4-th Int. Conf. on Application of Concurrency to System Design, Hamilton, Canada, 16–18 June 2004, pp.7-15.
 (DOI 10.1109/CSD.2004.1309111).

Enhanced Interleaved Multithreaded Multiprocessors and Their Performance Analysis

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

Abstract

In interleaved multithreading, the thread changes in each processor cycle, consecutive instructions are issued from different threads, and no data dependencies can stall the pipeline. Enhanced inteleaved multithreading maintains a number of additional threads which are used to replace an active thread when it initiates a long–latency operation. Instruction issuing slots, which are lost in pure interleaved multithreading, are thus used by instructions from the new thread. The paper studies performance improvements due to enhanced multithreading by analyzing a timed Petri net model of an enhanced multithreaded architecture at the instruction execution level.

Keywords: *Interleaved multithreaded architectures, distributed–memory multiprocessors, timed Petri nets, performance analysis, event–driven simulation.*

1 Introduction

Due to continuous progress in manufacturing technologies, the performance of microprocessors has been steadily improving over the last decades, doubling every 18 months (the so called Moore's law [5]). At the same time, the capacity of memory chips has also been doubling every 18 months, but the performance has been improving less than 10% per year [9]. The latency gap between the processor and its memory doubles approximately every six years, and it is not unusual that as much as 60% of the processor's time is spent on waiting for the completion of memory operations [11]. Matching the performances of the processor and the memory is an increasingly difficult task [13].

In distributed–memory systems, the latency of memory accesses is much more pronounced than in centralized–memory systems as memory access requests may need to be forwarded through several intermediate nodes before they reach their destination, and then the results need to be sent back to the original nodes. Each of the "hops" introduces some delay, typically assigned to the switches that control the traffic between the nodes [3], [4].

Instruction–level multithreading is a technique of tolerating long–latency memory accesses and synchronization delays in multiprocessor systems [1], [2], and in particular, in distributed–memory systems. The general idea is quite straightforward. In block multithreading, when a long–latency memory operation occurs, the processor, instead of waiting for the completion of this operation (which in distributed–memory systems can require hundreds and even thousands of processor cycles), switches to another thread if such a thread is ready for execution. If different threads are associated with different sets of processor registers, switching from one thread to another can be done very efficiently [1], [3].

In interleaved multithreading (also known as fine–grain multithreading), the thread changes in every processor cycle [12]; this approach is advantageous for eliminating data dependencies that slow–down the processor's pipeline; since consecutive instructions are issued from different threads, they have no data dependencies. Typically, the number of threads is equal to the number of pipeline stages, so no inter-instruction dependencies can stall the pipeline. In pure interleaved multithreading, a thread issuing a long–latency memory operation becomes 'waiting' for the result of the requested operation. If a waiting thread is selected for execution, its slot simply remains empty (i.e., no instruction is issued), which is equivalent to a single–cycle pipeline stall. Since the threads issue their instructions one after another, fewer processor cycles are lost during a long–latency operation of a single thread.

In enhanced interleaved multithreading [17], additional threads are available to replace any active thread when it initiates a long–latency operations and becomes inactive until the end of the initi-

ated operation. Consequently, the processor cycles are not lost, the utilization of processors increases and this improves the performance of the system. The enhanced interleaved multithreading combines elements of interleaved and block multithreading within one architecture.

The main objective of this paper is to study the performance improvements that can be obtained by enhancing interleaved multithreading. In particular, the relationship between the number of additional threads and the improvements which they can provide is investigated. This investigation is performed using a timed Petri net model of an enhanced multithreaded distributed–memory system at the instruction execution level. The performance of this model is studied as a function of several modeling parameters.

A distributed memory 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, in which all connections between nodes are actually double because they are used for communication in both directions.

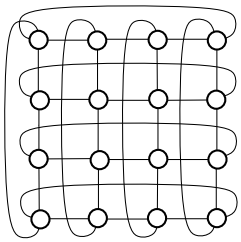

Fig.1. Outline of a 16–processor system.

It is assumed that all messages 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 shortest paths between two nodes, each of them is equally likely to be used. The average length of the shortest path between two nodes, or the average number of hops (from one node to another) that a message must perform to reach its destination, 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 these details can easily be adjusted to other systems by changing the values of a few model parameters [19].

Each node in the network shown in Fig.1 is a fine-grain multithreaded processor which contains a processor, local memory, and two network interfaces, as shown in Fig.2. The outbound interface 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 deals with incoming traffic, i.e., results of remote requests that "return" to this node and remote requests to access memory at this node.

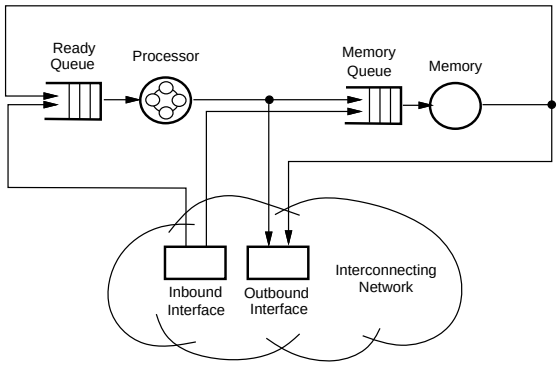

Fig.2. Outline of a single multithreaded processor.

Fig.2 shows the processor which cyclically changes the threads (4 threads in Fig.2), issuing and executing the instructions. The queue of ready threads is used whenever one of the active threads initiates a long–latency operation; such a thread is replaced by a thread selected from this queue if the queue is non–empty. When the operation initiated by a replaced thread is completed, the thread joins the queue of ready threads, waiting for another access to the processor.

2 Timed Petri Net Models

Petri nets have become a popular formalism for modeling systems that exhibit parallel and concurrent activities [10], [8]. In timed nets [15], [14], 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.

A timed Petri net model of a interleaved 4–threaded processor at the level of instruction execution is outlined in Fig.3, in which timed transitions are represented by solid bars and immediate transitions by thin bars; all transition names begin with letter "T".

Cyclic issuing of instruction from consecutive threads is represented by a "thread control" section in the left part of Fig.3, where four threads are represented by four identical sections connected in a cycle. The details of this control are discussed later on in this section. Each issued instruction corresponds to a token deposited in place $Pnxt$ (in the center of Fig.3). The execution of each issued instruction is

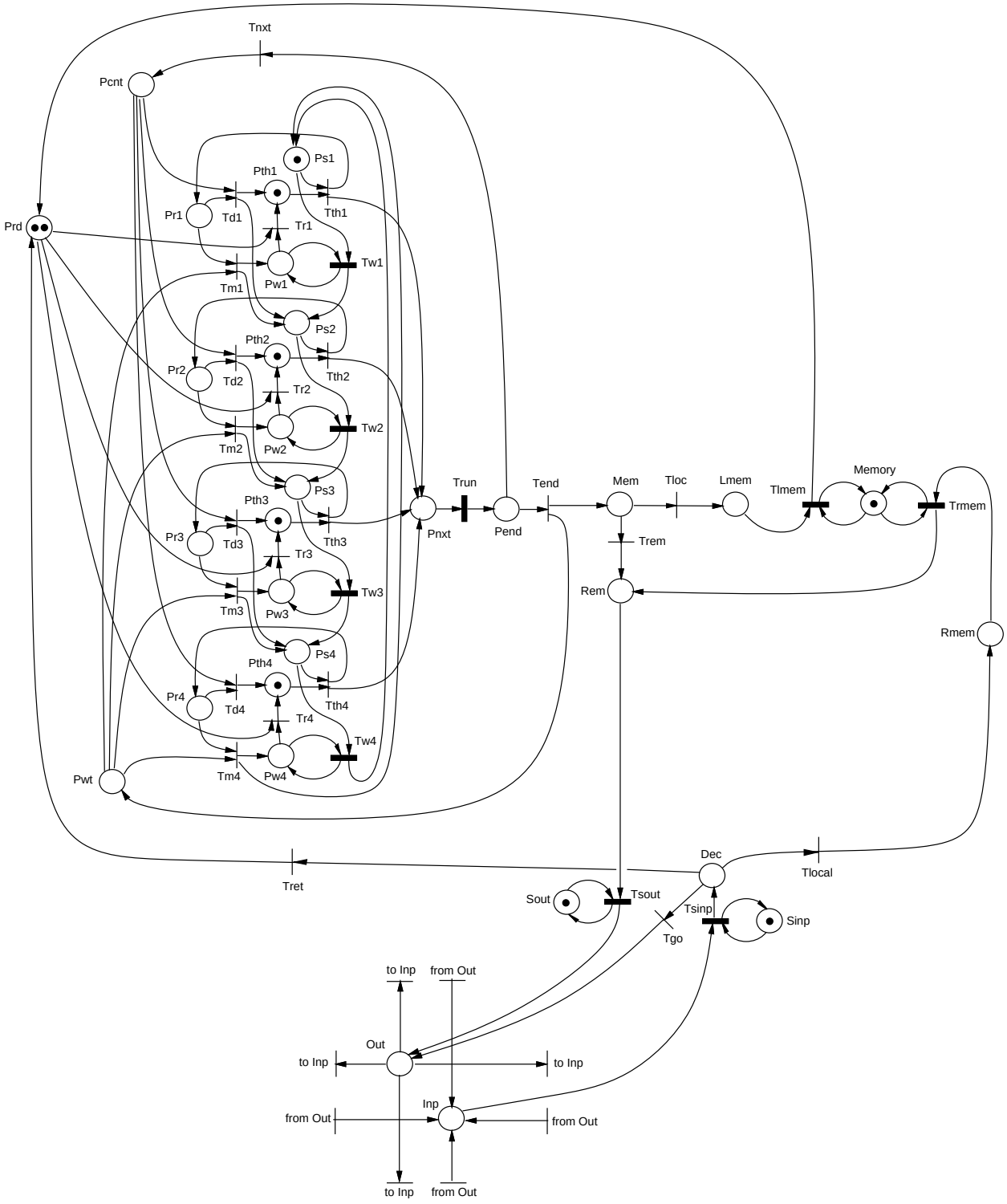

Fig.3. Instruction–level Petri net model of a interleaved multithreaded processor.

modeled by transition $Trun$ (in the center of Fig.3). $Pend$ is a free–choice place with the choice probabilities reflecting the runlength, ℓ_t, of threads (i.e., the average number of thread instructions executed between long–latency operations). 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 choosing $Tnxt$ 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 $1/\ell_t$.

The selection of $Tend$ for firing indicates a long–latency memory access issued by the current thread. The access request (to local or remote memory) is placed in Mem, and a token is also deposited in Pwt to indicate a possible thread replacement. Prd (left boundary of Fig.3) is the queue of available threads; in Fig.3 there are two enhancement threads, represented by two initial tokens assigned to this place.

Mem (in the center of Fig.3) 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 $Lmem$ where it waits for availability of *Memory*, and after accessing the memory, the thread returns to the queue of waiting threads, Prd. *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 requests 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 returning to their 'home' nodes; Tgo, which represents requests as well as responses forwarded to another node (another 'hop' in the interconnecting network); 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* becomes available. The free–choice probabilities associated with $Tret$, Tgo and $Tlocal$ characterize the interconnecting network and are determined on the basis of the average number of hops required to reach the destination node [4].

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 $Trem$).

The thread control (upper left part of Fig.3) may look somewhat complicated, but it has a regular structure repeated for each represented thread. This basic structure, for thread "2", is shown in Fig.4. The idea of this model is as follows. If the thread is active, a token is waiting in $Pth2$ for a 'control token' to appear in $Ps2$ (the marking of $Ps2$ in Fig.4 indicates that an instruction from thread "2" is going to be issued in the next processor cycle). Place $Ps2$ is an element of a 'thread ring' (in Fig.3 this ring connects $Ps1$, $Ps2$, $Ps3$, $Ps4$ and back to $Ps1$; there are several different ways connecting consecutive threads). This 'thread ring' contains a single token ($Ps1$ in Fig.3 and $Ps2$ in Fig.4).

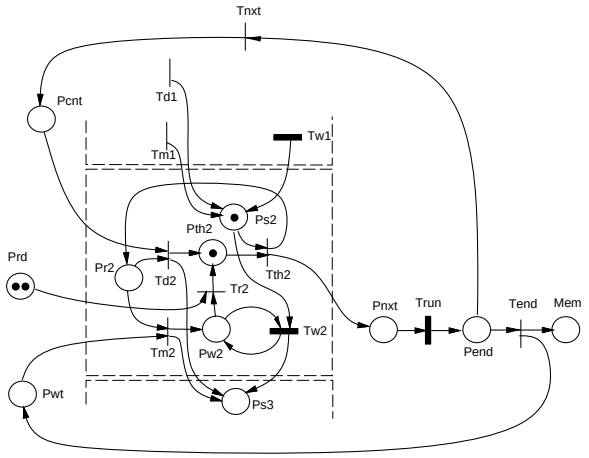

Fig.4. Single thread control section.

If the selected thread is active, the firing of $Tth2$ inserts a token in $Pnxt$ (the next instruction to be executed by $Trun$), and another token in $Pr2$. If the issued instruction does not initiate long–latency operation, the free–choice transition $Tnxt$ is fired (with the probability depending upon the thread runlength ℓ_t), and a token is deposited in $Pcnt$. This token (together with a token in $Pr2$) enables $Td2$, firing of which regenerates a token in $Pth2$ and forwards the control token to $Ps3$.

If transition $Tend$ is selected for firing rather than $Tnxt$, a long–latency memory access (local or remote) is initiated, and a token is deposited in Pwt. In this case $Tm2$ becomes enabled, and its firing inserts a token in $Pw2$ (to indicate that the thread is waiting for termination of its long-latency memory

access), and also the control token is forwarded to $Ps3$.

If the queue of ready threads, Prd, is nonempty, transition $Tr2$ becomes enabled and its firing replaces the current thread by a new one, regenerating a token in $Pth2$ (immediate transitions take precedence in firing over the timed ones), so that another instruction will be issued when thread "2" is selected again. If, however, Prd contains no tokens, the current thread remains 'waiting' for the completion of its long–latency operation (or for a ready thread entering Prd).

If a thread is 'waiting' (in $Pw2$) and a selection token appears in $Ps2$, the timed transition $Tw2$ fires and, after a unit of time (one processor cycle), deposits a control token in $Ps3$ (without issuing an instruction in this case).

The interconnecting network is characterized by two parameters, the delay of network switches, t_s, and the average number of hops, n_h, that a memory access request needs to perform to reach its destination. For a 16–processor system (shown in Fig.1), the value of this parameter, assuming uniform distribution of information over the nodes of the system, can be estimated from the (shortest) distances between pair of nodes. Since, for each node, there are 4 nodes that can be reached in 1 hop, 6 nodes that can be reach in 2 hops, 4 nodes requiring 3 hops, and just 1 node requiring 4 hops, the value of n_h is:

$$n_h \;=\; \frac{1*4+2*6+4*3+4*1}{15} \;\approx\; 2.$$

Also, it is convenient to represent all timing information in relative rather than absolute units, and the processor cycle has been assumed as the unit of time. Consequently, all temporal data are expressed in processor cycles; e.g., $t_m = 10$ means that the memory cycle time (t_m) is equal to 10 processor cycles, $t_s = 5$ means that the switch delay (t_s) is equal to 5 processor cycles.

The main parameters of the enhanced interleaved multithreading, and their typical values, are shown in Tab.1.

Table 1: Main parameters of interleaved multithreaded architectures and their typical values.

parameter	*values*
n_p – number of processors	16
n_t – number of processor's threads	4, 8
n_a – number of additional threads	0, ..., 4
ℓ_t – thread runlength	5, 10
t_p – processor cycle time	1
t_m – memory cycle time	5, 10, 20
t_s – switch delay	5, 10
n_h – average number of hops	2
p_ℓ – prob. of accessing local memory	0.1, ..., 0.9
p_r – prob. of accessing remote memory	$1 - p_\ell$

3 Performance Analysis

Performane results are obtained by simulation of the model shown in Fig.3. The simulation results can be verified by analytical performance estimated for the extreme values of p_ℓ, i.e., $p_\ell = 0$ and $p_\ell = 1$. If $p_\ell = 1$, all long–latency memory accesses are to local memory (the nodes can be analyzed in isolation one from another), and the utilization of each processor is determined by the ratio of slots used for issuing instructions to the total number of slots (used as well as not used):

$$u_p(1) \;=\; \frac{\ell_t * n_t}{\ell_t * n_t + t_m}.$$

For $p_\ell = 0$, all long–latency memory accesses are to remote memory, so the processor's utilization is estimated as:

$$u_p(0) \;=\; \frac{\ell_t * n_t}{\ell_t * n_t + t_m + 2*(n_h+1)*t_s}$$

where the additional term in the denominator describes the delays of the interconnecting network (for both directions, the request sent to the destination node, and the result of the memory access sent back to the 'home' node). The above estimates do not take queueing delays into account so they are actually upper bounds on the utilization of processors.

Although the above formulas can be refined in a number of ways, they nicely capture one of interleaved multithreading trends; increasing the number of processor's threads improves the performance of multithreaded processors.

The utilization of 4–thread processors, as a function of p_ℓ, the probability of long–latency accesses to local memory, and n_a, the number of additional threads, is shown in Fig.5 (each point on the surface shown in Fig.5 is obtained by simulating the behavior of the model shown in Fig.3, with the corresponding vaues of model parameters).

The predicted utilization of processors for pure interleaved multithreading (i.e., with zero additional threads available) for $p_\ell = 0$ is equal to 0.8 (or 80%) in this case, which is slightly higher than the utilization obtained from simulation as it does not take into account the queuing delays for memory accesses. For $p_\ell = 0$, the predicted value is equal

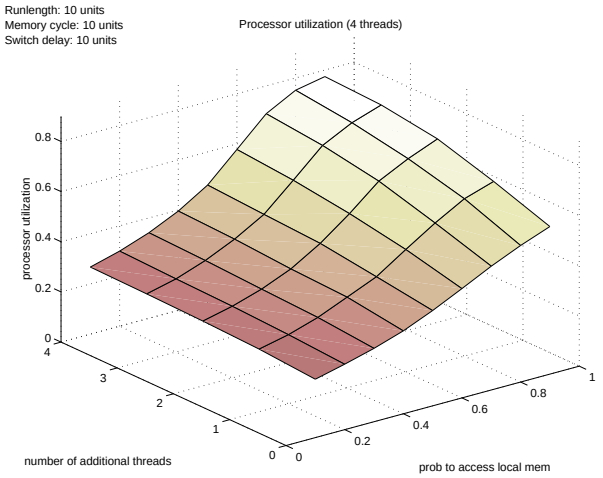

Fig.5. Processor utilization for a 4–thread system ($\ell_t = 10, t_m = 10, t_s = 10$).

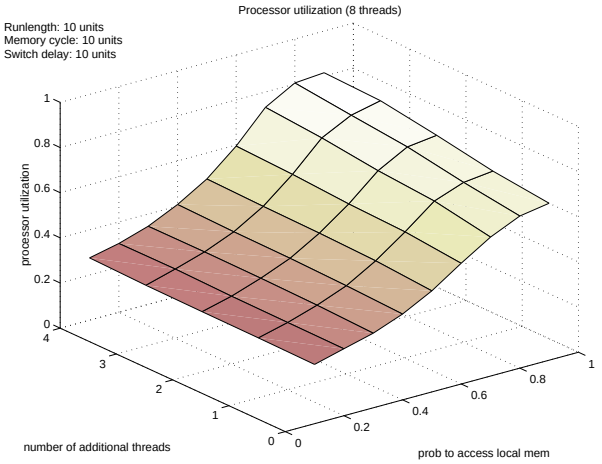

Fig.6. Processor utilization for an 8–thread system ($\ell_t = 10, t_m = 10, t_s = 10$).

to 0.36, and is also higher that than the simulation results because the prediction ignores the queuing delays in the interconnecting network.

It can be observed that the effect of enhancements is more pronounced for values of p_ℓ close to 1 (i.e., when most of accesses are to local memory); for small values of p_ℓ (in this particular case) the availability of additional threads does not have any significant effect on the utilization of processors.

The improvement of the performance, due to the availability of additional threads, is up to 30% (for 4 additional threads).

Utilization of processors for an 8–thread system are shown in Fig.6. The results are better than for the 4–thread system, but the effects of enhanced multithreading are less significant than in Fig.5.

Both Fig.5 and Fig.6 show that the performance of processors decreases quite significantly for small values of p_ℓ, i.e., when most of long–latency operations are accesses to remote memory. This is an indication that the interconnecting network may be the limiting component of this system. This is also the reason that the enhancement of multithreading has practically no effect in the region of small values of p_ℓ (or values of p_r close to 1); the interconnecting network, and more precisely, the delay of its switches, determine the performance of the system. Indeed, the utilization of the input switch, for the 4–thread system, as a function of p_ℓ and the number of available threads, is shown in Fig.7 (for the 8–thread system, the utilization of the input switch is very similar to Fig.7). The region of low utilization of processors in Fig.5 and Fig.6, i.e., the region of small values of p_ℓ, corresponds to almost 100% utilization of the input switches, which indicates that the switches are the bottleneck of this system, limiting its performance; the switches are simply too slow for this system.

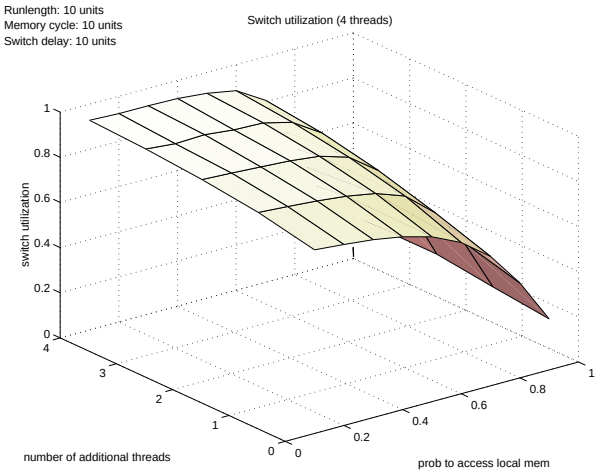

Fig.7. Switch utilization for a 4–thread system ($\ell_t = 10, t_m = 10, t_s = 10$).

Fig.8 shows the utilization of processors for the case when the switch delay is one half of that used in Fig.5 and Fig.7, while Fig.9 shows the corresponding utilization of the input switches. The utilization of processors is generally much improved, and the effects of enhanced multithreading are also more significant. Fig.9 indicates that the input switch remains the bottleneck only for very small values of p_ℓ.

Further improvement of the processor's performance (for small values of p_ℓ) can be obtained by using even faster switches or by using several paral-

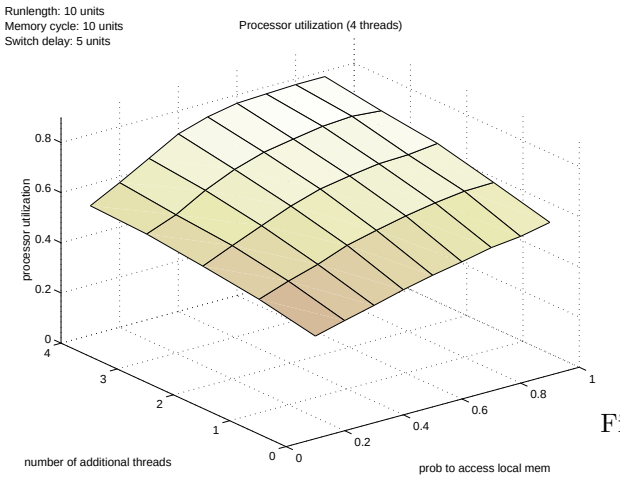

Fig.8. Processor utilization for a 4–thread systems ($\ell_t = 10, t_m = 10, t_s = 5$).

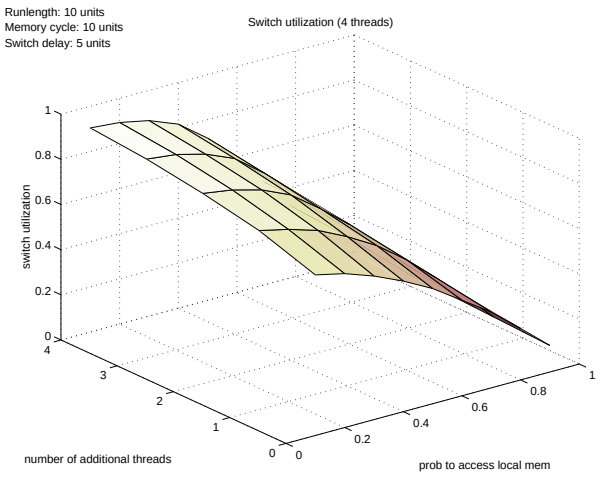

Fig.9. Switch utilization for a 4–thread systems ($\ell_t = 10, t_m = 10, t_s = 5$).

lel switches and sharing the load among them [18]; it appears that when a component (such as a switch) is the system's bottleneck, its throughput is more important for the performance of the entire system than the component's response time; in this sense, several slower parallel components can provide the same performance improvement as a single fast component [18].

Fig.10 shows the relative improvement of the processor utilization when 4 additional threads are used. The values in Fig.10 are obtained by subtracting processor utilization for $n_a = 0$ from that for $n_a = 4$ and dividing it by the utilization for $n_a = 0$. The maximum improvement of more than 40% can be achieved when approximately one half of long–latency memory accesses are local.

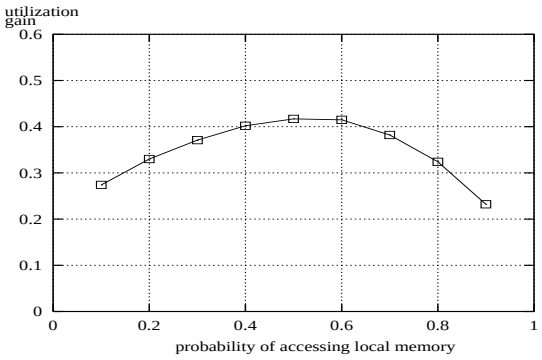

Fig.10. Utilization gain for an enhanced interleaved 4–thread system ($\ell_t = 10, t_m = 10, t_s = 5$).

The potential improvements of the performance in enhanced multithreading can be reduced by the presence of system bottlenecks performance improvement in Fig.5 is much smaller than that in Fig.8). System bottlenecks can be identified by analysis of service demands for different components of a system; the component with the maximum service demand is the bottleneck because it will reach the upper limit of its utilization first. For multithreaded multiprocessors, service demands can be considered with respect to a runlegth of a single thread, i.e., to a sequence of instructions executed (by a single thread) between consecutive long–latency operations. In such a context, the processor's service demand is equal to ℓ_t cycles, the service demand for memory (local and remote, combined) is equal to t_m, while the service demand for input switches depends on the probability of accessing remote memory, and is equal to $2 * p_r * n_h * t_s$, where the factor 2 represents the two directions of traffic (requests and results), n_h is the average number of hops in the interconnecting network, and t_s is the switch delay. For the case illustrated in Fig.5, Fig.6 and Fig.7, the switch becomes the bottleneck when its service demand is greater than that of other components, i.e., when $2{*}p_r{*}n_h > 1$ (because, in Fig.5, Fig.6 and Fig.7, $\ell_t = t_m = t_s$), and since n_h is approximately equal to 2 for a 16–processor system (Section 2), the switch is the bottleneck for $p_r > 0.25$, or $p_\ell < 0.75$; this is well illustrated in Fig.5 and Fig.6.

4 Concluding Remarks

The paper presents a timed Petri net model of fine-grain multithreaded multiprocessor system at the instruction execution level, and analyzes the effects of enhanced multithreading in which a number of

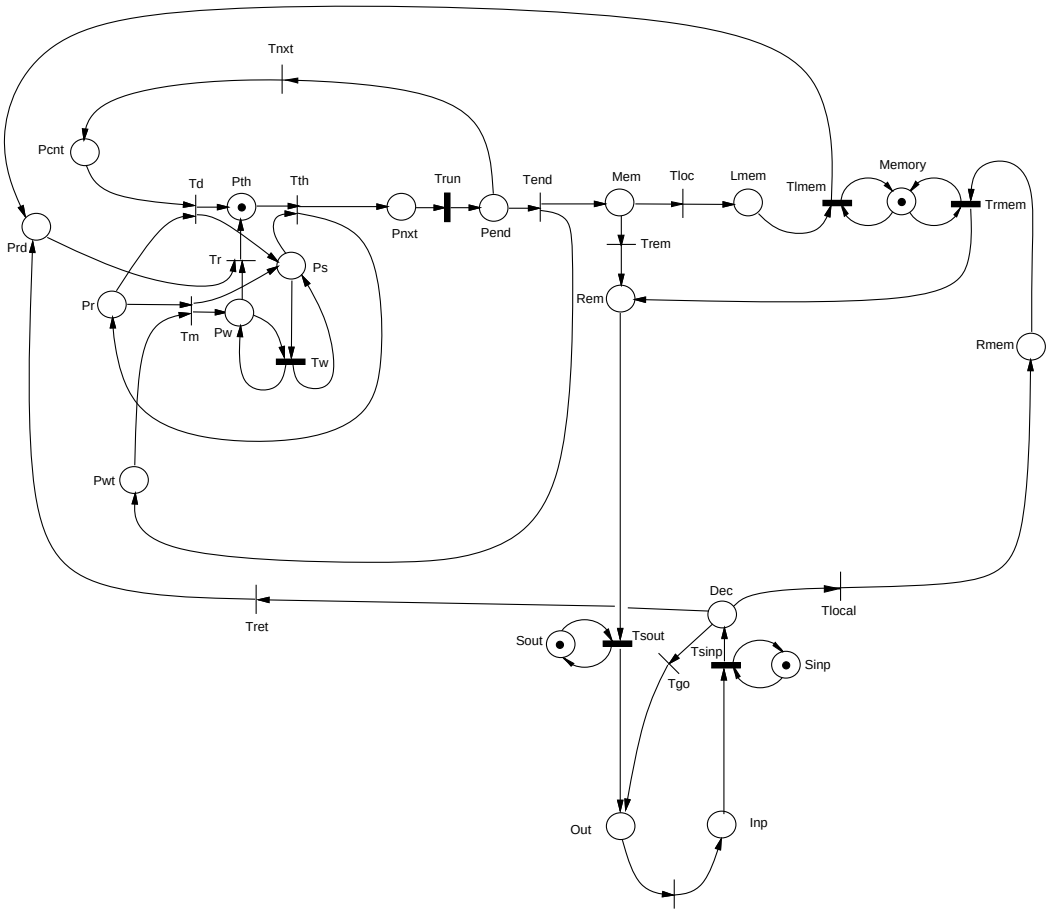

Fig.11. Instruction–level colored Petri net model of a interleaved multithreaded multiprocessor system.

additional threads are available to be used as replacements for threads which become inactive waiting for the completion of their long–latency operations. It appears that a small number of such threads can quite significantly improve the performance of the system.

The Petri net model of interleaved multithreaded architectures may seem unnecessarily complicated by incorporating several identical sections of thread control. Such repetitions can easily be eliminated by using high–level models, for example, colored Petri nets [6], [7]. The complete model of a interleaved multithreaded multiprocessor systems (for any number of processors) is shown in Fig.11.

It should be noticed that the model is much more compact than the place/transition model, but the analysis of such high-level models is usually much more difficult than analysis of place/transition nets.

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.

The processor model uses a very simple representation of memory which does not include the typical levels of memory hierarchy with different performance characteristics for different levels of this hierarchy. This simplification follows the results of an earlier study [18] which demonstrated that the results for a detailed model of memory hierarchy differ only insignificantly from results obtained for a simple model with the average values of parameters.

The results obtained for a 2–dimensional torus–like network are also valid for other interconnecting networks with the same connectivity characteristics. For example, Fig.12 shows a hypercube network for a 16–processor system that is composed

of two 8–processor subsystems. Since the average number of hops in this network is the same as in the two–dimensional network shown in Fig.1, the performance characteristics of both networks are also the same (although the two interconning networks scale in different ways).

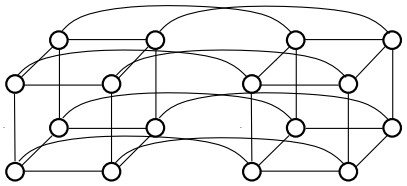

Fig.12. Outline of a 16–processor system.

Although the discussion and presented results refer to a 16–processor system, the model needs only a few small changes to represent other multiprocessor systems. For example, the only changes that need to be made to represent a 25–processor or a 36–processor system, are the values of the free–choice probabilities associated with the transitions of *Dec*. Other aspects of performance equivalence in distributed–memory multiprocessor systems are discussed in [19], [20].

A comparison of the performance of interleaved multithreading and block multithreading is presented in [16].

References

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

[2] Dennis, J.B., Gao, G.R., "Multithreaded architectures: principles, projects, and issues"; in: "Multithreaded Computer Architecture: a Summary of the State of the Art", pp.1-72, Kluwer Academic 1994.

[3] 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.

[4] 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 (PNPM'97), St. Malo, France, pp.153-162, 1997.

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

[6] Jensen, K., "Coloured Petri nets"; in: "Advanced Course on Petri Nets 1986" (Lecture Notes in Computer Science 254), pp.248-299, Springer-Verlag 1987.

[7] Kristensen, L.M., Christensen, S., Jensen, K., "The practitioner's guide to coloured Petri nets"; International Journal on Software Tools for Technology Transfer, vol.2, no.2, pp.98-132, 1998.

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

[9] Patterson, D.A., Hennessy, J.L., "Computer architecture – a qualitative approach"; Morgan Kaufman 1996.

[10] Reisig, W., "Petri nets – an introduction" (EATCS Monographs on Theoretical Computer Science 4); Springer-Verlag 1995.

[11] Sinharoy B., "Optimized thread creation for processor multithreading"; The Computer Journal, vol.40, no.6, pp.388-400, 1997.

[12] Smith, B.J., "Architecture and applications of the HEP multiprocessor computer System"; Proc. SPIE – Real-Time Signal Processing IV, vol. 298, pp. 241-248, 1981.

[13] Sohi, G.S., "Microprocessors – 10 years back, 10 years ahead"; in: "Informatics: 10 Years Back, 10 Years Ahead" (Lecture Notes in Computer Science 2000); pp.209-218, 2001.

[14] Wang, J., "Timed Petri nets"; Kluwer Academic Publ. 1998.

[15] Zuberek, W.M., "Timed Petri nets – definitions, properties and applications"; Microelectronics and Reliability, (Special Issue on Petri Nets and Related Graph Models), vol.31, no.4, pp.627-644, 1991.

[16] Zuberek, W.M., "Performance comparison of fine-grain and block multithreaded architectures"; Proc. High Performance Computing Symposium 2000, Washington, DC, pp.383-388, 2000.

[17] Zuberek, W.M., "Performance analysis of enhanced fine–grain multithreaded distributed–memory systems"; Proc. IEEE Systems, Man and Cybernetics Conf., Tucson, AZ, pp.1101-1106, 2001.

[18] Zuberek, W.M., "Analysis of performance bottlenecks in multithreaded multiprocessor systems"; Fundamenta Informaticae, vol.50, no.2, pp.223-241, 2002.

[19] Zuberek, W.M., "Performance equivalence in the simulation of multiprocessor systems"; International Journal of Simulation, vol.3, no.1-2, pp.80-88, 2002.

[20] Zuberek, W.M., "Performance analysis of fine-grain multiprocessors"; International Journal of Simulation, vol.4, no.3-4, pp.12-20, 2003.