35-th Annual Simulation Symp. of Advanced Simulation Technologies Conf., San Diego, CA, 14-18 April 2002, pp.107-113.
Copyright © 2002 IEEE (DOI 10.1109/SIMSYM.2002.1000105).

Approximate Simulation of Distributed–Memory Multithreaded Multiprocessors

W.M. Zuberek
Department of Computer Science
Memorial University of Nfld
St.John's, Canada A1B 3X5
wlodek@cs.mun.ca

Abstract

The performance of modern computer systems is increasingly limited by long latencies of accesses to their memory systems. Instruction–level multithreading is a technique to tolerate long latencies of memory accesses by switching from one instruction thread to another. The paper shows that the simulation–based performance evaluation of distributed–memory multithreaded multiprocessor systems can be significantly simplified by using approximate models, composed of only a few processors, but with some parameters adjusted to represent the behavior of the original system.

1 Introduction

In modern computer systems, the performance of memory is increasingly often becoming the factor limiting the performance of the system. Due to continuous progress in manufacturing technologies, the performance of processors has been doubling every 18 months (the so–called Moore's law [6]). However, the performance of memory chips has been improving by only 10% per year [12], creating a "performance gap" in matching processor's performance with the required memory bandwidth. In effect, it is becoming more and more often the case that the performance of applications depends on the performance of machine's memory hierarchy.

Memory hierarchies, and in particular multi–level cache memories, have been introduced to reduce the effective latency of memory accesses. Cache memories provide efficient access to information when the information is available at lower levels of memory hierarchy; occasionally, however, long–latency memory operations are needed to transfer the information from the higher levels of memory hierarchy to the lower ones. Much research has focused on reducing and tolerating these large memory access latencies. Techniques for reducing the frequency and impact of cache misses include hardware and software prefetching [4, 8], speculative loads and execution [13] and multithreading [1, 3].

Instruction–level multithreading, and in particular block–multithreading [1, 2, 3], tolerates long–latency memory accesses and synchronization delays by switching to another thread rather than waiting for the completion of a long–latency operation which, in a distributed–memory system, can require hundreds or even thousands of processor cycles. A combination of multithreading and superscalar architecture is also an approach used in high–performance microprocessors [9].

The purpose of this paper is to study a simplification of the simulation–based performance evaluation of distributed–memory multithreaded multiprocessors which is based on the symmetries that exist in such systems. More specifically, the paper shows that a simplified model, using only a few processors, provides a good approximation of the complete system if some modeling parameters are adjusted to the values representing the original system. The simulation–time reduction achieved in this way, for a 16–processor system presented in this paper, is more than five times; for analysis of larger systems the gain is even more significant.

Timed Petri nets [14] are used to model multithreaded multiprocessor systems at the instruction execution level. Temporal characteristics of the original system are represented by time attributes associated with the transitions of the net model. Free–choice net structures are used to model probabilistic aspects of the system. Performance characteristics of the analyzed system are obtained by discrete–event simulation of its net model.

A multiprocessor system with 16 processors connected by a 2–dimensional torus–like network is used as a running example in this paper. An outline of such a system is shown in Fig.1.

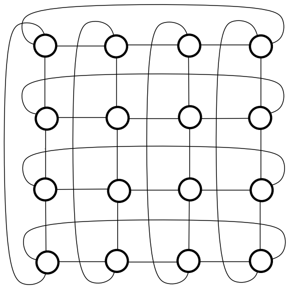

Fig.1. Outline of a 16–processor system.

It is usually assumed that the memory access requests sent from one node to another are routed along the shortest paths. It is also assumed that this routing is done in a nondeterministic way, i.e., if there are several shortests paths between two nodes, each of them is equally likely to be used. Consequently, the traffic is assumed to be uniformly distributed in the interconnecting network. The average length of the shortest path between two nodes, or the average number of hops (from one node to another) that a request must perform to reach its destination, n_h, is usually determined assuming that the memory accesses are uniformly distributed over the nodes of the system.

Although many specific details refer to this 16–processor system, most of them can easily be adjusted to other systems by changing the values of a few parameters.

Each node in the system shown in Fig.1 is a multithreaded processor which contains a processor, local memory, and two network interfaces, as shown in Fig.2.

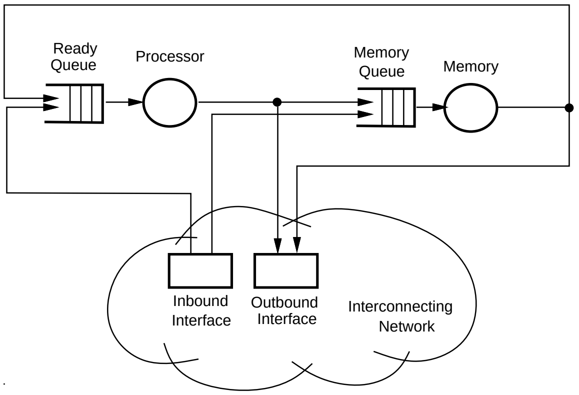

Fig.2. Outline of a single multithreaded processor.

The outbound switch handles outgoing traffic, i.e., requests to remote memories originating at this node as well as results of remote accesses to the memory at this node; the inbound interface handles incoming traffic, i.e., results of remote requests that 'return' to this node and remote requests to access memory at this node.

Fig.2 also shows a queue of ready threads; whenever the processor performs a context switching (i.e., switches from one thread to another), a thread from this queue is selected for execution and the execution continues until another context switching is performed. In block multithreading, context switching is performed for all long–latency memory accesses by 'suspending' the current thread, forwarding the memory access request to the relevant memory module (local, or remote using the interconnecting network) and selecting another thread for execution; when the result of this request is received, the status of the thread changes from 'suspended' to 'ready', and the thread joins the queue of ready threads, waiting for another execution phase on the processor.

The average number of instructions executed between context switching is called the runlength of a thread, ℓ_t, and is one of important modeling parameters. It is directly related to the probability that an instruction requests a long–latency memory operation.

Another important modeling parameter is the probability of long–latency accesses to local, p_ℓ, (or remote, $p_r = 1 - p_\ell$) memory (in Fig.2 it corresponds to the "decision point" between the Processor and the Memory Queue); as the value of p_ℓ decreases (or p_r increases), the effects of communication overhead and congestion in the interconnecting network (and its switches) become more pronounced; for p_ℓ close to 1, the nodes can be practically considered in isolation.

The (average) number of available threads, n_t, is yet another basic modeling parameter. For very small values of n_t, queueing effects can be practically neglected, so the performance can be predicted by taking into account only the delays of system's components. On the other hand, for large values of n_t, the system can be considered in saturation, which means that one of its components will be utilized in almost 100 %, limiting the utilization of other components as well as the whole system. Identification of such limiting components (called the bottlenecks [7]) and improving their performance is the key to the improved performance of the entire system.

2 Petri net model

Petri nets [11, 10] are popular models of systems that exhibit concurrent and parallel activities. In timed Petri nets [14], the durations of modeled activities are also taken into account in order to study the performance characteristics of the systems.

A timed Petri net model of a multithreaded processor at the level of instruction execution is shown in Fig.3 [15]. As usual, timed transitions are represented by "thick" bars, and immediate ones, by "thin" bars.

The execution of each instruction of the 'running' thread is modeled by transition $Trun$, a timed transition with the firing time representing one processor cycle. Place *Proc* represents the (available) processor (if marked) and place *Ready* – the queue of threads waiting for execution. The initial marking of *Ready* represents the average number of available threads, n_t. It is assumed that this number does not change in time.

If the processor is available (i.e., *Proc* is marked) and *Ready* is not empty, a thread is selected for execution by firing the immediate transition $Tsel$. Execution of consecutive instructions of the selected thread is performed in the loop *Pnxt*, $Trun$, *Pend* and $Tnxt$. *Pend* is a free–choice place with the choice probabilities reflecting the runlength, ℓ_t, of the thread. In general, the free–choice probability assigned to $Tnxt$ is equal to $(\ell_t - 1)/\ell_t$, so if ℓ_t is equal to 10, the probability of $Tnxt$ is 0.9; if ℓ_t is equal to 5, this probability is 0.8, and so on. The free–choice probability of $Tend$ is just $1/\ell_t$.

If $Tend$ is chosen for firing rather than $Tnxt$, the execution of the thread ends, a request for a long–latency access to (local or remote) memory is placed in *Mem*, and a token is also deposited in *Pcsw*. The timed transition $Tcsw$ represents the context switching. When its firing is finished, another thread is selected for execution (if it is available).

Mem is a free–choice place, with a random choice of either accessing local memory ($Tloc$) or remote memory ($Trem$); in the first case, the request is directed to *Lmem*

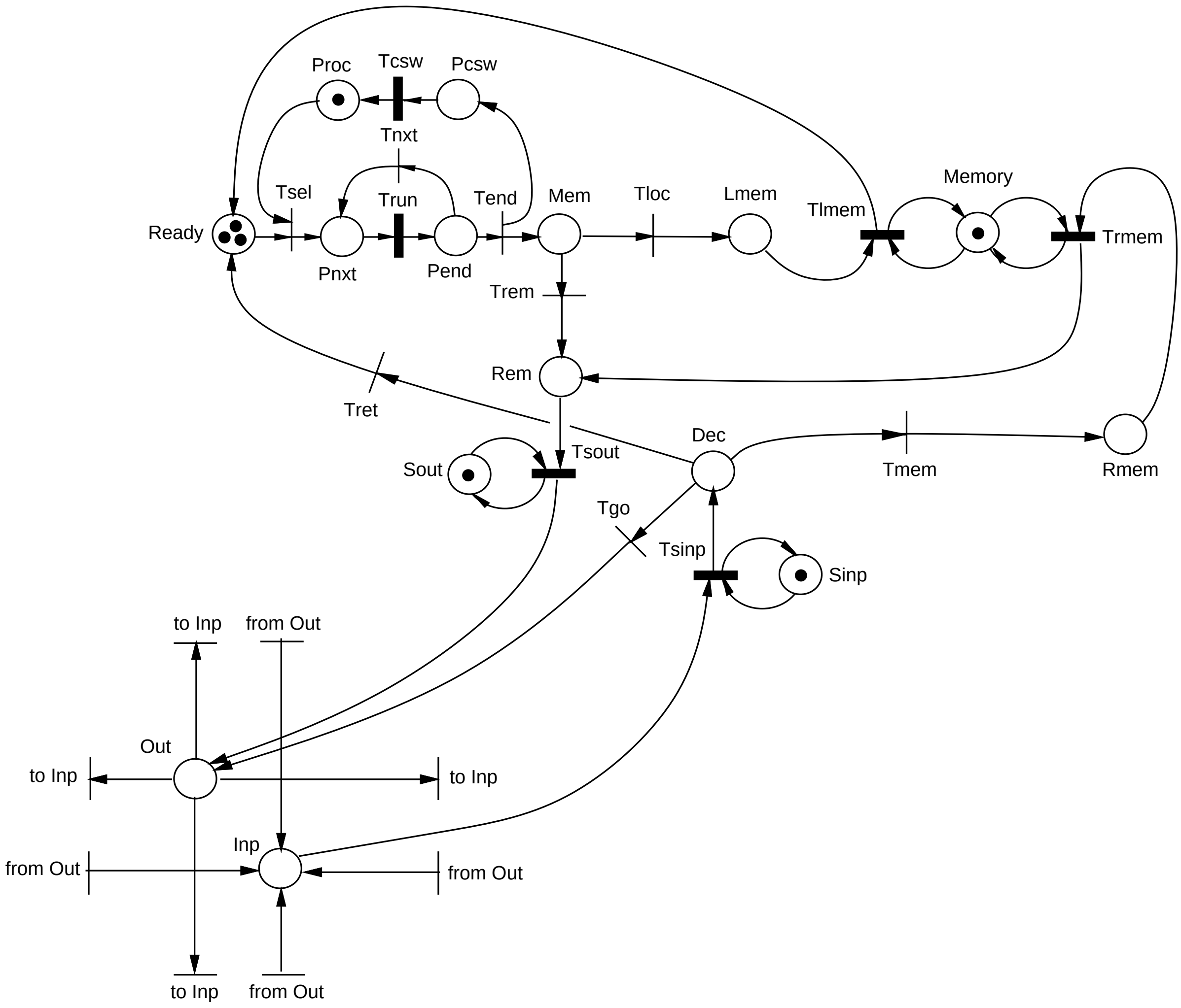

Fig.3. Instruction–level Petri net model of a multithreaded processor.

where 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, is the probability of long–latency accesses to remote memory; the free–choice probability of *Tloc* is $p_\ell = 1 - p_r$.

Requests for remote accesses are directed to *Rem*, and then, after a sequential delay (the outbound 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 is collected from all neighboring nodes in *Inp*, and, after a sequential delay (the inbound switch *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' in the interconnecting network); and *Tmem*, 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 free–choice probabilities associated with *Tret*, *Tgo* and *Tmem* characterize the interconnecting network [5]. For a 16–processor system (as in Fig.1), and for memory accesses uniformly distributed among the nodes of the system, the free–choice probabilities of *Tmem* and *Tgo* are 0.5 for forward moving requests, and 0.5 for *Tret* and *Tgo* for returning requests.

The traffic outgoing from a node (place *Out*) 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*).

3 Simplified model

Since, in the model of a complete system, each processor is represented by the net shown in Fig.3, the complete model can easily become rather complex. Therefore, its performance characteristics are typically obtained by a simulation. However, since all processors are identical, the simulation of the complete system may not be necessary; the simulation time could be reduced significantly is only a small subsystem, for example with 4 processors, as shown in Fig.4, could be used providing a reasonable approximation of the original system.

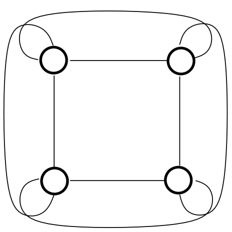

Fig.4. Outline of a 4–processor system.

Each processor interacts with the remaining system through its four neighboring nodes; each processor sends a stream of requests for remote memory accesses, and receives (in the steady state) an identical stream of requests from its neighbors. So, if these streams of requests in the simplified model can be made the same as in the original system, the simulation model could be restricted to just a 4–processor system providing a reasonable approximation of the original system.

The stream of requests for remote memory accesses depends on the number of processors, and this dependency is not a straightforward one; if the number of processors increases, the average number of hops in the interconnecting networks that a request needs to perform to reach its destination, also increases, so the effective latency of accessing remote memory increases, and the utilization of the processor decreases (especially for values of the probability p_r close to 1).

To obtain the same effect in a simplified model, two conditions should be satisfied: (i) the average number of hops, n_h, should be adjusted to the same value as in the original model (this is needed to preserve the latency of each remote memory access request), and (ii) the average number of transfers in each link of the interconnecting network needs to be the same as in the original system (which is needed to preserve the traffic in the network and the delays which are introduced by this traffic). It appears that requirement (ii) is closely related to (i). In a multiprocessor system with n_p nodes connected by a two–dimensional torus-like network, there are $2n_p$ links connecting the nodes. In a steady–state, in each (average) time period of executing instructions and issuing a long–latency operation, each node issues (on average) p_r requests to access remote memory, and each such request performs (on average) n_h hops in the interconnecting network. Taking into account that (remote) requests are followed by their results sent back to "home" nodes, the total number of transfers in the interconnecting network is $2p_rn_hn_p$, so the (average) number of transfers per link is p_rn_h. Consequently, if n_h in the simplified model is adjusted to the value in the original system, the average number of transfers in each link will also follow the original system.

If the model shown in Fig.4 is supposed to approximate the 16–processor model, the average number of hops in the interconnecting network should be made equal to 2 (the average number of hops can be approximated reasonably well by $\sqrt{n_p}/2$ [16]). In the model shown in Fig.3, the average number of hops is controlled by the free–choice probabilities associated with transitions Tgo, $Tret$ and $Tmem$. In particular, if the free–choice probability associated with Tgo is equal to q, the average number of hops that requests (and responses) perform before reaching their destination nodes is equal to $1/(1-q)$; for n_h equal to 2, this probability should be equal to 0.5 (and then the free–choice probabilities of $Tret$ and $Tmem$ are both equal to $1-q$).

In order to simulate the performance of a 16–processor system using a 4–processor model, the free–choice probabilities associated with transitions Tgo, $Tret$ and $Tmem$ (Fig.3) need to be adjusted to the values representing the original, 16–processor system (i.e., 0.5).

4 Performance results

It is convenient to assume that all timing characteristics are expressed in processor cycles (which is assumed to be 1 unit of time). The basic model parameters and their typical values are as follows:

symbol	*parameter*	*typical values*
n_t	the number of threads	2,...,20
ℓ_t	thread runlength	5,10,15
t_{cs}	context switching time	1,2
t_m	memory cycle time	10
t_s	switch delay	10,5
p_ℓ, p_r	probability of accesses to local/remote memory	0.1,...,0.9

Fig.5 shows the utilization of the processor, in a 16–processor system, as a function of the number of available threads, n_t, and the probability of long–latency accesses to local memory, p_ℓ, for fixed values of other parameters.

Fig.6 shows the utilization of the processor, in a 4–processor system with the adjusted value of n_h, also as a function of the number of available threads, n_t, and the probability of long–latency accesses to local memory, p_ℓ. It should be observed that the difference are not significant, and are within a few percent of the values obtained by simulation of the 16–processor system.

The utilization of the processor in an unmodified 4–processor system is shown in Fig.7. The processor's utilization is significantly different than in Fig.5 and Fig.6 because the influence of the interconnecting network is different; in this case, the service demand for the switches in the interconnection network is one half of that in a 16–processor system [16], so the effect of the switch delays is much less significant than in the 16–processor system (Fig.5).

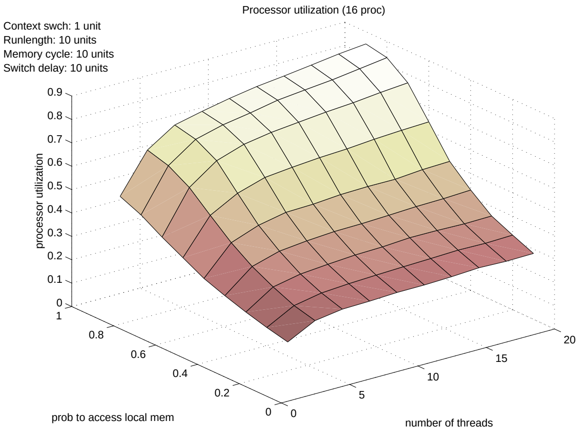

Fig.5. Processor utilization – 16 processors;
$t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 10$.

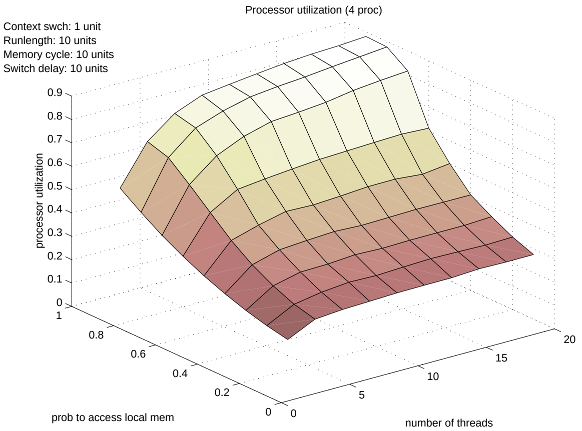

Fig.6. Processor utilization – adjusted 4 processors;
$t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 10$.

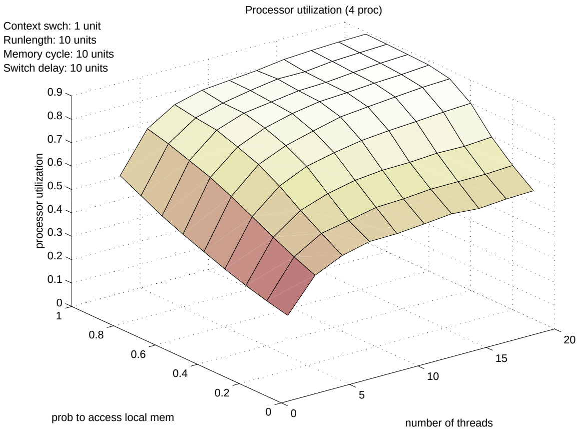

Fig.7. Processor utilization – original 4 processors;
$t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 10$.

The strong influence of the switch delay in Fig.5 and Fig.6 (for larger values of p_r, the probability of accessing remote memory) is an indication that the switches are the bottleneck in this system as they are utilized in almost 100% and they limit the performance of the entire system. Indeed, Fig.8 shows the utilization of the (input) switches in the 16–processor system as a function of the number of available threads, n_t, and the probability of long–latency accesses to remote (not local) memory, p_r (so the front part of Fig.8 corresponds to the back part of Fig.5).

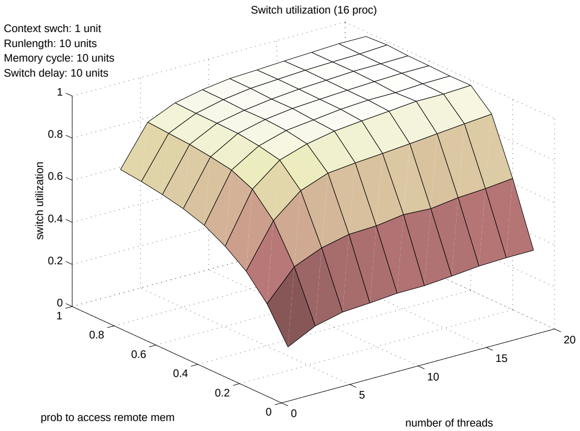

Fig.8. Switch utilization – 16 processors;
$t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 10$.

Fig.9 and Fig.10 show the utilization of the processors in the 16–processor system and in the adjusted 4–processor system, respectively, for the case when the switch delay is reduced two times.

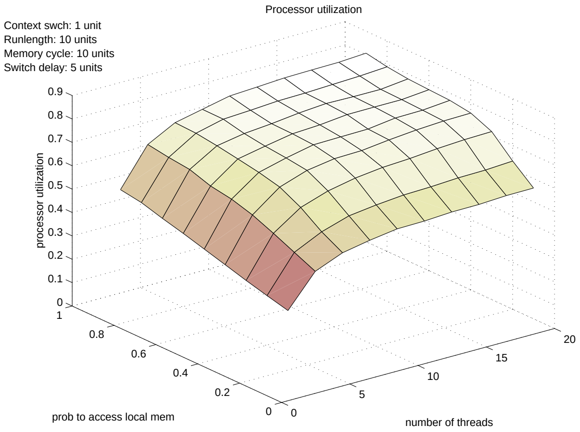

Fig.9. Processor utilization – 16 processors;
$t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 5$.

It should be observed that the performance is significantly better than in the original system (Fig.5 and Fig.6), and that the region in which the switch is the bottleneck, is substantially reduced (although it could be reduced even more). As before, the agreement of the results obtained for the original 16–processor model and the simplified 4-processor one, is quite good. Further reduction of the switch delay would extend the “flat” region of the performance surface in Fig.9 and Fig.10.

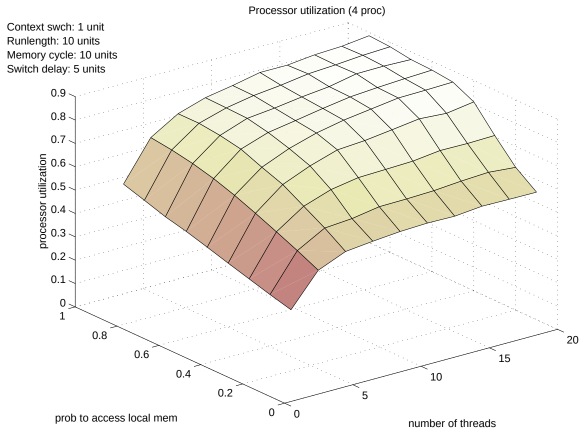

Fig.10. Processor utilization – adjusted 4 processors; $t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 5$.

Fig.11 and Fig.12 show the influence of another parameter, the runlength of a thread, ℓ_t.

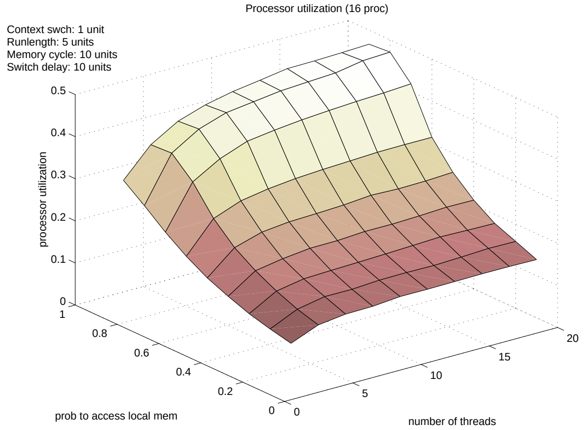

Fig.11. Processor utilization – 16 processors; $t_{cs} = 1$, $\ell_t = 5$, $t_m = 10$, $t_s = 10$.

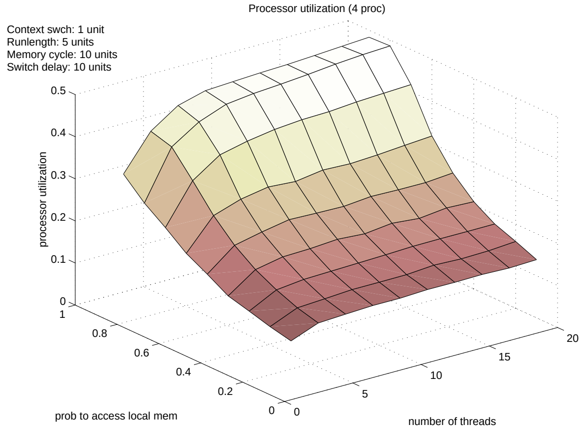

Fig.12. Processor utilization – adjusted 4 processors; $t_{cs} = 1$, $\ell_t = 5$, $t_m = 10$, $t_s = 10$.

It can be shown [16] that when the value of ℓ_t is smaller than t_m, the utilization of the processor has an upper bound of ℓ_t/t_m, so in this case the bound is 0.5. Indeed, the plots are very similar one to another (which again indicates that the simplified model is a good approximation of the complete model), and are also similar to the plots in Fig.5 and Fig.6, but with a different scale (the utilization in Fig.11 and Fig.12 is one half of that in Fig.5 and Fig.6).

5 Concluding remarks

The presented performance results for distributed–memory multithreaded multiprocessor systems indicate that significant simulation time reductions can be achieved by using simplified models with some parameters adjusted to the values corresponding to the original systems. Since the simulation time of Petri net models increases more than linearly with the size of the model, the gains in the simulation time also increase more than linearly with the size of the (original) model.

Moreover, an improved accuracy of simulation results can be obtained using (simplified) models with larger number of nodes; for example, a model of a 9–processor system with parameters adjusted to 16–processor values can be used to provide results more accurate than the 4–processor model but still requiring much less simulation time than the complete 16–processor model. Fig.13, Fig.14 and Fig.15 show the utilizations of processors for the three sets of modeling parameters that correspond to results in Fig.5 and Fig.6, Fig.10 and Fig.11, and Fig.12 and Fig.13, respectively.

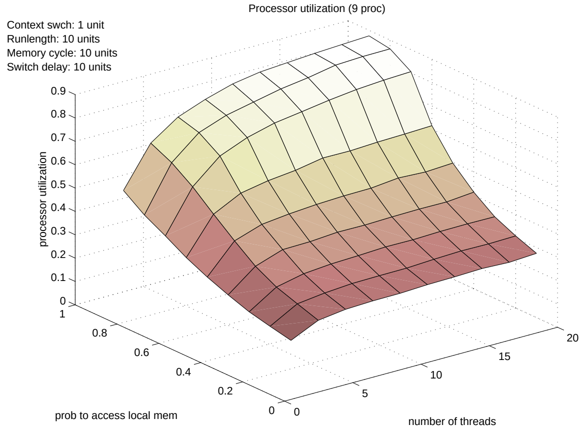

Fig.13. Processor utilization – adjusted 9 processors; $t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 10$.

The derived models assume that accesses to memory are uniformly distributed over the nodes of the system. If this assumption is not realistic and some sort of 'locality' is present, the only change that needs to be done is an adjustment of the value of n_h; for example, if the probability of accessing nodes decreases with the distance (i.e., nodes which are close are more likely to be accessed that the distant ones), the value of n_h will be smaller than that determined for the uniform distribution of accesses, and will result in improved performance.

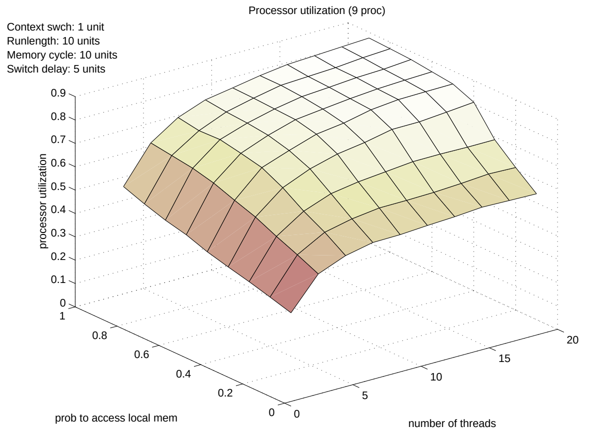

Fig.14. Processor utilization – adjusted 9 processors; $t_{cs} = 1$, $\ell_t = 10$, $t_m = 10$, $t_s = 5$.

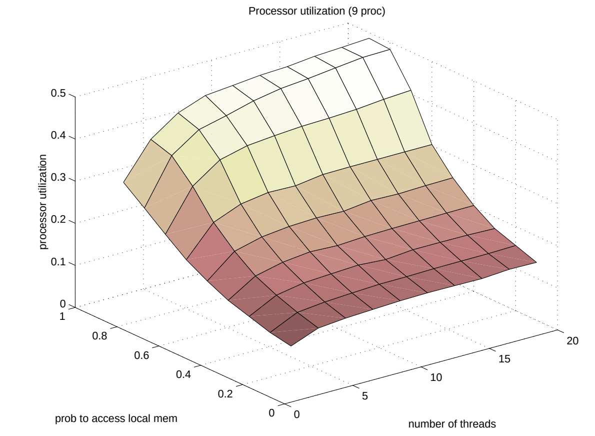

Fig.15. Processor utilization – adjusted 9 processors; $t_{cs} = 1$, $\ell_t = 5$, $t_m = 10$, $t_s = 10$.

Finally, it should be noted that the presented results provide only some insight into the behavior of multithreaded systems; in the real systems some of the assumptions are not satisfied – for example, the number of threads is rarely constant, the probabilities of accessing local or remote memory may significantly change during the executions of programs, and so on.

Acknowledgment

The Natural Sciences and Engineering Research Council of Canada partially supported this research through grant RGPIN–8222.

References

[1] Agarwal, A., "Performance tradeoffs in multithreaded processors"; IEEE Trans. on Parallel and Distributed Systems, vol.3, no.5, pp.525-539, 1992.

[2] Boothe, B. and Ranade, A., "Improved multithreading techniques for hiding communication latency in multiprocessors"; Proc. 19-th Annual Int. Symp. on Computer Architecture, Gold Coast, Australia, pp.214-223, 1992.

[3] Byrd, G.T. and Holliday, M.A., "Multithreaded processor architecture"; IEEE Spectrum, vol.32, no.8, pp.38-46, 1995.

[4] Chen, T-F. and Baer, J-L., "A performance study of software and hardware data prefetching scheme"; Proc. 21-st Annual Int. Symp. on Computer Architecture, Chicago, IL, pp.223-232, 1994.

[5] Govindarajan, R., Suciu, F. and 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.

[6] Hamilton, S., "Taking Moore's law into the next century"; IEEE Computer Magazine, vol.32, no.1, pp.43-48, 1999.

[7] Jain, R., "The art of computer systems performance analysis"; J. Wiley & Sons 1991.

[8] Klaiber, A.C. and Levy, H.M., "An architecture for software-controlled data prefetching"; Proc. 18-th Annual Int. Symp. on Computer Architecture, Toronto, Canada, pp.43-53, 1991.

[9] Loh, K.S. and Wong, W.F., "Multiple context multithreaded superscalar processor architecture"; Journal of Systems Architecture, vol.46, pp.243-258, 2000.

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

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

[12] Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P. and Ovens, J.D., "Memory access scheduling"; Proc. 27-th Annual Int. Symp. on Computer Architecture, Vancouver, Canada, pp.128-138, 2000.

[13] Rogers, A. and Li, K., "Software support for speculative loads"; Proc. 5-th Symp. on Architectural Support for Programming Languages and Operating Systems, pp.38-50, 1992.

[14] Zuberek, W.M., "Timed Petri nets – definitions, properties and applications"; Microelectronics and Reliability, vol.31, no.4, pp.627–644, 1991.

[15] Zuberek, W.M., "Performance modeling of multithreaded distributed memory architectures", Proc. 2-nd Workshop on Hardware Design and Petri Nets, Williamsburg, VA, pp.63–82, 1999.

[16] Zuberek, W.M. and Govindarajan, R., "Performance balancing in multithreaded multiprocessor systems"; Proc. 4-th Australasian Conf. on Parallel and Real-Time Systems (PART'97), Newcastle, Australia, pp.15-26, 1997.