentially distributed) firing times are associated with transitions, and transition firings occur in real-time, i.e., tokens are removed from input places at the beginning of the firing period, and they are deposited to the output places at the end of this period. In color nets [11], attributes (called colors) are associated with tokens, so different activities can be assigned to tokens of different types (i.e., colors), within the same structure of the net. For modeling of multithreaded architectures, colors are used for two different purposes. One use of token attributes is to forward requests of remote memory accesses to target nodes and then send the responses back to home nodes. The second use is to fold all the (identical) processor subnets.

Analyzing the performance of multiprocessor architectures is rather involved as it depends on a number of parameters related to the architecture — memory latency time, context switching time, switch delay in the interconnection network — and a number of application parameters — number of parallel threads, runlengths of threads, remote memory access pattern and so on. The performance of multithreaded architectures have been evaluated using discrete-event simulation [9, 12, 7], analytical models — using either queuing networks or Petri nets [2, 17], or using trace-driven simulation [19]. Event-driven simulation of the timed colored net models [21] was used to obtain the results presented in this paper.

This paper presents Petri net models of several multithreaded multiprocessor architectures, discusses the development of colored net models and includes some performance results as an illustration of model analysis. The influence of architectural and application parameters on the performance of the system is also discussed.

## 2. Timed Petri nets

This section first recalls elementary concepts of place/transition nets and colored nets, and then reviews the basics of timed Petri nets.

### 2.1. Basic concepts of Petri nets

The basic marked place/transition Petri net  $\mathcal{M} = (P, T, A, m_0)$  is usually defined as a system composed of a finite, nonempty set of places  $P$ , a finite, nonempty set of transitions  $T$ , a set of directed arcs  $A$ ,  $A \subset P \times T \cup T \times P$ , and an initial marking function  $m_0$  which assigns nonnegative numbers of so called tokens to places of the net,  $m_0 : P \rightarrow \{0, 1, \dots\}$ . Usually the set of places connected by (directed) arcs to a transition is called the *input set* of a transition, and the set of places connected by (directed) arcs outgoing from a transition, its *output set*.

A place is shared if it belongs to the input set of more than one transition. A net is conflict-free if it does not contain shared places. A shared place is (generalized) free-choice if all transitions sharing it have the same input sets. Each free-choice place determines a class of free-choice transitions sharing it, and free-choice classes of transitions determined by different free-choice places are disjoint. It is assumed that selection of a transition for firing in each free-choice class of transitions is a random process which can be described by (free-choice) probabilities assigned to transitions. Moreover, it is usually assumed that the random choices in different free-choice classes are mutually independent.

A shared place which is not free-choice, is a conflict place. Enabled transitions sharing a conflict place are called conflicting transitions. The class of conflicting transitions is defined in a transitive way, i.e., two transitions are conflicting if they share a conflict place or if there exists another transition such that it shares a conflict place with one of these two transitions and is conflicting with the other transition. The conflicting relation is an equivalence relation in the set of transitions, which implies a partition of this set into disjoint classes of conflicting transitions. For each conflicting class, the probabilities of firings can be determined on the basis of relative frequencies of transition firings [10]; the probability of firing an enabled transition  $t$  is determined as the ratio of  $t$ 's (relative) frequency to the sum of relative frequencies of all transitions in the conflict class of  $t$ . Quite often such relative frequencies (and probabilities of firings) are dynamic, depending upon the marking function, for example, by using the number of tokens in a place rather than a fixed, constant number as the relative frequency. The determination of conflicting transitions depends upon the marking function, so the probabilities of firing conflicting transitions must be determined in a dynamic way.

In basic nets, the tokens are indistinguishable, so their distribution can be described by a simple marking function  $m : P \rightarrow \{0, 1, \dots\}$ . In colored Petri nets [11], tokens have

attributes called colors, so a marking function becomes  $m : P \rightarrow C \rightarrow \{0, 1, \dots\}$  where  $C$  is a finite set of token colors (or attributes). Token colors can be modified by (firing) transitions and also a transition can have several different occurrences (or variants) of its firings for different combinations of (colored) tokens in its input places. This is captured by a definition of a colored net  $\mathcal{C} = \{P, T, A, C, a, m_0\}$ , where the *arc* function  $a$  describes the numbers of colored tokens required in input places and deposited to output places for different *colors* of transitions' firings (or different occurrences of transitions),  $a : A \rightarrow C \rightarrow C \rightarrow \{0, 1, \dots\}$  (the definition is slightly different from the one used by Jensen [11], but is consistent with it). It should be observed that if  $C$  contains just one color, the colored net reduces to a 'standard' place/transition net.

The basic idea of colored nets is to 'fold' an ordinary place/transition net. The original set of places is partitioned into a set of disjoint classes, and each class is replaced by a single place with token colors indicating which of the original places the tokens belong to. Similarly, the original set of transitions is partitioned into a set of disjoint classes, and each class is replaced by a single transition with occurrences indicating which of the original transitions the firing corresponds to. Any partition of places and transitions will result in a colored net. One of the extreme partitions will combine all original places into one place, and all original transitions into one transition; this will create a very simple net (one place and one transition only) but with quite complicated rules describing the use of colors. The other extreme partition will create one-element classes of places and transitions, so the colored net will be isomorphic to the original net, with only one color. To be useful in practice, colored nets must constitute a reasonable balance between these two extreme cases.

## 2.2. Timed Petri nets

In order to study performance aspects of Petri net models, the duration of activities must also be taken into account and included into model specifications. Several types of Petri nets 'with time' have been proposed by assigning firing times to the transitions or places of a net. In timed nets, firing times are associated with transitions (or occurrences), and transition firings are 'real-time' events, i.e., tokens are removed from input places at the beginning of the firing period, and they are deposited to the output places at the end of this period (sometimes this is also called a 'three-phase' firing mechanism as opposed to a 'one-phase', instantaneous firings of nets without time or stochastic nets).

In timed nets, all firings of enabled transitions are initiated in the same instants of time in which the transitions become enabled (although some enabled transition cannot initiate their firings). If, during the firing period of a transition, the transition becomes enabled again, a new, independent firing can be initiated, which will overlap with the other firing(s). There is no limit on the number of simultaneous firings of the same transition (sometimes this is called 'infinite firing semantics'). Similarly, if a transition is enabled 'several times' (i.e., it remains enabled after initiating a firing), it may start several independent firings in the same time instant.

A timed (colored) net can be defined as a quadruple  $\mathcal{T} = (\mathcal{N}, m_0, c, f)$ , where  $\mathcal{N}$  is a (colored) net structure,  $m_0$  is the initial marking function,  $c$  is the choice function which assigns free-choice probabilities to free-choice occurrences of transitions and relative frequencies of firings to (potentially) conflicting occurrences of transitions,  $c : T \rightarrow C \rightarrow \mathbf{R}^+$ ,

and  $f$  is the firing time function which assigns the (average) firing times to occurrences of transitions,  $f : T \rightarrow C \rightarrow \mathbf{R}^{\oplus}$ .

The firing times of some transitions may be equal to zero, which means that the firings are instantaneous; all transitions with zero firing times are called *immediate* (while the other are called *timed*). Since the immediate transitions have no tangible effect on the (timed) behavior of the model, it is convenient to assume that first all (enabled) immediate transitions are fired, and then (still in the same time instant), when no more immediate transitions are enabled, to start the firings of (enabled) timed transitions. It should be noted that such a convention introduces the priority of immediate transitions over the timed ones, so the conflicts of immediate and timed transitions should be avoided. Similarly, the free-choice classes of transitions must be ‘uniform’, i.e., all transitions in each free-choice class must be either immediate or timed.

The behavior of timed nets can be described by states and state transitions where each state represents the distribution of (remaining) tokens in places as well as distribution of firing transitions (some transitions can fire several times). The transitions between states can be combined into a graph of reachable states. For timed nets with exponentially distributed firing times, this graph is simply a Markov chain of the stochastic process represented by the timed net. For nets with deterministic firing times, this graph is a semi-Markov (or embedded Markov) chain. In both cases standard techniques developed for Markov process can be used to find stationary properties of states (if the state space is finite) and then derive performance characteristics from these stationary probabilities [20].

### 3. Multithreaded processors

In the multithreaded execution model, a program is a collection of partially ordered threads, and a thread consists of a sequence of instructions which are executed in the conventional von Neumann model. Scheduling of different threads follows the data-driven approach.

A Petri net model of a single processor for the case when context switching is performed for all long-latency (local and remote) memory accesses, is shown in Fig.3.1, which also shows the two switches,  $T_{sinp}$  and  $T_{sout}$ , for incoming and outgoing traffic, respectively. The interconnection of the node with its four neighbors (for the case of a two-dimensional torus-like interconnecting network; the interconnection networks are discussed in greater detail in Section 4) is also shown in Fig.3.1.

The execution of (ready) threads is modeled by transition  $Trun$  with place  $Proc$  representing the (available) processor (if marked) and  $Ready$  – the pool of threads waiting for execution. The initial marking of  $Ready$  represents the average number of threads,  $n_t$ , one of important model parameters. It is assumed that this number does not change in time.

The firing time of  $Trun$  is exponentially distributed (all other firing times are deterministic) and its average value represents the runlength of threads, i.e., the average number of instructions executed before context switching occurs. The runlength,  $\ell_t$ , is another important parameter of the model.

$Mem$  is a free-choice place, with a random choice of either accessing local memory ( $T_{loc}$ ) or remote memory ( $T_{rem}$ ); in the first case, the request is directed to  $Lmem$  where

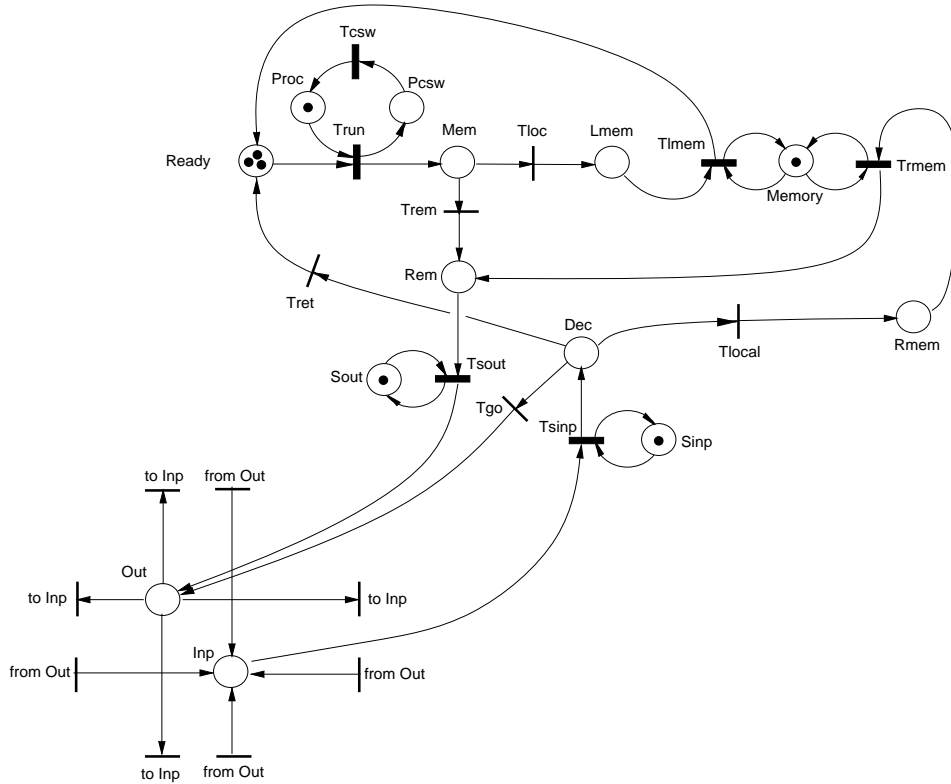


Fig.3.1. Petri net model of a single processor; context switch on all memory accesses.

it waits for availability of *Memory*, and after accessing the memory, the thread returns to the queue of waiting threads, *Ready*. *Memory* is a shared place with two conflicting transitions, *Trmem* (for remote accesses) and *Tlmem* (for local accesses); the resolution of this conflict (if both accesses are waiting) is based on marking-dependent (relative) frequencies determined by the numbers of tokens in *Lmem* and *Rmem*, respectively.

The free-choice probability of *Trem*,  $p_r$  (or of *Tloc*,  $p_l = 1 - p_r$ ), is another basic parameter of the model.

Requests for remote accesses are directed to *Rem*, and then, after a sequential delay (the switch modeled by *Sout* and *Tsout*), forwarded to *Out*, where a random selection is made of one of the four (in this case) adjacent nodes (all nodes are selected with equal probabilities). Similarly, the incoming traffic (memory access requests from remote nodes and responses to remote accesses) are collected from all neighboring nodes in *Inp*, and, after a sequential delay (*Sinp* and *Tsinp*), forwarded to *Dec*. *Dec* is a free-choice place with three transitions sharing it: *Tret*, which represents the satisfied requests reaching their 'home' nodes; *Tgo*, which represents requests as well as responses forwarded to another node (another 'hop'); and *Tlocal*, which represents remote requests accessing the memory at the destination node; these remote requests are queued in *Rmem* and served by *Trmem* when the memory module *Memory* becomes available.

The traffic outgoing from a node (place *Out*) is composed of requests and responses for-

warded to another node (transition  $Tgo$ ), responses to requests from other nodes (transition  $Trmem$ ) and remote memory requests originating in this node (transition  $Trem$ ).

There are six timed transitions in the model shown in Fig.3.1. It is convenient to assume that all firing times are expressed in terms of the ‘processor cycle time’ used as a ‘unit of time’; then the parameters and their typical values are:

| <i>symbol</i> | <i>parameter</i>                               | <i>typical values</i> |
|---------------|--|-----------------------|
| $n_t$         | the (average) number of threads                | 2,...,20              |
| $\ell_t$      | thread runlength                               | 5,10,20               |
| $t_{cs}$      | context switching time                         | 1,10                  |
| $t_m$         | memory cycle time                              | 10                    |
| $t_s$         | switch delay                                   | 5, 10                 |
| $p_\ell, p_r$ | probability of accesses to local/remote memory | 0.1,...,0.9           |

Fig.3.2 shows a processor’s model for the case when the context switching is performed for remote memory accesses only (if the time of context switching is comparable with the access time to local memory, context switching for accesses to local memory is not justified).

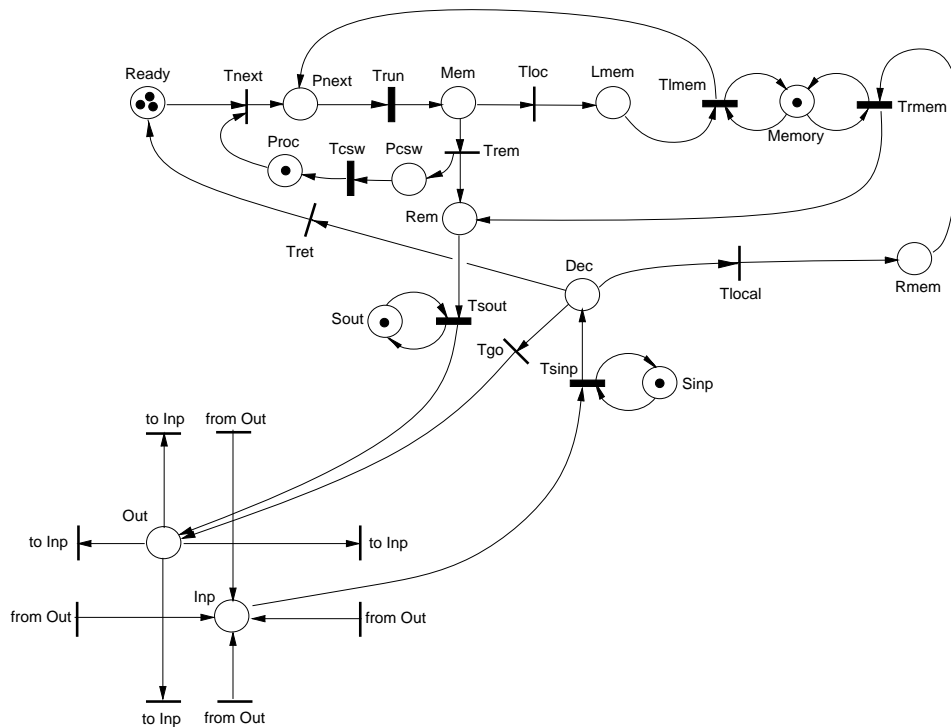


Fig.3.2. Petri net model of a single processor; context switching on remote memory accesses.

Selection of a thread for execution is represented by transition  $T_{next}$ . During accesses to local memory ( $T_{lmem}$ ), the processor is waiting for the completion of the access, after which the execution of the same thread continues (transition  $T_{run}$ ). For accesses to remote memory, transition  $T_{rem}$  initiates context switching by depositing a token in  $P_{cs}$ ;  $T_{cs}$  represents context switching (by its firing time), after which the processor is available for execution of another thread from  $Ready$ , the pool of waiting threads.

## 4. Interconnecting networks

Processors are usually connected by either a torus-like network or a hypercube-type connection. Fig.4.1 shows a 16-node two-dimensional torus network where each node contains a processor (as in Fig.3.1 or Fig.3.2), and all connections between pairs of nodes are two-directional.

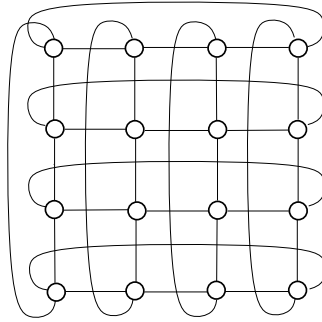


Fig.4.1. Torus-like network for a 16-processor system.

The free-choice probability of *Tgo* can be determined from the ‘traffic patterns’ in the interconnecting network. Assuming that all (remote) memory requests are uniformly distributed over the nodes, the average number of hops can be calculated from the lengths of the (shortest) paths between the nodes. For a 16-processor system, for each node there are 15 remote nodes, 4 of which are at the distance of 1 hop, 6 at the distance of 2 hops, 4 at the distance of 3 hops, and 1 node at the distance of 4 hops, as sketched in Fig.4.2, where “0” denotes the ‘reference node’. The average distance is thus:

$$\frac{4 * 1 + 6 * 2 + 4 * 3 + 1 * 4}{15} \approx 2 \text{ hops}$$

If a simple geometric distribution of the number of hops is assumed, the average value for this distribution is:

$$\frac{1}{1 - p} = 2 \text{ hops}$$

so  $p$ , the probability that a request is forwarded to a next node (i.e., the free-choice probability of *Tgo*) is  $p = 0.5$ . The model of geometric distribution is very simple (as shown in Fig.3.1 and Fig.3.2) but this distribution does not restrict forwarded requests to 4 hops; consequently, there is a small probability that some requests can be forwarded beyond the limit of 4 hops. Also, the probability density function of the number of hops for the geometric distribution does not represent the real distribution very accurately; it appears that the real distribution can be modeled very conveniently by colored nets, as described further.

In order to realistically represent the two-directional traffic in the interconnecting network, i.e., streams of requests (for remote accesses) and streams of responses (to these requests) that ‘return’ to the original nodes, the model uses two colors of tokens in the interconnecting network, “F” for forward moving requests and “B” for backward moving responses. The tokens (requests) generated by *Trem* are thus of color “F”, while those

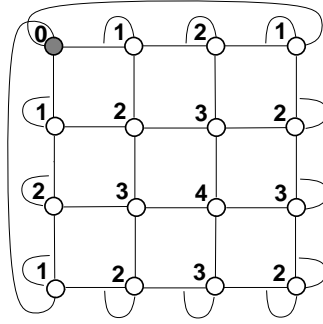


Fig.4.2. The minimal distances between nodes.

generated by  $Trmem$  are of color “B”. The stream of colored tokens reaching  $Dec$  is separated so that only tokens of color “F” enable  $Tlocal$  and only tokens of color “B” enable  $Tret$  ( $Tgo$  is enabled by tokens of both colors). Consequently, the free-choice probability of  $Tlocal$  is 0.5 for color “F” (and 0 for color “B”), and that of  $Tret$  is 0.5 for color “B” (and 0 for color “F”); free-choice probability of  $Tgo$  is 0.5 for both colors, “F” and “B”.

There is one more consequence of using colored tokens in the interconnecting network: the switches represented by  $Tsinp$  and  $Tsout$  have different occurrences for different colors of tokens, and these occurrences are in conflict because of sharing common places  $Sinp$  and  $Sout$ , respectively. The solution used to resolve these conflicts is based on marking-dependent (relative) frequencies, determined by the numbers of tokens in  $Inp$  and  $Rem$ , respectively (similarly to the conflict resolution of  $Tlmem$  and  $Trmem$ ).

If the two occurrences of transitions  $Tsinp$  and  $Tsout$  are denoted by “F” and “B”, the following tables describe the arc function  $a$  for arcs incident with the two switches (“U” denotes the ‘universal’ color, when only one color is needed):

| $Tsinp$ | $Inp$ | $Sinp$ | $Dec$ | $Sinp$ | $Tsout$ | $Rem$ | $Sout$ | $Out$ | $Sout$ |
|---------|-------|--------|-------|--------|---------|-------|--------|-------|--------|
| F       | F:1   | U:1    | F:1   | U:1    | F       | F:1   | U:1    | F:1   | U:1    |
| B       | B:1   | U:1    | B:1   | U:1    | B       | B:1   | U:1    | B:1   | U:1    |

where “ $X : n$ ” denotes a function  $a : C \rightarrow \{0, 1, \dots\}$  such that  $a(X) = n$ , and for all other colors  $Y \in C - \{X\}$ ,  $a(Y) = 0$ .

The geometric distribution of the number of hops in the network (Fig.3.1 and Fig.3.2) can be refined to a more accurate, but also more complicated representation. Since the exact probabilities of making another hop are known (Fig.4.2), the real values of these probabilities can be used in the model instead of the geometric distribution; Fig.4.3(a) shows the real probability density function of the number of hops for remote memory accesses in a 16-processor system while Fig.4.3(b) shows the density function of the geometric distribution with the same average value as the distribution in Fig.4.3.(a) (this average value is 2).

In order to model the distribution of Fig.4.3(a), four “forward” colors are needed (say, F1, F2, F3 and F4) as there are four (discrete) values in this distribution, and four “backward colors” (for example, B1, B2, B3, B4). Transition  $Trem$  generates a token of color F1 (for the first hop); if a token of color F1 continue for another hop (free-choice transition  $Tgo$ ), its color is changed from F1 to F2 (by occurrence F1 of  $Tgo$ ); similarly, if a token of color F2 continues for another hop, its color is changed to F3 (by occurrence F2 of  $Tgo$ ),

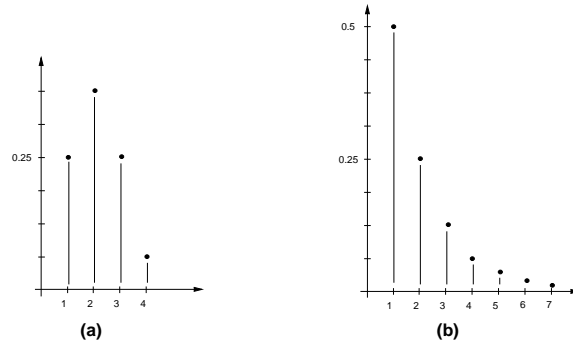


Fig.4.3. The distributions of the number of hops; 16 processors.

etc. If a token of color  $F_i$  accesses memory (through  $T_{local}$ ,  $R_{mem}$  and  $T_{mem}$ ), its color changes from  $F_i$  to  $B_i$ , and then  $T_{go}$  forwards all  $B_i$  tokens “decreasing” their colors from  $B_4$  to  $B_3$ , from  $B_3$  to  $B_2$ , and from  $B_2$  to  $B_1$ . Tokens of color  $B_1$  enable only  $T_{ret}$ , and, after firing this transition, are returned to  $Ready$  (with the ‘universal’ color  $U$ ).

The free-choice transitions  $T_{local}$  and  $T_{go}$  will have four occurrences (say  $F_1$ ,  $F_2$ ,  $F_3$ , and  $F_4$ ) with the following free-choice probabilities:

| <i>occurrence</i> | $T_{local}$ | $T_{go}$ |
|-------------------|-------------|----------|
| $F_1$             | 4/15        | 11/15    |
| $F_2$             | 6/11        | 5/11     |
| $F_3$             | 4/5         | 1/5      |
| $F_4$             | 1.00        | 0        |

These probabilities correspond to the distribution in Fig.4.3(a), “rescaled” for consecutive hops; after the first hop, the probability of accessing memory is 4/15 (there are 4 nodes in distance of 1 hop in Fig.4.2); the probability of accessing memory after 2 hops is 6/15 (Fig.4.3(a)), or 11/15\*6/11 (11/15 to continue after the first hop, and 6/11 to select memory accessing after the second hop), and so on. After 4 hops, in a 16-processor system there is no choice; the requests have to access the memory.

The free-choice probabilities of  $T_{go}$  and  $T_{ret}$  are:

| <i>occurrence</i> | $T_{go}$ | $T_{ret}$ |
|-------------------|----------|-----------|
| $B_1$             | 0        | 1.00      |
| $B_2$             | 1.00     | 0         |
| $B_3$             | 1.00     | 0         |
| $B_4$             | 1.00     | 0         |

The second major use of the colors of tokens is to “fold” all subnets representing the processors (or nodes). Such a “folded” model is shown in Fig.4.4, with 16 colors,  $N_{ij}$ ,  $i = 1, 2, 3, 4$ , and  $j = 1, 2, 3, 4$ , representing the original array of  $4 \times 4$  processors.

Transition  $T_{net}$  has  $16 \times 4$  occurrences, which are denoted  $N_{ijk}$ ,  $i = 1, 2, 3, 4$ ,  $j = 1, 2, 3, 4$ ,  $k = N, E, S, W$ . These occurrences correspond to the 4 connections (N – north, E – east, etc.) of each of the 16 nodes in the original network. All these occurrences form free-choice

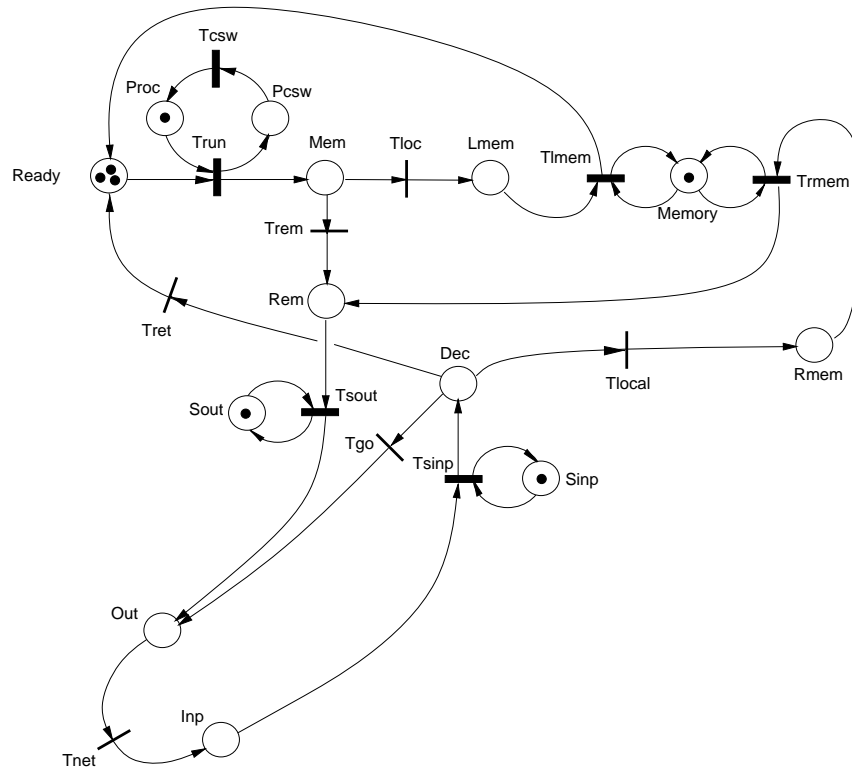


Fig.4.4. Colored net model of a multiprocessor system; context switching on all memory accesses.

classes in groups of 4, for different values of  $k$ ; if the traffic in the network is uniformly distributed, all free-choice probabilities of these occurrences are equal. A partial definition of the arc functions, showing the transformation of token colors, is:

| $T_{net}$ | $Out$ | $Inp$ |
|-----------|-------|-------|
| N11N      | N11:1 | N41:1 |
| N11E      | N11:1 | N12:1 |
| N11S      | N11:1 | N21:1 |
| N11W      | N11:1 | N14:1 |
| N12N      | N12:1 | N42:1 |
| N12E      | N12:1 | N13:1 |
| N12S      | N12:1 | N22:1 |
| N12W      | N12:1 | N11:1 |
| ....      | ...   | ...   |

Since  $T_{net}$  is the only transition with such a large number of occurrences, it may be more reasonable to replace  $T_{net}$  by four free-choice transitions, representing the choice of a neighbor N, W, S or E, as shown in Fig.4.5, each transition with 16 occurrences corresponding to 16 nodes. A partial definition of the arc functions for this solution is:

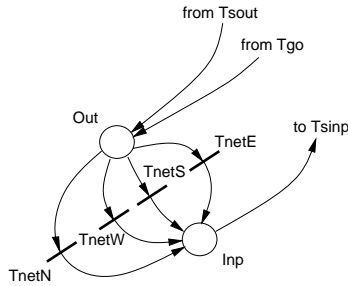


Fig.4.5. Refined colored model of the interconnecting network.

| <i>occurrence</i> | <i>Out</i> | <i>TnetN</i> | <i>TnetE</i> | <i>TnetS</i> | <i>TnetW</i> |
|-------------------|------------|--------------|--------------|--------------|--------------|
| N11               | N11:1      | N41:1        | N12:1        | N21:1        | N14:1        |
| N12               | N12:1      | N42:1        | N13:1        | N22:1        | N11:1        |
| N13               | N13:1      | N43:1        | N14:1        | N23:1        | N12:1        |
| N14               | N14:1      | N44:1        | N11:1        | N24:1        | N13:1        |
| N21               | N21:1      | N11:1        | N22:1        | N31:1        | N24:1        |
| N22               | N22:1      | N12:1        | N23:1        | N32:1        | N21:1        |
| ...               | ...        | ...          | ...          | ...          | ...          |
| N44               | N44:1      | N34:1        | N41:1        | N14:1        | N43:1        |

For hypercube-type interconnecting networks, the main difference is in the average number of hops that a request must make to reach its destination (and then return to the home node). In a net with  $N = 2^n$  nodes (or processors) spanning along  $n$  dimensions, each node is connected with  $n$  other nodes. An example of a 4-cube network (for  $2^4 = 16$  nodes) is shown in Fig.4.6.

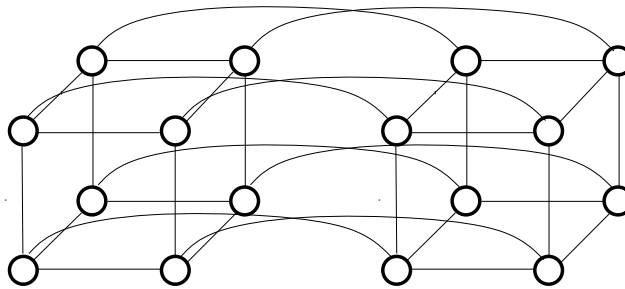


Fig.4.6. 4-cube interconnecting network obtained from two 3-cubes.

For example, in a 64-processor system, each node is connected with 6 other nodes ( $64 = 2^6$ ), and the average number of hops is equal to 3.1. In a two-dimensional torus-like interconnecting network with 64 nodes, each node is connected with 4 neighbors, and the average number of hops is equal to 4. Consequently, in a torus-line network there will be more traffic and greater service demands for switches than in a hypercube-type network with the same number of nodes and the same parameters. The delays of remote memory accesses will thus be longer, and the processors utilization will be lower for the torus-like interconnecting network.

It should be observed that for a 16-processor system the two types of interconnecting networks are equivalent; each processor is connected to 4 other processors, the average numbers of hops are the same, so the throughputs are the same, and processor utilizations also.

## 6. Performance results

All results presented in this section have been obtained by simulation of the corresponding net models [21].

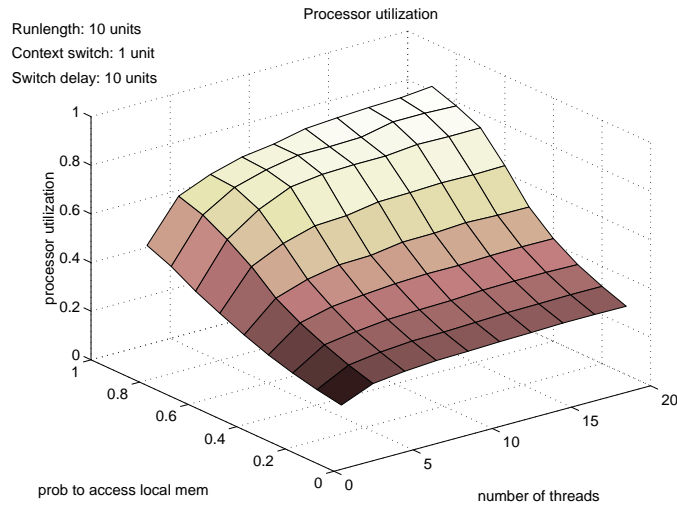


Fig.5.1. Processor utilization ( $t_s = 10, t_{cs} = 1$ ).

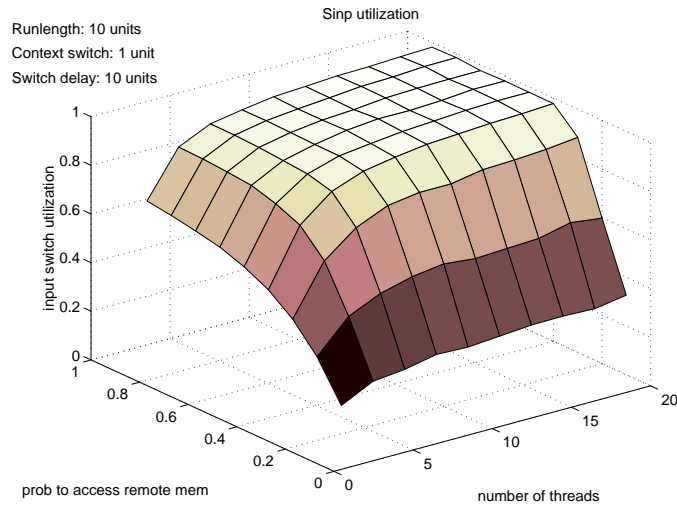


Fig.5.2. Input switch utilization ( $t_s = 10, t_{cs} = 1$ ).

Fig.5.1 shows the utilization of the processor as a function of the (average) number of threads,  $n_t$ , and the probability of accesses to local memory,  $p_\ell$ ; the remaining parameters

are: context switching time  $t_{cs} = 1$ , runlength  $\ell_t = 10$ , memory cycle  $t_m = 10$ , and switch delay  $t_s = 10$ .

With the exception of the region corresponding to  $p_\ell$  close to one (and  $p_r$  close to zero), the utilization tends to “saturate” very quickly at the levels significantly below the maximum value. This is a characteristic indication that some other (than processor) component limits the performance of the system, and becomes the bottleneck [6]. Indeed, it appears that the input switch  $T_{sinp}$  is too “slow”, and is utilized is almost 100 %, as shown in Fig.5.2 (note that probability of remote accesses,  $p_r$ , is used in Fig.5.2 rather than  $p_\ell$ , so the “front” of Fig.5.2 corresponds to the “back” of Fig.5.1).

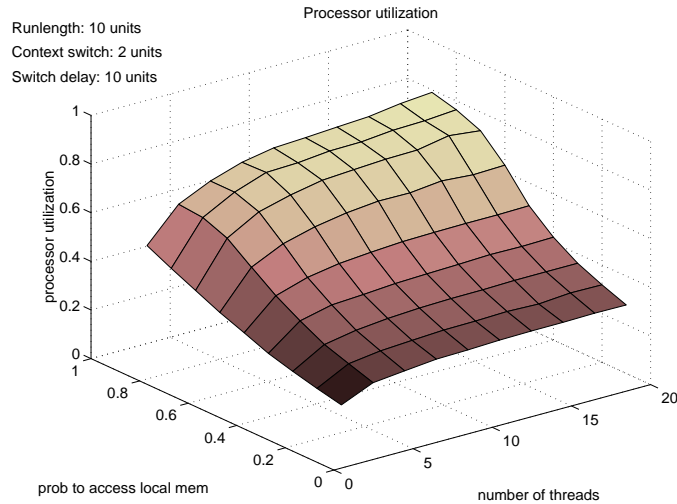


Fig.5.3. Processor utilization ( $t_s = 10, t_{CS} = 2$ ).

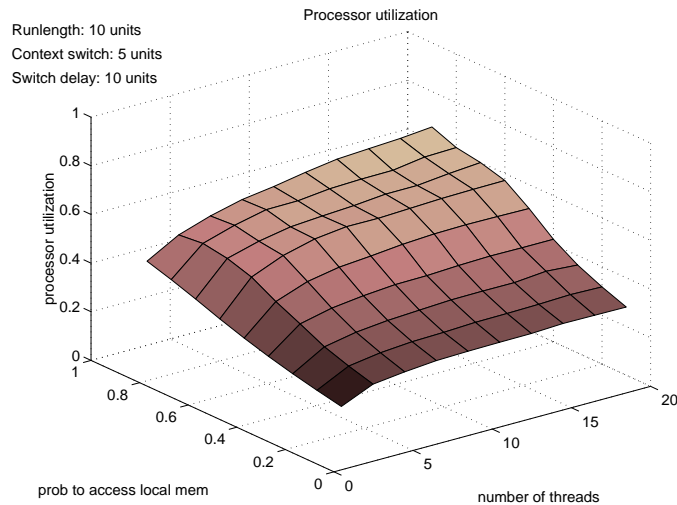


Fig.5.4. Processor utilization ( $t_s = 10, t_{CS} = 5$ ).

The influence of the context switching time can be observed in Fig.5.3 and Fig.5.4, which show the processor utilization as a function of the number of threads and the probability of accesses to local memory, as in Fig.5.1, but for the context switching times equal to 2 and 5, respectively; the maximum values of the processor utilization clearly decrease when

the context switching time increases (for the “saturated” region they remain practically the same).

It can be shown that for the probability  $p_\ell$  close to 1, for large numbers of threads, and for  $l_t \geq t_m$ , the upper bound on the value of processor utilization is equal to  $l_t / (l_t + t_{cs})$ ; for Fig.5.3, this bound is equal to 0.83, and for Fig.5.4 is equal to 0.67 (for Fig.5.1 the bound is equal to 0.91).

Fig.5.5 shows the utilization of the input switch for a smaller values of the switch delay,  $t_s = 5$ , with all other parameters are as in Fig.5.1. It can be observed that the saturated region is reduced to less than one half of that shown in Fig.5.2.

The consequence of the “relaxed” utilization of the input switch is that the utilization of the processor (and the performance of the whole system) improves, as shown in Fig.5.6 (again, the “front” part of Fig.5.6 corresponds to the “back” part of Fig.5.5, as the probability of remote accesses is used in Fig.5.5 rather than local ones used in Fig.5.6).

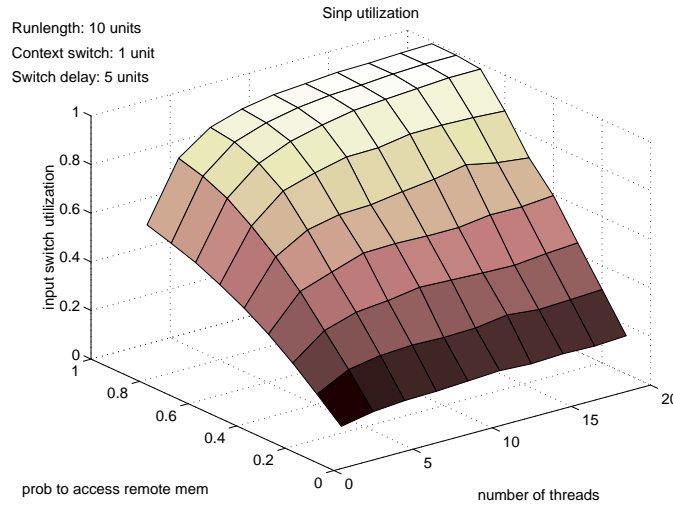


Fig.5.5. Input switch utilization ( $t_s = 5, t_{cs} = 1$ ).

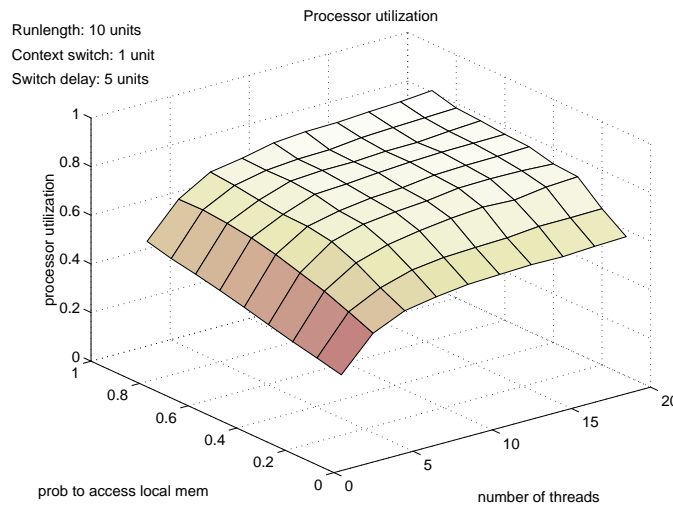


Fig.5.6. Processor utilization ( $t_s = 5, t_{cs} = 1$ ).

Further reduction of the switch delay results in even better performance of the system. Many other performance characteristics of multithreaded multiprocessor systems can be obtained in a similar way. Some other results are shown in [8].

## 5. Concluding remarks

It has been shown that a simple colored Petri net model of a fairly complex system can be derived in a systematic way, taking advantage of similarities between components of the system. The simple structure of the model does not mean, in general, that analysis is as straightforward as the model might imply.

The presented model (Fig.4.4) does not actually depend upon the number of processors in the original system, although some model parameters (for example, free-choice probabilities) must be adjusted if the number of processors changes. The whole interconnection network is represented by a single transition in Fig.4.4 (or a single class of free-choice transitions in Fig.4.5), and this representation is valid for torus-like and hypercube-type connections as well as many other types of interconnecting networks (the number of free-choice transitions in Fig.4.5 changes with the number of node connections).

The colored model captures very nicely the most important features of the system, illustrates the flow of requests, and represents the decision and conflicts which must be resolved by some additional criteria. The simplicity and clarity of the model seems to be one of major advantages of the proposed approach. Many less important details are hidden in the descriptions of model elements.

Even very simple performance analyses indicate that the system's behavior is quite sensitive to the values, and in fact relations between the values of model parameters [22]. If service demands to different components (such as processor, memory or switches) are not balanced, the performance of the whole system is limited by the most demanded component (called the bottleneck); any improvement in the performance of the system can be achieved only if the performance of this critical component is improved.

A throughput-preserving approach to model simplification has been proposed in [8]. Taking advantage of symmetries in the multiprocessor system, the model is substantially reduced by removing most of the nodes and by adjusting the throughputs of the remaining switches to the same values as in the original system; since the relative values of these throughputs can easily be derived from the steady-state conditions, the adjustments can be made without the knowledge of the actual values. In effect, the performance of the whole system can be obtained by evaluating a much simpler model. More research is needed in this direction to understand the limitations of such an approach.

## References

- [1] Agrawal, A., Lim, B-H., Kranz, D., Kubiawicz, J., "April: a processor architecture for multiprocessing"; Proc. 17-th Annual Int. Symp. on Computer Architecture, pp.104-114, 1990.
- [2] Agrawal, A., "Limits on interconnection network performance"; IEEE Trans. on Parallel and Distributed Systems, vol.2, no.4, pp.398-412, 1991.

- [3] Alverson, R., Callahan, D., Cummings, D., Koblenz, B., Posterfield, A., Smith, B., "The Tera computer system"; Proc. Int. Conf. on Supercomputing, Amsterdam, The Netherlands, pp.1–6, 1990.
- [4] Boothe, B. and Ranade, A., "Improved multithreading techniques for hiding communication latency in multiprocessors"; Proc. 19-th Annual Int. Symp. on Computer Architecture, pp.214–223, 1992.
- [5] Culler, D.E., et al., "Fine-grain parallelism with minimal hardware support: a compiler controlled threaded abstract machine"; Proc. 4-th Int. Conf. on Architectural Support of Programming Languages and Operating Systems, Santa Clara, CA, pp.164–175, 1991.
- [6] Ferrari, D., "Computer systems performance evaluation"; Prentice–Hall 1978.
- [7] Govindarajan, R., Nemawarkar, S.S., LeNir, P., "Design and performance evaluation of a multithreaded architecture"; Proc. First IEEE Symp. on High–Performance Computer Architecture, Raleigh, NC, pp.298–307, 1995.
- [8] Govindarajan, R., Suciu, F., Zuberek, W.M., "Timed Petri net models of multithreaded multiprocessor architectures"; Proc. 7-th Int. Workshop on Petri Nets and Performance Models, St. Malo, France, pp.153–162, 1997.
- [9] Hirata, H., et al., "An elementary processor architecture with simultaneous instruction issuing from multiple threads"; Proc. 19-th Annual Int. Symp. on Computer Architecture, pp.136–145, 1992.
- [10] Holliday, M.A., Vernon, M.K., "Exact performance estimates for multiprocessor memory and bus interference"; IEEE Trans. on Computers, vol.36, no.1, pp.76–85, 1987.
- [11] Jensen, K., "Coloured Petri nets"; in: "Advanced Course on Petri Nets 1986" (Lecture Notes in Computer Science 254), Rozenberg, G. (ed.), pp.248–299, Springer Verlag 1987.
- [12] Keckler, S.W., Dally, W.J., "Processor coupling: integration of compile-time and runtime scheduling for parallelism"; Proc. 19-th Annual Int. Symp. on Computer Architecture, pp.202–213, 1992.
- [13] King, P.J.B., "Computer and communication systems performance modelling"; Prentice–Hall 1990.
- [14] Moore, S.W., "Multithreaded processor design"; Kluwer Academic 1996.
- [15] Murata, T., "Petri nets: properties, analysis and applications"; Proceedings of IEEE, vol.77, no.4, pp.541–580, 1989.
- [16] Reisig, W., "Petri nets - an introduction" (EATCS Monographs on Theoretical Computer Science 4); Springer Verlag 1985.
- [17] Saavedra–Bareera, R.H., Culler, D.E., von Eicken, T., "Analysis of multithreaded architectures for parallel computing"; Proc. 2-nd Annual Symp. on Parallel Algorithms and Architectures, Crete, Greece, 1990.
- [18] Smith, B.J., "Architecture and applications of the HEP multiprocessor computer System"; Proc. SPIE – Real-Time Signal Processing IV, vol. 298, pp. 241–248, San Diego, CA, 1981.

- [19] Weber, W.D., Gupta, A., “Exploring the benefits of multiple contexts in a multiprocessor architecture: preliminary results”; Proc. 16-th Annual Int. Symp. on Computer Architecture, pp.273–280, 1989.
- [20] Zuberek, W.M., “Timed Petri nets – definitions, properties and applications”; Microelectronics and Reliability, vol.31, no.4, pp.627–644, 1991 (available through anonymous ftp at [ftp.cs.mun.ca/pub/publications/91-Mar.ps.Z](ftp://ftp.cs.mun.ca/pub/publications/91-Mar.ps.Z)).
- [21] Zuberek, W.M., “Modeling using timed Petri nets – event-driven simulation”; Technical Report #9602, Department of Computer Science, Memorial Univ. of Newfoundland, St. John’s, Canada A1B 3X5, 1996 (available through anonymous ftp at [ftp.cs.mun.ca/pub/techreports/tr-9602.ps.Z](ftp://ftp.cs.mun.ca/pub/techreports/tr-9602.ps.Z)).
- [22] Zuberek, W.M., Govindarajan, R. “Performance balancing in multithreaded multiprocessor architectures”; Proc. 4-th Australasian Conf. on Parallel and Real-Time Systems (PART’97), Newcastle, Australia, pp.15–26, 1997.