7-th Int. Workshop on Petri Nets and Performance Models (PNPM'97), Saint Malo, France, June 2-6, 1997, pp.153-162.
 (DOI 10.1109/PNPM.1997.595546).

Timed Petri Net Models of Multithreaded Multiprocessor Architectures

R. Govindarajan
Supercomputer Education and Research Center
Indian Institute of Science
Bangalore 560 012, India

F. Suciu and W.M. Zuberek
Department of Computer Science
Memorial University
St.John's, Canada A1B 3X5

Abstract

Multithreaded distributed–memory multiprocessor architectures are composed of a number of (multithreaded) processors, each with its memory, and an interconnection network. The long memory latencies and unpredictable 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 architectures. However, accurate net models of multithreaded multiprocessors become quite complicated, so their analysis can be a nontrivial task.

This paper describes a timed colored Petri net model of a multithreaded multiprocessor architecture, and presents some results obtained by simulation of this model. A simplified approach to modeling such architectures is also proposed.

1. Introduction

Current microprocessors employ various techniques to increase parallelism and processor utilization. For example, modern superscalar architectures, such as DEC's Alpha [9], PowerPC [22] or SUN's UltraSPARC [23], issue up to four instructions per cycle. Multiple instruction issue increases performance, but is limited by instruction dependencies and long–latency operations within the single executing thread.

Multithreading provides a means of tolerating long, unpredictable communication latency and synchronization delays. Its basic idea is quite straightforward. In a traditional architecture, when a processor accesses a location in memory, it waits for the result, possibly after executing a few instructions that are independent of the memory operation. In a large multiprocessor, this wait may involve more than one hundred cycles [8] since the memory request may need to be transmitted across the communication network to a remote memory module, serviced, and then the value returned. As a result, the utilization of the processor tends to be low. Alternatively, if the processor maintains multiple threads of execution, instead of waiting the processor can switch to another thread and continue doing useful work. With multithreading, the processor utilization is largely independent of the latency in completing remote accesses.

Several multithreaded architectures have recently been proposed in the literature [1, 6, 8, 10, 11, 15]. Analyzing the performance of such 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 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 [11, 15, 10], analytical models — using either queuing networks or Petri Nets [2, 5, 17, 18, 14, 20], or using trace-driven simulation [24].

Petri nets have been proposed as a simple and convenient formalism for modeling systems that exhibit parallel and concurrent activities [19, 16]. 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 [25], 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 [13], tokens are associated with attributes (called colors), so different activities can be assigned to tokens of different types.

The behavior of timed net models can be determined by systematic analysis of all possible states (the so called reachability analysis), or by simulation.

Since timed nets are discrete–event systems, typical discrete–event simulation techniques, and in particular, event–driven simulation, can be used for analysis of the behavior of net models.

This paper describes a multiprocessor multithreaded architecture in a greater detail, outlines its timed Petri net model, and presents some performance results obtained by simulation of this model. An alternative modeling approach, taking into account the symmetries of the model, is also presented and its results are compared with those obtained for the original model.

2. Multithreaded Multiprocessor Architecture

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.

Switching from one thread to another is performed according to one of the following policies [7]:

- Switching on every instruction: the processor switches from one thread to another every cycle. In other words, it interleaves the instructions from different threads on a cycle-by-cycle basis [21].
- 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 [6].
- Switching on remote load: processor switches to another thread only when current thread encounters a remote access [1].

A model where the context switching takes place on every load is assumed in this paper (concluding remarks contain a brief discussion of the other models). That is, if the executed instruction issues an operation for accessing either a local or a remote memory location, the execution of the current thread *suspends*, the thread changes its state to waiting, and another *ready* thread is selected for execution. When the long–latency operation for which a thread was waiting is satisfied, the thread becomes *ready* and joins the pool of ready threads waiting for execution. The thread that is being executed is said to be *executing*. Thus in a model of multithreaded execution, a thread undergoes state transitions shown in Fig.2.1.

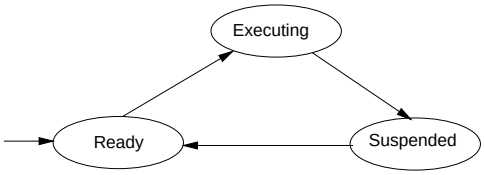

Fig.2.1. State transitions of a thread.

The average number of instructions executed by a thread before issuing a load operation (and switching to another thread) is called *thread runlength*, and is one of the important parameters that affect the performance of multithreaded architectures. When a thread gets suspended, the processor saves the context of the current thread and switches to another thread from the *Ready Pool* (refer to Fig.2.2). It is assumed that the context switching overhead is small in comparison to the thread execution time.

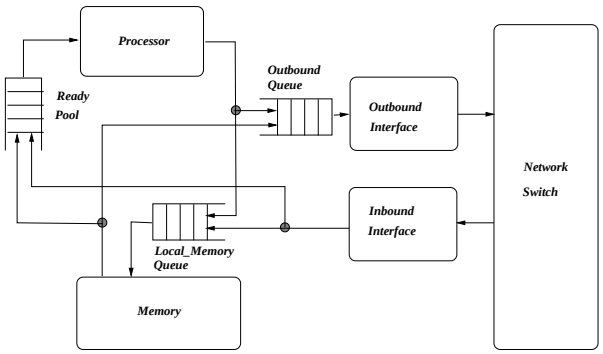

Fig.2.2. Architecture of single multithreaded processor.

The memory operation request of a suspended thread is sent either to the *Local Memory Queue* or to the *Outbound Queue* depending on whether the access is local or remote. Operation requests from other processors to this memory also arrive at the *Local Memory Queue* through the *Network Switch* and *Inbound Interface* (Fig.2.2). For each request serviced by this *Memory* unit, a message is sent either to the *Outbound Queue* or to the local synchronization unit (not shown in Fig.2.2) depending on whether the operation request has come from a remote node or this processor. Thus remote memory requests (generated by this processor) and responses to remote processors (generated by the local memory) are queued in the *Outbound Queue*. These are serviced by the *Outbound Interface* and sent to remote nodes through the *Network Switch*. A message sent to the synchronization unit (either from the local memory or a remote mem-

ory) causes the appropriate thread to become *ready* and join the *Ready Pool*.

A constant parallelism model, under which each processor owns n parallel threads which can be executed only in that processor, is assumed. Threads interact through shared memory locations. The shared memory model assumed in this work is distributed shared memory. That is, each processor contains some part of the shared address space. As mentioned earlier, all accesses to shared space are considered to be long–latency operations whether the shared address space is available locally or in a remote processor. The memory access pattern is assumed to be uniformly distributed over all the processors in the system. The probability that a long–latency operation is an access to a remote processor is another important parameter, which characterizes (in some sense) the locality of memory references.

The nodes of a multithreaded multiprocessor architecture are connected by a two-dimensional torus interconnection network. Fig.2.3 sketches a 16–processor system connected by a 4×4 torus network. 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. The delay for a message is proportional to the number of hops between the source and destination, and it also depends upon the traffic in the chosen path. The interface between the network switch and processor node is through a pair of outbound and inbound network interfaces, as shown in Fig.2.2.

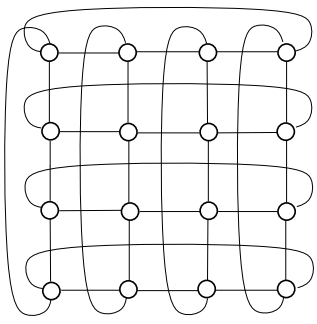

Fig.2.3. 16–processor system.

The delay of a single hop (the switch delay) is another parameter in performance studies of multiprocessor architectures.

3. Petri Net Models

This section first briefly recalls basic concepts of timed Petri nets, and then describes the timed net model of the multithreaded architecture.

3.1. Basic concepts of timed Petri nets

The basic (place/transition) Petri net 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, $m_0 : P \to \{0, 1, ...\}$. Usually the set of places connected by (directed) arcs to a transition is called the input set of a transition, and the set of placed 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. It is assumed that selection of a transition for firing in a free–choice class of transitions is a random process which can be described by (free–choice) probabilities assigned to transitions in each free–choice class. Moreover, it is usually assumed that the random choices in different free–choice classes are independent one from another.

A shared place which is not free–choice, is a conflict place. The class of enabled transitions sharing a conflict place depends upon the marking function, so the probabilities of firing conflicting transitions must be determined in a dynamic (i.e., marking–dependent) way. A simple but usually satisfactory solution is to use relative frequencies of transition firings assigned to conflicting transitions [12]; the probability of firing an enabled transition is then determined by the ratio of transition's (relative) frequency to the sum of (relative) frequencies of all enabled transitions in a conflict class. Another generalization is to make such relative frequencies (and probabilities of firings) 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.

In basic nets the tokens are indistinguishable, so their distribution can be described by a simple marking function $m : P \to \{0, 1, ...\}$. In colored Petri nets [13], tokens have attributes called colors. Token colors can be modified by (firing) transitions and also a transition can have several different occurrences (or variants) of its firings.

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. 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.

The firing times of some transitions may be equal to zero, which means that the firings are instantaneous; all such transitions 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 fire first the (enabled) immediate transitions, 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.

3.2. Petri net model of the multithreaded architecture

Each node in the torus network shown in Fig.2.3 contains a processor. A Petri net model of a node is shown in Fig.3.1 which also shows the two switches (one for incoming and one for outgoing traffic) and the interconnection of the node with its four neighbors.

Execution of (ready) threads is modeled by $Trun$ with *Proc* representing the (available) processor and *Ready* – the queue of threads waiting for execution. *Mem* is a free–choice place, with a random choice representing a request for accessing either local memory ($Tloc$) or remote memory ($Trem$); in the first case, the request is directed to *Lmem* where it waits for the *Memory*, and after accessing the memory, the thread returns to the queue *Ready* of waiting threads. *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 $Trem$ (or of $Tloc$) is one of model parameters.

Requests for remote accesses are directed to *Out*, and then, after a sequential delay (the switch modeled by *Sout* and $Tsout$), to Net, where they are distributed to all four adjacent nodes with equal probabilities. Similarly, the incoming messages (memory access requests from remote nodes and responses to remote accesses) are collected from all four neighboring nodes in *Inp*, and, after a sequential delay (*Sinp* and $Tsinp$), enter *Dec*. *Dec* is a free–choice place with three transitions sharing it: $Tret$, which represents satisfied requests returning to 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 this node; remote requests are queued in *Rmem* and served by $Trmem$ when the memory module becomes available.

The traffic outgoing from a node (place Net) is composed of requests and responses forwarded to another node (transition Tgo), responses to requests from other nodes (transition $Trmem$) and remote memory requests originating in this node (transition $Trem$). 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.3.2, where "0" denotes the 'reference node'. The average distance is thus:

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

Since the distribution of the distance of remote requests, as modeled in Fig.3.1, corresponds to the geometric distribution, the average value for this distribution is:

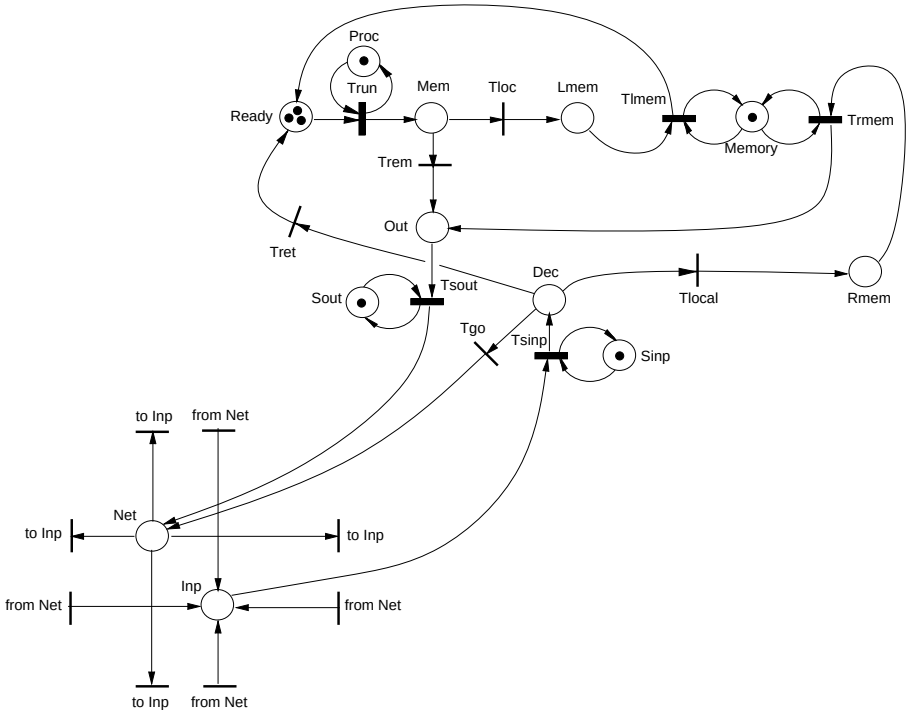

Fig.3.1. Petri net model of a single node.

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

so p, the probability that a request is forwarded to a next node (i.e., the free–choice probability of Tgo) is $p = 0.5$. It should be observed that this simple model does not restrict forwarded requests to 4 hops; consequently, there is a small probability (of the order of 0.5^4) of requests forwarded beyond the limit of 4 hops.

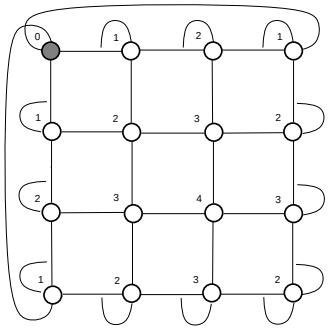

Fig.3.2. The minimal distance between nodes.

Accurate representation of the behavior of the interconnecting network is quite important for the performance of the whole system. Since the traffic is 'two–directional', i.e., there are streams of requests (for remote accesses) and also streams of responses (to these requests) that 'return' to the original nodes, the total traffic in the switches of the interconnecting network must take into account both these streams. In order to separate these two streams, the model uses two colors of tokens in the interconnecting network, "F" for forward moving requests and "B" for backward moving responses. The requests generated by $Trem$ are thus of color "F", while those by $Trmem$ are of color "B". Moreover, 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 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 Out, respectively (similarly to the conflict resolution of $Tlmem$ and $Trmem$).

There are five 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 'cycle time' used as a 'unit of time'.

The memory cycle time ($Tlmem$ and $Trmem$) is assumed to be 10 units of time.

The execution time of a thread (or the runlength), represented by the firing time of $Trun$, which is one of the main parameters of the performance study, is assumed to be exponentially distributed; typical values of the runlength are 5, 10 and 20 units of time (in fact, a geometric distribution with the same average value would be a more realistic representation of the runlength, but the model with exponential distribution is simpler and more efficient for simulation, providing results which are practically the same; the differences between results for these two distributions are less than 1 percent).

It should be observed that the context switching time is not explicitly represented in the model; it is assumed that this time is included in the firing time of $Trun$ (explicit representation of context switching, with a typical value of 1 time unit, does not affect the performance results in a significant way; the differences between exact representation of context switching and simplified one, as in Fig.3.1, are less than 1 percent).

The switch delay time (represented by the firing times of $Tsinp$ and $Tsout$) is another model parameter with typical values of 5 and 10 units of time.

The initial marking function assigns a single token to $Proc$, $Memory$, $Sinp$ and $Sout$, and a number of tokens to $Ready$. The initial marking of $Ready$ is equal to the (average) number of threads; it is another important parameter of the model, and its typical values are between 2 and 20.

The model parameters are thus (m_0 denotes the initial marking, f the firing times, and c the free–choice probabilities of transitions):

Parameter	Model
Number of threads	$m_0(Ready)$
Average runlength of a thread	$f(Trun)$
Probability of local access	$c(Tloc)$
Switch delay	$f(Tsinp)$, $f(Tsout)$

4. Simulation Results

Event–driven simulation [26] was used to obtain performance characteristics for the timed net model of the multithreaded multiprocessor architecture with different combinations of the values of modeling parameters. Some of these results are presented in Fig.4.1 to Fig.4.4. Fig.4.1 shows the utilization of the processor (at each node) as a function of the number of threads and the probability of long–latency accesses to the local memory; the remaining two parameters are constant ($f(Trun) = 10$, $f(Tsinp) = f(Tsout) = 10$). The results confirm the rather straightforward prediction that the utilization grows with the number of threads (the probability that the processor is idle becomes smaller) and with the probability of local long–latency accesses (because the threads remain *suspended* for shorter periods of time).

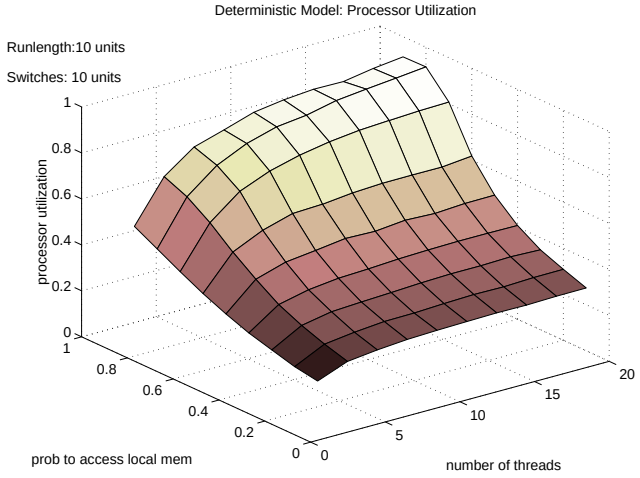

Fig.4.1. Processor utilization.

Fig.4.2 shows the utilization of the memory for remote accesses as a function of the number of threads and the probability of remote memory accesses (it should be noted that in Fig.4.2 the probability of remote accesses is used rather than local accesses).

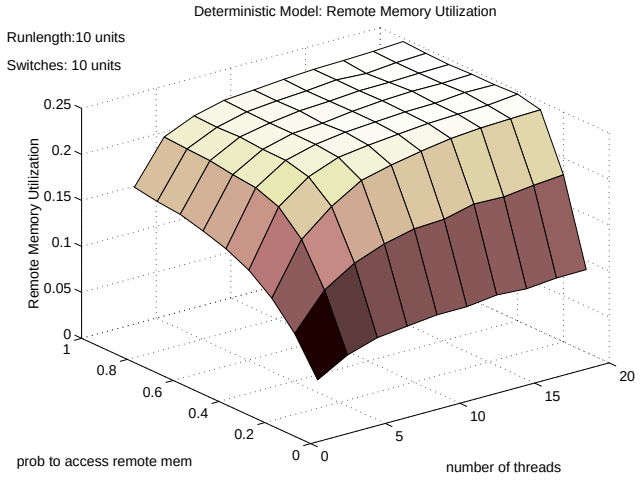

Fig.4.2. Memory utilization for remote accesses.

The results show a large 'saturation' region corresponding to larger number of threads and smaller

probabilities of local memory accesses (i.e., greater probabilities of remote accesses). This saturation, at the level of utilization equal to 0.25, is due to the input switches $Tsinp$ which become the bottleneck of the system. It can be observed that the throughput of transitions $Tsinp$ in the 16–processor system is four times greater than the throughput of $Trmem$ (the free–choice probability of $Tlocal$ is 0.5 for color "F", that of $Tret$ is 0.5 for color "B", and the average numbers of tokens of colors "F" and "B" passing through $Tsinp$ are equal). Since the firing times of $Tsinp$ and $Trmem$ are equal (both are 10 time units), the utilization of $Trmem$ close to 0.25 corresponds to near maximum (i.e., close to 1.0) utilization of $Tsinp$. It should be noticed that if the switch delay (i.e., $f(Tsinp)$ and $f(Tsout)$) is reduced to 5 time units (and all other parameters remain the same), the saturation level of $Trmem$'s utilization increases to 0.5 .

Fig.4.3 shows the utilization of the processor as a function of the number of threads and the probability of long–latency accesses to the local memory (as in Fig.4.1), but with a different value of the runlength ($f(Trun) = 5$ in Fig.4.3).

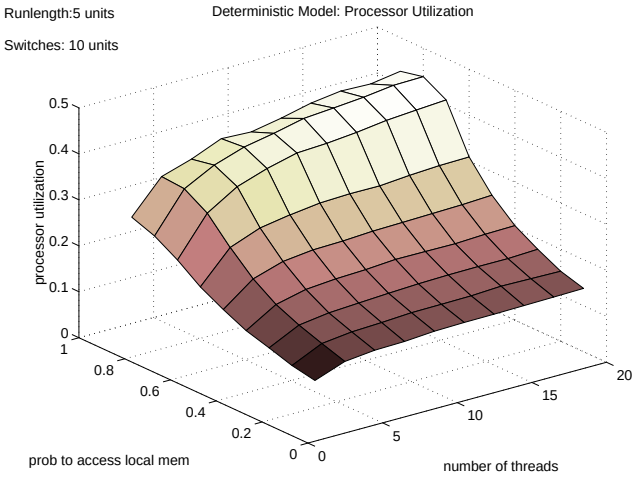

Fig.4.3. Processor utilization.

The utilization of the processor is approximately half of that in Fig.4.1 (if all long–latency accesses are to the local memory, the interconnecting network can be ignored, each processor can be analyzed in isolation, and the processor utilization, for large numbers of threads, approaches $f(Trun)/f(Trmem)$ if $f(Trun) \leq f(Trmem)$).

Fig.4.4 shows the effect of the probability distribution function on the utilization of the processor; more specifically, the results shown in Fig.4.4 correspond to the case when all firing times are exponentially distributed (so the model is Markovian), and the average values of firing times are equal to those used for the results shown in Fig.4.1.

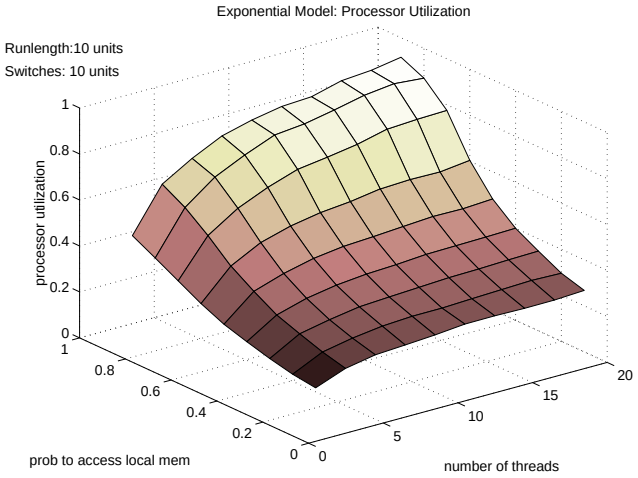

Fig.4.4. Processor utilization.

The results in Fig.4.4 are very similar to those in Fig.4.1, which indicates that the model is rather insensitive to the probability distribution functions.

Many other characteristics can be obtained from simulation results, for example, the waiting times of *Ready* threads, utilizations of the network switches, average queue lengths and waiting times of requests at network switches (in the forward and backward directions), the latency of the interconnecting network.

5. Simplified Model

The model discussed in Section 3 is composed of many identical submodels (of nodes), connected by a two–dimensional torus network. An obvious question is if this 'regularity' of the model can be used for model simplification.

A similar problem is discussed in [4] and [3]. In both cases, however, the structure of the models is significantly different. In [4], the processors are connected by buses, so the 'folding' of individual processor models does not require any other adjustments of the model. In [3], the multiprocessor system is composed of (identical) transputer–like processors, each of them running memory handling, receiver, transmitter and application tasks. The model can easily be 'folded' to a single processor in which the outgoing 'stream' is becoming the incoming stream of requests as the effects of messages passing through a node are not analyzed.

The model discussed in this paper is composed of two distinct parts, a number of (identical) processors and the interconnecting network. For any reduction of the number of processors, the model of the interconnecting network must be adjusted in such a way

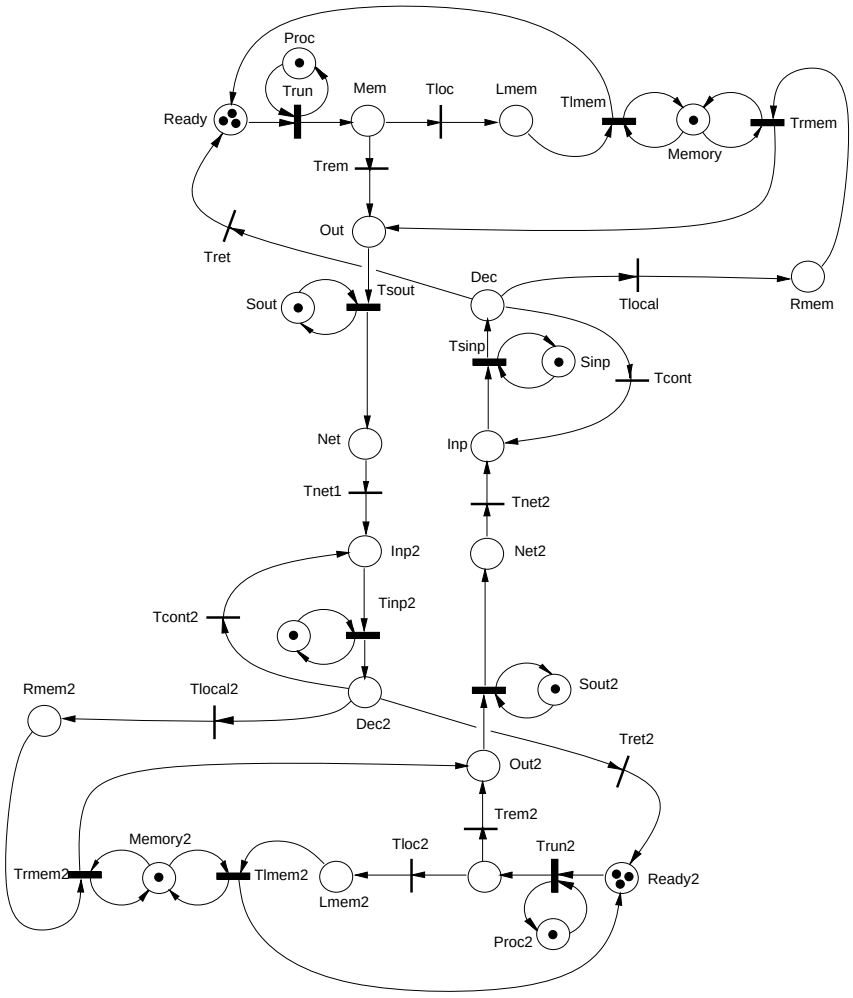

Fig.5.1. Simplified model of a multithreaded multiprocessor architecture.

that the behavior of the (remaining) processor(s) is preserved.

A simplification of the 'expanded' model can be based on an observation that if the original throughputs of the switches associated with a node can be preserved in a simplified model, the node's behavior should be the same as in the original 'expanded' model because the two switches (the input and the output switch) provide the only interaction (or an 'interface') of the node with the other nodes. It appears that a 2–node model, shown in Fig.5.1, can preserve the behavior of the switches quite accurately.

The simplified model is based on the following consideration. Let the throughput of $Trem$ (i.e., the rate of generating remote memory access requests), in the original model, be θ_{rem}. In the steady–state, the throughputs of $Tret$ and $Trmem$ are also equal to θ_{rem}. Then it follows immediately that the throughput of $Tsout$, the output switch, is $2\theta_{rem}$, and since the free–choice probability of Tgo (for a 16–processor system) is 0.5 (Section 3), the throughput of $Tsinp$, the input switch, is $4\theta_{rem}$.

The same conclusions can be reached by considering the remote requests. Since the average distance of remote requests is 2 hops (Section 3), each request passes (on average) 3 switches in the forward direction, and also 3 switches in the backward direction. Out of these 6 switches, 4 are $Tsinp$ switches, and 2 are $Tsout$ switches. For a uniform distribution of requests over the whole system, in the steady–state, the throughputs of $Tsinp$ and $Tsout$ are thus $4\theta_{rem}$ and $2\theta_{rem}$, respectively. In order to provide the required throughput of $Tsinp$ in the simplified model (Fig.5.1), a loop is introduced from Dec to Inp and from $Dec2$ to $Inp2$ with free–choice probabilities (of $Tcont$ and $Tcont2$) equal to 0.5; the consequence of these loops is

that each token (on average) passes the switch twice, which effectively doubles the original throughputs of these switches.

Fig.5.2 shows the processor utilization obtained by simulation of the simplified model. The results are very similar to those in Fig.4.1 (they are obtained for the same values of model parameters; the differences between these two sets of results are within a few percent).

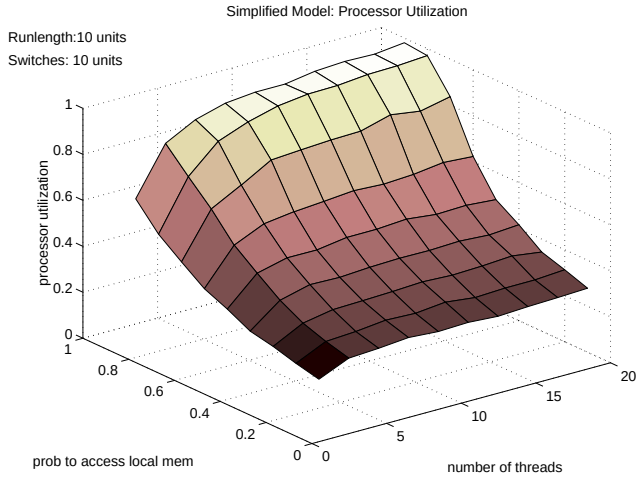

Fig.5.2. Processor utilization – simplified model.

6. Concluding Remarks

A Petri net model of a multithreaded multiprocessor architecture is presented at a more detailed level than than the one in [17]. The model is developed using colored timed nets. Event–driven simulation is used to obtain performance characteristics of the model.

An approach to simplified modeling is also discussed and illustrated by results which match quite accurately the results obtained for the original model.

The accuracy of the results obtained from the simplified model indicates that the 'expanded' model may be needed rather infrequently, and that quite simple models can be used for analysis of complex systems. However, more research is needed in this area in order to identify the limitations of the simplified approach.

The same simplified model (Fig.5.1) can be used for many architectures with only minor adjustments. For example, it can be used for analysis of 16–processor systems as well as 9–processor systems, 25–processor ones, etc.; the only modification which needs to be done is an adjustment of the free–choice probability of $Tcont$ and $Tcont2$ in Fig.5.1.

The simplified model is sufficiently small to be analyzed by the reachability analysis, especially for the small numbers of threads (provided that the deterministic and stochastic transitions can be dealt with in the same model, otherwise one of these two types of transitions must be converted into the other).

The simulation results are consistent with results presented in the literature [17]. Because of several low–level differences between the models, and differences in probability distribution functions, the numerical results are slightly different, but all the relationships and trends seem to be preserved.

The timed Petri net model of the multithreaded processor (Fig.3.1) is very simple; it is much simpler than stochastic net models of (similar) processors presented for example in [4].

Although only one specific multithreaded architecture was discussed in this paper, the models can very easily be adjusted to the other architectures indicated in Section 2. For example, only a minor modification of transitions $Tloc$ and $Tlmem$ (Fig.3.1) is needed to model "switching on remote load"; on the other hand, however, it can be observed that the utilization of processors for this approach will be lower than for "switching on every load" because the processor will be idle during the (local) memory accesses (typically 10 time units) while the context switching typically requires only one time unit. These utilization differences will be more pronounced for high probabilities of long–latency accesses to local memory. Similarly, "switching on block of instructions" should result in (slightly) lower utilization since context switching is (sometimes) performed when it is not necessary. The effect of 'unnecessary' context switching is even more significant for "switching on every instruction". The "switching on every load" approach is thus expected to provide the best performance (in the sense of utilization of processors) unless the timing parameters differ very significantly from the assumed ones (e.g., when the context switching time is rather long). Also, more research is needed to use all the resources of a multithreaded multiprocessor system in the optimal way.

Acknowledgements

Several interesting remarks and suggestions of four anonymous reviewers are gratefully acknowledged. In particular, the relevance of reference [3] was indicated by one of reviewers.

The Natural Sciences and Engineering Research Council of Canada partially supported this research through Research Grant OGP8222. Memorial's Dean of Science travel support for the first author is appreciated.

References

[1] Agrawal, A., Lim, B-H., Kranz, D., Kubiatowicz, 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] Ajmone Marsan, M., Balbo, G., Chiola, G., Conte, G., "Modeling the software architecture of a prototype parallel machine"; Proc. SIGMETRICS'87, Performance Evaluation Review, vol.15, no.1, pp.175-185, 1987.

[4] Ajmone Marsan, M., Balbo, G., Conte, G., "Performance models of multiprocessor systems"; MIT Press 1986.

[5] Alkalaj, L., Boppana, R.V., "Performance of a multithreaded execution in a shared–memory multiprocessor"; Proc. 3-rd Annual IEEE Symp. on Parallel and Distributed Processing, Dallas, TX, pp.330-333, 1991.

[6] 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.

[7] 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.

[8] 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.

[9] Edmondson, J., Rubinfield, P., Preston, R., Rajagopalan, V., "An overview of the 21164 AXP microprocessor"; IEEE Micro, vol.15, no.2, pp.33-43, 1995.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] Johnson, K., "The impact of communication locality on large–scale multiprocessor performance"; Proc. 19-th Annual Int. Symp. on Computer Architecture, pp.392-402, 1992.

[15] Keckler, S.W., Dally, W.J., "Processor coupling: integration of compile-time and run-time scheduling for parallelism"; Proc. 19-th Annual Int. Symp. on Computer Architecture, pp.202-213, 1992.

[16] Murata, T., "Petri nets: properties, analysis and applications"; Proceedings of IEEE, vol.77, no.4, pp.541–580, 1989.

[17] Nemawarkar, S.S., Govindarajan, R., Gao, G.R., Agarwal, V.K., "Analysis of multithreaded multiprocessors with distributed shared memory"; Micronet Report, Department of Electrical Engineering, McGill University, Montreal, Canada H3A 2A7, 1993.

[18] Nemawarkar, S.S., Gao, G.R., "Performance analysis of multithreaded architectures using an integrated system model"; Parallel Computing (submitted for publication).

[19] Reisig, W., "Petri nets - an introduction" (EATCS Monographs on Theoretical Computer Science 4); Springer Verlag 1985.

[20] 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.

[21] 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.

[22] Song, S.P., Denman, M., Chang, J., "The PowerPC 604 RISC microprocessor"; IEEE Micro, vol.14, no.5, pp.8-17, 1994.

[23] Tremblay, M., Greenlay, D., Normoyle, K., "The design of microarchitecture of UltraSPARC-I"; Proceedings of IEEE, vol.83, no.12, pp.1653-1663, 1995.

[24] 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.

[25] 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` as `/pub/publications/91-MaR.ps.Z`).

[26] 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` as `/pub/techreports/tr-9602.ps.Z`).