Chapter 5

Performance Analysis of Shared-Memory Bus-Based Multiprocessors Using Timed Petri Nets

Wlodek M. Zuberek

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.75589

Abstract

In shared-memory bus-based multiprocessors, the number of processors is often limited by the (shared) bus; when the utilization of the bus approaches 100%, processors spend an increasing amount of time waiting to get access to the bus (and shared memory) and this degrades their performance. The limitations imposed by the bus depend upon many parameters, and different parameters affect the performance in different ways. This chapter uses timed Petri nets to model shared-memory bus-based multiprocessors at the instruction execution level and shows how the performance of processors and the system are affected by different modeling parameters. Discrete-event simulation of the developed net models is used to get performance results.

Keywords: shared-memory multiprocessors, bus-based multiprocessors, timed Petri nets, performance analysis, discrete-event simulation

1. Introduction

Due to continuous progress in manufacturing technologies, the performance of microprocessors has been steadily improving over several decades, doubling every 18 months (the so-called Moore's law [1]). The capacity of memory chips has also been doubling every 18 months, but the performance has been improving less than 10% per year [2]. The performance gap [3] between the processor and its memory have been doubling approximately every 6 years, and an increasing part of the processor's time is being spent on waiting for the completion of memory operations. Although multilevel cache memories are used to reduce the average latencies of memory accesses, matching the performances of the processor and the memory is an increasingly difficult task. In effect, it is often the case that more than 50% of processor

cycles are spent waiting for the completion of memory accesses [4]. A model of a pipelined processor at the instruction execution level is used in this chapter to study the mismatch of processor and memory performances.

This model of a processor is then used for performance analysis of shared-memory bus-based multiprocessors. The main objective of this analysis is to study the degradation of the processor's performance when the utilization of the (shared) bus approaches 100%. This performance degradation limits the number of processors in bus-based systems.

Modeling and analysis of shared-memory bus-based systems requires a flexible formalism that can easily handle concurrent activities as well as synchronization of different events and processes that occur in such systems. Petri nets [5, 6] are such formal models.

As formal models, Petri nets are bipartite directed graphs in which the two types of vertices represent, in a very general sense, conditions and events. An event can occur only when all conditions associated with it (represented by arcs directed to the event) are satisfied. An occurrence of an event usually satisfies some other conditions, indicated by arcs directed from the event. Hence, an occurrence of one event causes some other event (or events) to occur, and so on.

In order to study the performance aspects of systems modeled by Petri nets, the durations of modeled activities must also be taken into account. This can be done in different ways, resulting in different types of temporal nets [7]. In timed Petri nets [8], occurrence times are associated with events, and the events occur in real time (as opposed to instantaneous occurrences in other models). For timed nets with constant or exponentially distributed occurrence times, the state graph of a net is a Markov chain (or an embedded Markov chain). If the state space of a timed net is finite and reasonably small, the stationary probabilities of states can be determined by standard methods [9]. Then these stationary probabilities are used for the derivation of many performance characteristics of the model [10]. In other cases, discrete event simulation [11] is used to find the performance characteristics of a timed net.

In this chapter, timed Petri nets are used to model shared-memory bus-based multiprocessor systems. Section 2 recalls basic concepts of Petri nets and timed Petri nets. Section 3 discusses a model of a pipelined processor and its performance as a function of modeling parameters. Shared-memory bus-based systems are described and analyzed in Section 4. Section 5 concludes the chapter.

2. Timed Petri nets

In Petri nets, concurrent activities are represented by tokens that can move within a (static) graph-like structure of the net. More formally, a marked place/transition Petri net $\mathcal{M}$ is defined as a pair $\mathcal{M} = (\mathcal{N}, m_0)$, where the structure N is a bipartite directed graph, $\mathcal{N} = (P, T, A)$, with two types of vertices, a set of places P and a set of transitions T, and a set of directed arcs A connecting places with transitions and transitions with places, $A \subseteq P \times T \cup T \times P$. The initial

http://dx.doi.org/10.5772/intechopen.75589

marking function m_0 assigns nonnegative numbers of tokens to places of the net, $m_0 : P \rightarrow \{0, 1, \ldots\}$. Marked nets can be equivalently defined as $\mathcal{M} = (P, T, A, m_0)$.

A place is shared if it is connected to more than one transition. A shared place p is free-choice if the sets of places connected by directed arcs to all transitions sharing p are identical. A shared place p is (dynamically) conflict-free if for each marking reachable from the initial marking, at most one transition sharing p is enabled. If a shared place p is not free-choice and not conflict-free, the transitions sharing p are conflicting.

In timed nets [8], occurrence times are associated with transitions, and transition occurrences are real-time events, i.e., tokens are removed from input places at the beginning of the occurrence period and are deposited to the output places at the end of this period. All occurrences of enabled transitions are initiated in the same instants of time in which the transitions become enabled (although some enabled transitions may not initiate their occurrences). If, during the occurrence period of a transition, the transition becomes enabled again, a new, independent occurrence can be initiated, which will overlap with the other occurrence(s). There is no limit on the number of simultaneous occurrences of the same transition (sometimes this is called infinite occurrence semantics). Similarly, if a transition is enabled "several times" (i.e., it remains enabled after initiating an occurrence), it may start several independent occurrences in the same time instant.

More formally, a timed Petri net is a triple, $\mathcal{T} = (\mathcal{M}, c, f)$, where $\mathcal{M}$ is a marked net, c is a choice function which assigns probabilities to transitions in free-choice classes, or relative frequencies of occurrences to conflicting transitions, $c \rightarrow [0, 1]$, and f is a timing function, which assigns an (average) occurrence time to each transition of the net, $f : T \rightarrow \mathbf{R}^+$, where $\mathbf{R}^+$ is the set of nonnegative real numbers.

The occurrence times of transitions can be either deterministic or stochastic (i.e., described by some probability distribution function). In the first case, the corresponding timed nets are referred to as D-timed nets [12]; in the second, for the (negative) exponential distribution of occurrence times, the nets are called M-timed nets (Markovian nets) [13]. In both cases, the concepts of state and state transitions have been formally defined and used in the derivation of different performance characteristics of the model. In simulation applications, other distributions can also be used, for example, the uniform distribution (U-timed nets) is sometimes a convenient option. In timed Petri nets, different distributions can be associated with different transitions in the same model providing flexibility that is used in simulation examples that follow.

In timed nets, the occurrence times of some transitions may be equal to zero, which means that the occurrences are instantaneous; all such transitions are called immediate (while the others are called timed). Since immediate transitions have no tangible effects on the (timed) behavior of the model, it is convenient to "split" the set of transitions into two parts, the set of immediate and the set of timed transitions, and to first perform all occurrences of the (enabled) immediate transitions, and then (still in the same time instant), when no more immediate transitions are enabled, to start the occurrences of (enabled) timed transitions. It should be noted that such a convention effectively introduces the priority of immediate transitions over

the timed ones, so the conflicts of immediate and timed transitions are not allowed in timed nets. Detailed characterization of the behavior of timed nets with immediate and timed transitions is given in [8].

3. Pipelined processors

A timed Petri net model of a pipelined processor [14] at the level of instruction execution is shown in **Figure 1** (as usual, timed transitions are represented by solid bars, and immediate transitions by thin bars). It is assumed that the first level cache does not delay the processor, while cache misses (at the first level cache) introduce a delay of t_c processor cycles. For simplicity, only two levels of cache memory are represented in the model; it appears that such a simplification does not affect the results in a significant way [15].

Place *Pnxt* is marked when the processor is ready to execute the next instruction. *Pnxt* is a free-choice place with three possible outcomes that model issuing an instruction without any further delay (*Ts0* with the choice probability p_{s0}), a single-cycle pipeline stall (modeled by *Td1* with the choice probability p_{s1} associated with *Ts1*), and a two-cycle pipeline stall (modeled by *Td2* and then *Td1* with the choice probability p_{s2} assigned to *Ts2*). Other pipeline stalls could be represented in a similar way, if needed.

Marked place *Cont* indicates that an instruction is ready to be issued to the execution pipeline. It is assumed that once the instruction enters the pipeline, it will progress through all the stages and, eventually, leave the pipeline. Since the details of pipeline implementation are not important for performance analysis of the processor, they are not represented here. Only the first stage of the execution pipeline is shown as timed transition *Trun*.

Done is another free-choice place which determines if the executing instruction results in a cache miss or not. Transition *Tnxt* occurs (with the corresponding probability) if cache miss does not occur and the processor can continue fetching and issuing instructions. Cache miss is represented by *Tsel*. The choice probability associated with *Tsel* determines the instruction

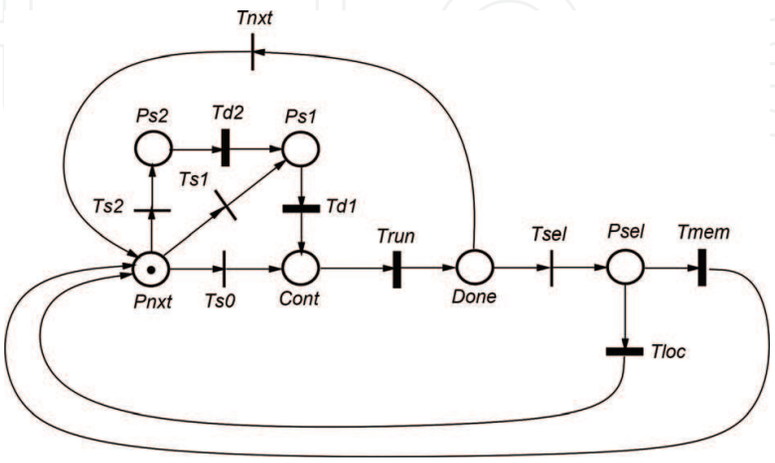

Figure 1. Instruction-level Petri net model of a pipelined processor.

http://dx.doi.org/10.5772/intechopen.75589

runlength, n_ℓ, i.e., the average number of instructions between two consecutive cache misses; if this choice probability is equal to 0.1, the runlength is equal to 10; if it is equal to 0.2, the runlength is 5; and so on.

Psel is another free-choice place; it models the hits and misses of the second-level cache. The probability associated with transition *Tloc* represents the hit ratio of the second-level cache (the occurrence time of *Tloc* is the average access time to the second-level cache, t_c) while the miss ratio is associated with transition *Tmem* which represents accesses to the main memory (with the occurrence time t_m).

Typical values of modeling parameters used in this chapter are shown in **Table 1**.

All temporal data in **Table 1** (i.e., cache and memory access times) are in processor cycles.

Processor utilization as a function of h_1, the hit rate of the first-level cache, is shown in **Figure 2** for two values of the second-level cache access time, $t_c = 5$, and $t_c = 10$. It should not be

Symbol	Parameter	Value
h_1	First-level cache hit rate	0.9
h_2	Second-level cache hit rate	0.8
t_p	First-level cache access time	1
t_c	Second-level cache access time	5
t_m	Main memory access time	25
p_{s1}	Prob. of one-cycle pipeline stall	0.1
p_{s2}	Prob. of two-cycle pipeline stall	0.05

Table 1. Modeling parameters and their typical values.

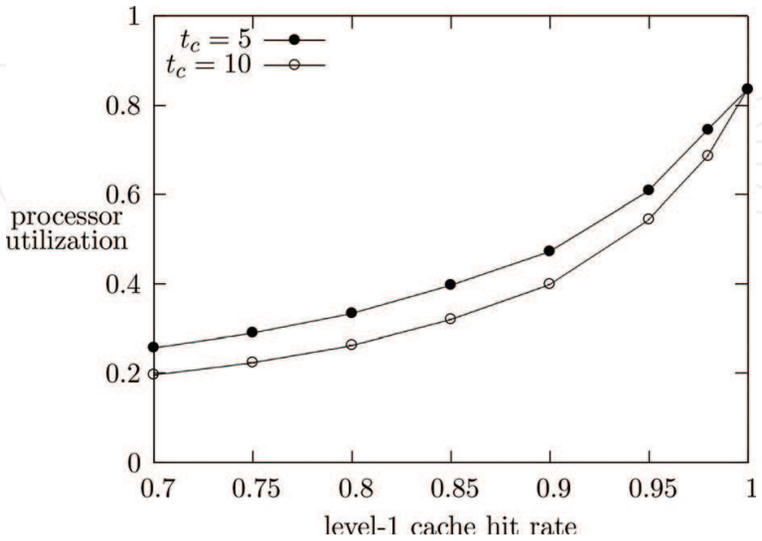

Figure 2. Processor utilization as a function of first-level cache hit rate for $h_2 = 0.8$, $p_s = 0.2$.

surprising that processor utilization is quite sensitive to the values of h_1, but is much less sensitive to the values of t_c.

Processor utilization as a function of h_2, the hit rate of the second-level cache, is shown in **Figure 3** for two values of the main memory access time, $t_m = 25$ and $t_m = 50$. Processor utilization is rather insensitive to values of h_2, and does not change much with t_m.

Processor utilization as a function of the probability of pipeline stalls $p_s = p_{s1} + 2p_{s2}$ is shown in **Figure 4** for three combinations of values of t_c and t_m.

Again, processor utilization is rather insensitive to the probability of pipeline stalls as well as the values of t_c and t_m.

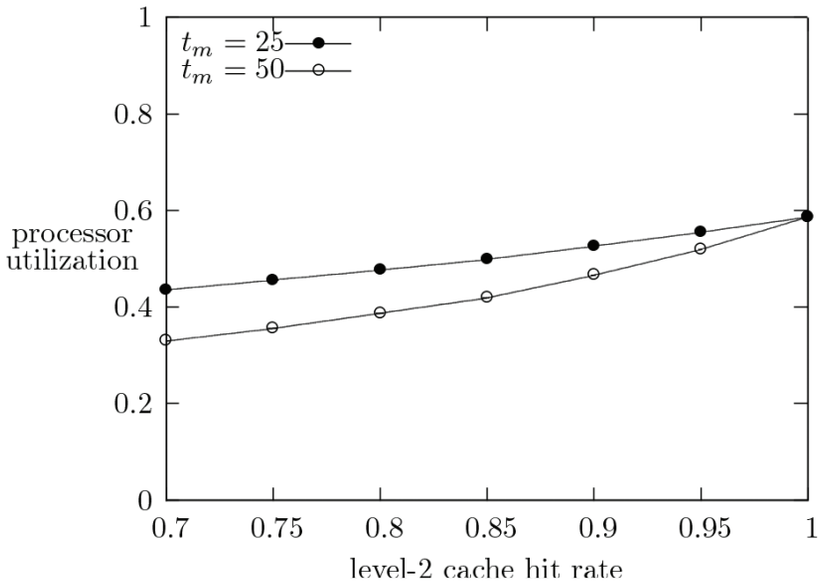

Figure 3. Processor utilization as a function of second-level cache hit rate for $h_1 = 0.9$, $p_s = 0.2$.

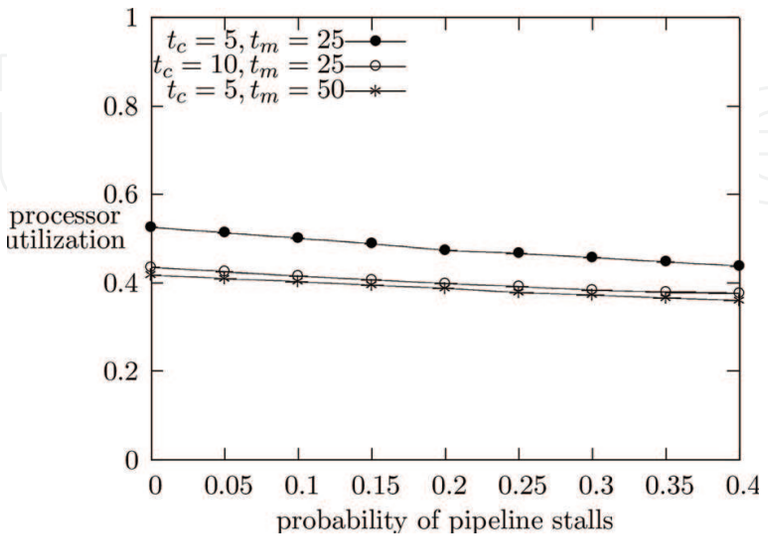

Figure 4. Processor utilization as a function of probability of pipeline stalls for $h_1 = 0.9$, $h_2 = 0.8$.

http://dx.doi.org/10.5772/intechopen.75589

For pipelined processors shown in **Figure 1**, processor utilization can be estimated using the following formula:

$$u_p = \frac{1}{1 + p_{s1} + 2{*}p_{s2} + (1 - h_1){*}(t_c + (1 - h_2){*}t_m)}. \tag{1}$$

For the values of modeling parameters shown in **Table 1**, processor utilization is:

$$u_p = \frac{1}{1 + 0.1 + 0.1 + 0.1{*}(5 + 0.2{*}25)} \approx 0.45. \tag{2}$$

The estimated values agree quite well with the values shown in **Figures 2–4**.

4. Shared-memory bus–based systems

An outline of a shared-memory bus-based multiprocessor is shown in **Figure 5**. The system is composed of n identical processors, which access the shared memory using a system bus. To reduce the average access time to the shared memory, the processors use (multilevel) cache memories. It is assumed that memory consistency is provided by a cache coherence mechanism [16], which usually increases the miss ratio of accessing caches (and is otherwise not represented in the model).

A timed Petri net model of a shared-memory bus-based multiprocessor is shown in **Figure 6**. It contains models of n processors (only two are shown in **Figure 6**), which are copies of the model shown in **Figure 1** except for the main memory (transition *Tmem*) which becomes shared memory in **Figure 6**. The remaining part of **Figure 6** is modeling the bus that coordinates accesses of processors to the shared memory.

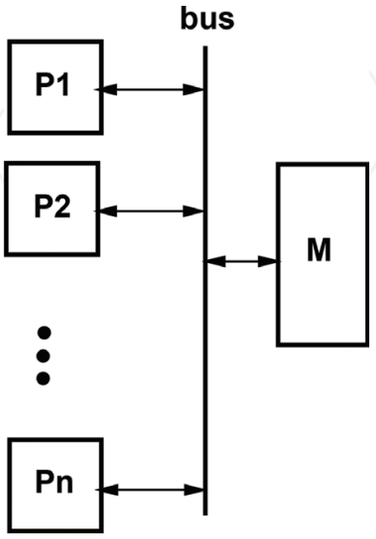

Figure 5. A shared-memory bus-based multiprocessor.

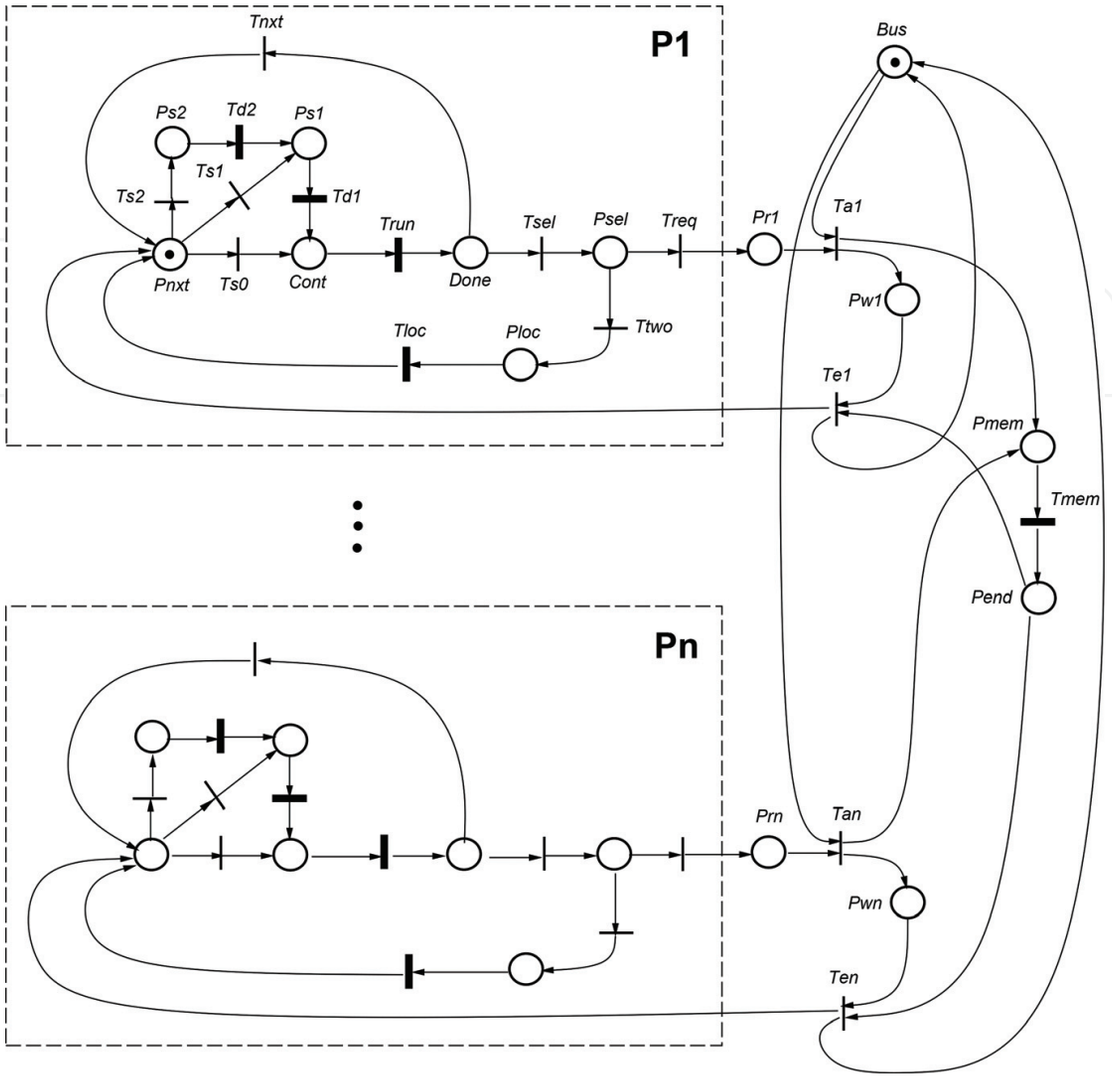

Figure 6. A timed Petri net model of bus-based shared-memory multiprocessor.

When a processor i, $i = 1, \ldots, n$, requests an access to shared memory, place *Pri* becomes marked. If the bus is available (i.e., if place *Bus* is marked), the occurrence of transition *Tai* indicates that processor i begins its access to shared memory. Transitions *Ta*1, ..., *Tan* constitute a conflict class with a fair resolution of conflicts (i.e., all conflicting processors have the same probability of being selected for accessing memory). In real systems, accessing the shared bus is often based on priorities assigned to processors; such priorities could easily be represented using inhibitor arcs in Petri nets.

Place *Pmem* collects memory access requests from all processors (occurrences of transitions *Tai*). The end of memory access (i.e., the end of the occurrence of *Tmem*) is indicated by an occurrence of transition *Tei* of the processor which initiated memory access. The occurrence of *Tei* also returns a token to *Bus*, allowing another access to shared memory to be executed.

Figure 7 shows the utilization of processors and the bus as functions of the number of processors in a shared-memory system for the values of modeling parameters shown in **Table 1**.

http://dx.doi.org/10.5772/intechopen.75589

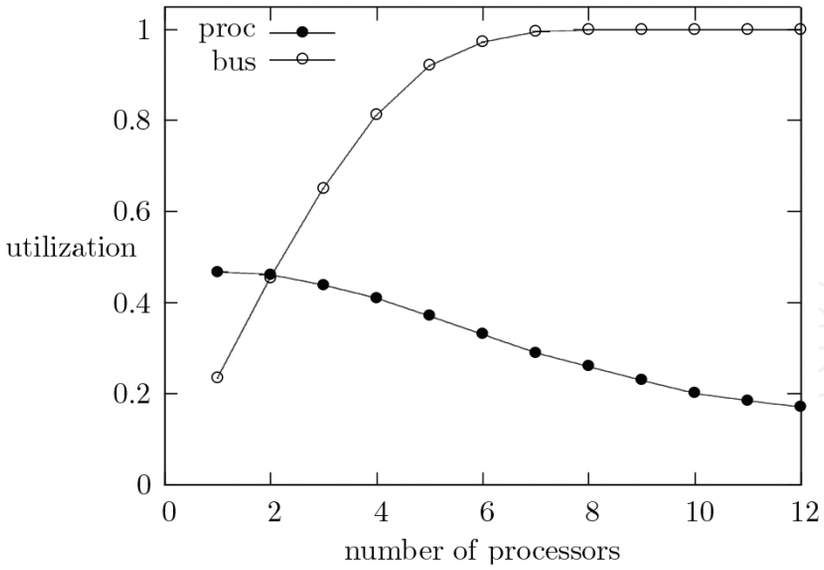

Figure 7. Processor and bus utilization as functions of the number of processors for $h_1 = 0.9$, $h_2 = 0.8$, $p_s = 0.2$.

In **Figure 7**, the bus utilization approaches 100% for about five processors. Moreover, the degradation of processors' performance due to increasing waiting times for accessing the bus (and shared memory) is well illustrated in **Figure 7**.

The average waiting time (in processor cycles) of accessing shared memory (i.e., the times from requesting memory access in place *Pri* to granting this access by an occurrence of *Tai*) is shown in **Figure 8** as a function of the number of processors in the system.

Figure 8 shows that the waiting times increase almost linearly with the number of processors when this number is greater than 5, i.e., when the bus (and shared memory) is utilized in almost 100%.

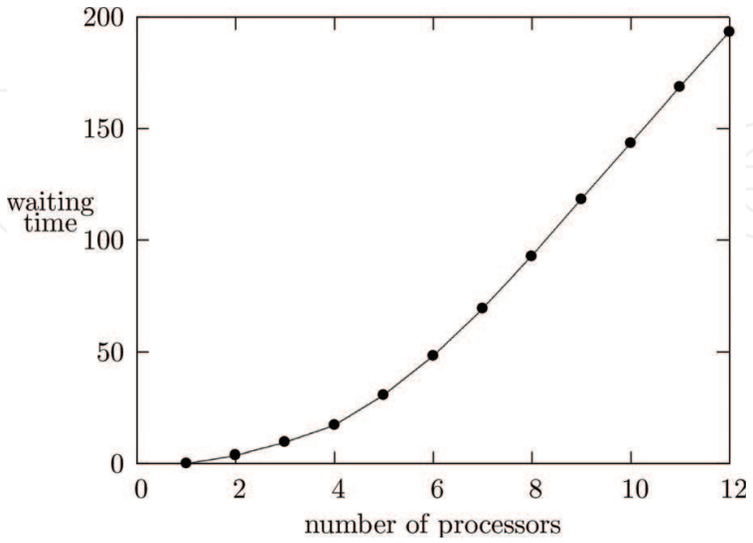

Figure 8. The average waiting time for accessing shared memory as a function of the number of processors for $h_1 = 0.9$, $h_2 = 0.8$, $p_s = 0.2$.

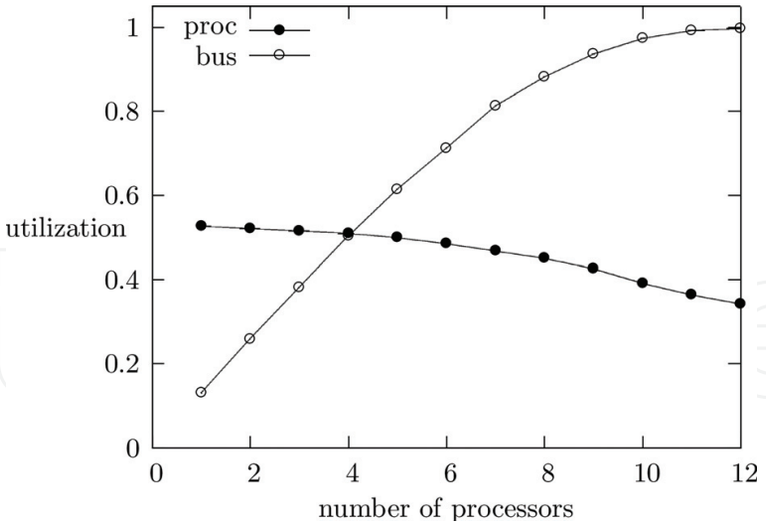

Figure 9. Processor and bus utilization as functions of the number of processors for $h_1 = 0.9$, $h_2 = 0.9$, $p_s = 0.2$.

If the value of the second-level cache hit rate, h_2, increases (and the other parameters do not change), the number of accesses to main memory is reduced, so that the performances of processors and the whole system improve. **Figure 9** shows the utilization of processors and the bus as functions of the number of processors in the system for $h_2 = 0.9$. It also shows that the reduced (in comparison with **Figure 7**) utilization of the bus allows the increase of the number of processors without significant degradation of their performance.

By a similar argument, reduced hit rate at the first-level cache, h_1, increases the number of accesses to the second-level cache as well as to the main memory, and this results in reduced performance of the system. **Figure 10** shows the utilization of processors and the bus as functions of the number of processors in the system for $h_1 = 0.8$. It provides a good illustration of the degradation of performance when compared with **Figure 7**.

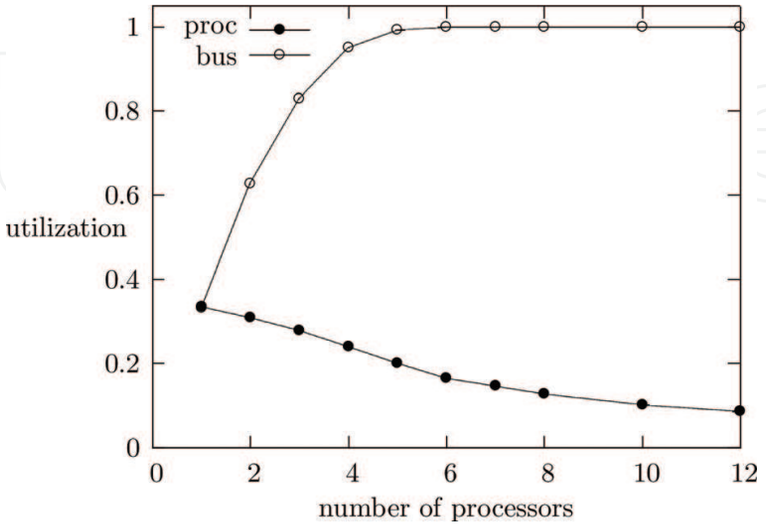

Figure 10. Processor and bus utilization as functions of the number of processors for $h_1 = 0.8$, $h_2 = 0.8$, $p_s = 0.2$.

http://dx.doi.org/10.5772/intechopen.75589

The number of processors for which the bus is used to almost 100% can be estimated by the following formula:

$$n_p = \frac{1 + p_{s1} + 2{*}p_{s2} + (1 - h_1){*}(t_c + (1 - h_2){*}t_m)}{(1 - h_1){*}(1 - h_2){*}t_m}. \tag{3}$$

For the case shown in **Figure 7**, this number is:

$$\frac{1 + 0.1 + 0.1 + 0.1{*}(5 + 0.2{*}25)}{0.1{*}0.2{*}25} = \frac{1.2 + 1.0}{0.5} = 4.4. \tag{4}$$

For the case shown in **Figure 9**, this value is 8.2.

There are several ways in which the number of processors can be increased in bus-based systems without sacrificing the processors' performance. The simplest approach is to introduce the second bus which allows two concurrent accesses to shared memory, provided the memory is dual port (it allows two concurrent accesses). **Figure 11** outlines a dual bus shared-memory system.

Petri net model of a dual bus system is the same as in **Figure 6**; the only difference is the initial marking of place *Bus*, which now requires two tokens to represent two concurrent accesses to shared memory. It should be observed observed that, for a small number of processors, the utilization of each bus in **Figure 12** is one half of that in **Figure 7**, and also the number of processors that can be used in such a dual bus system without degradation of their performance is twice as large as in a single bus system (**Figure 7**).

The results shown in **Figure 12** are very similar to those shown in **Figure 9**. The second bus in a shared-memory system allows to perform two concurrent accesses to the shared memory. From a single processor's performance point of view, this effect is similar to reducing two

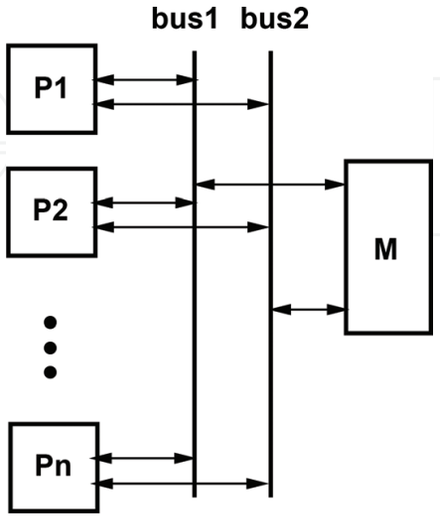

Figure 11. A dual bus shared-memory multiprocessor.

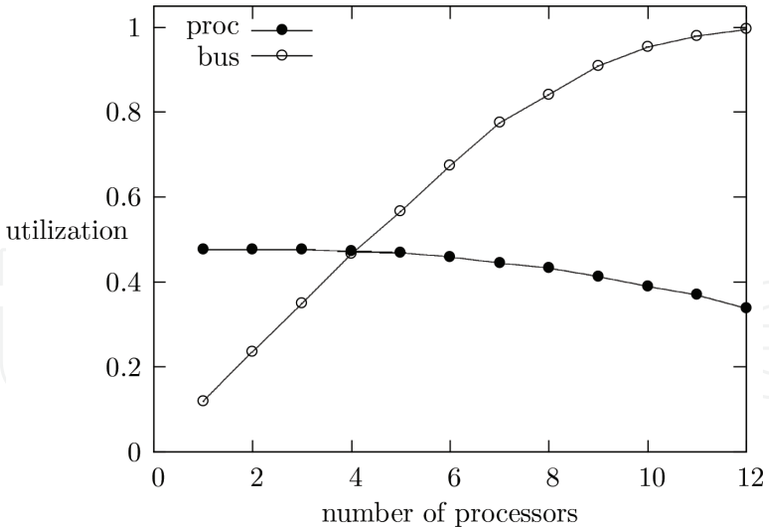

Figure 12. Processor and bus utilization as functions of the number of processors—dual bus system with $h_1 = 0.9$, $h_2 = 0.8$, $p_s = 0.2$.

times the number of accesses to the shared memory, and this is the effect of reducing two times the miss rate for the second-level cache (which is shown in **Figure 9**).

If dual port memory cannot be used, the shared memory can be split into several independent modules which can be accessed concurrently by the processors provided that the bus is also split into sections associated with each module, with processors accessing all such sections, as shown in **Figure 13** for four independent memory modules. The main difference between a multibus system (**Figure 11**) and a system with split bus is in accessing the shared memory; in a multiple bus system, the whole shared memory is accessed by each bus, while in a split bus system (**Figure 13**), each section of the bus accesses only one memory module. In the system

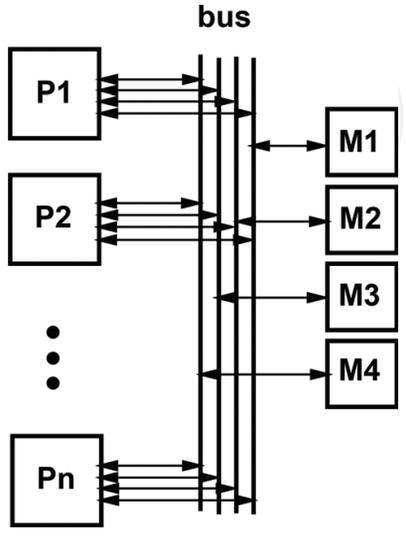

Figure 13. A shared-memory multiprocessor with multiple memory modules.

http://dx.doi.org/10.5772/intechopen.75589

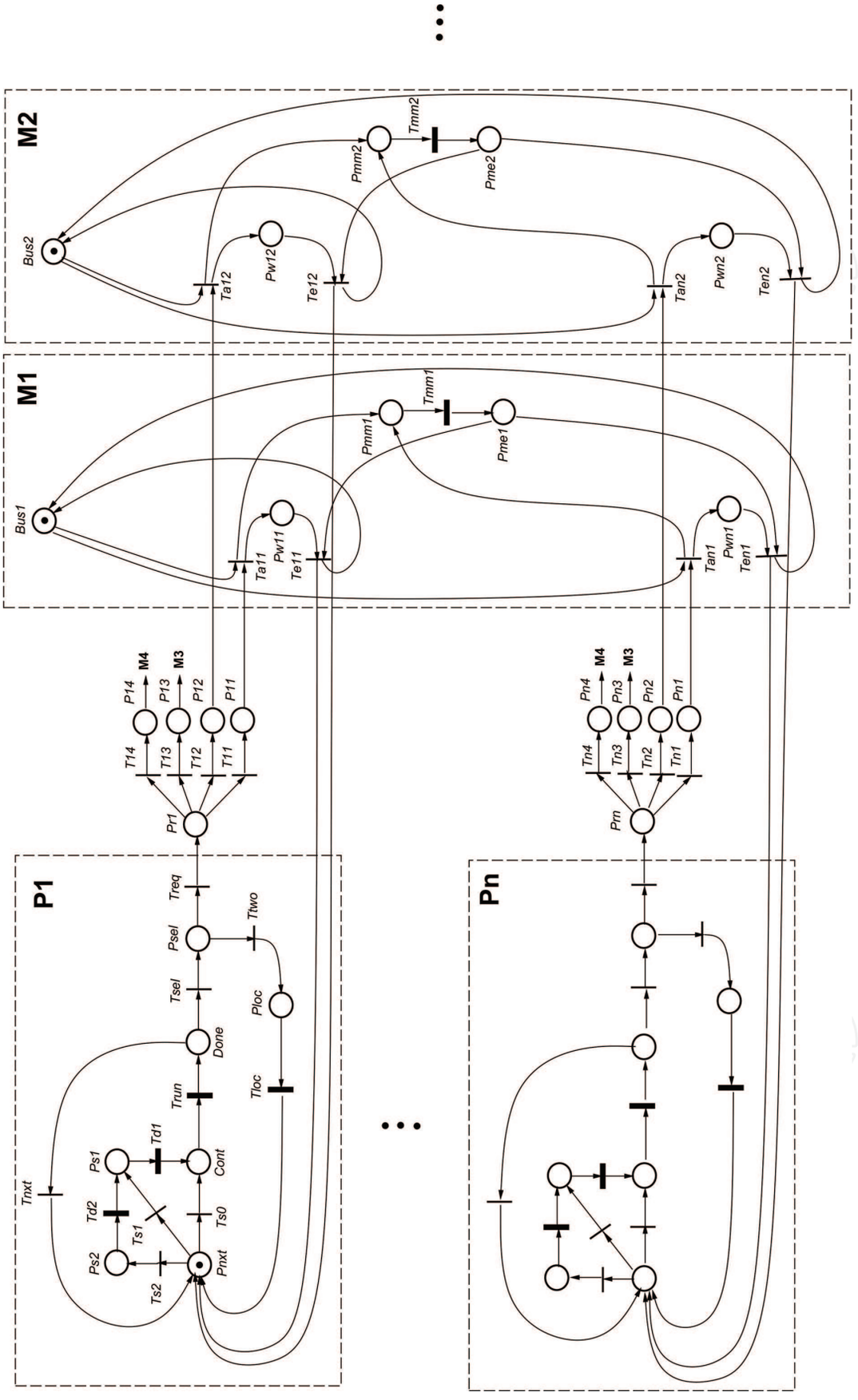

Figure 14. Petri net model of a shared-memory multiprocessor with multiple memory modules.

shown in **Figure 13**, up to four (the number of memory modules) memory accesses can be performed concurrently, but if two (or more) processors request access to the same memory module, the requests are served one after another.

Petri net model of a system outlined in **Figure 13** is shown in **Figure 14** where only two processors and two memory modules are detailed.

In **Figure 14**, there is a free-choice place *Pri* for each processor *i*, $i = 1, \ldots, n$. This free-choice place selects the requested memory module by transitions *Tij*, $j = 1, \ldots, 4$, and forwards the memory access request to the selected memory module (place *Pij*). If the selected module is available, i.e., if place *Busj* is marked, the access to shared memory is initiated by the occurrence of *Taij*. When this memory access is completed, the occurrence of *Teij* releases the memory modules (by returning a token to *Busj*) and resumes instruction execution in the processor that requested the memory access.

If memory module is not available when it is requested, the memory access is delayed (in *Pij*) until the requested module becomes available.

It is possible that more than one processor becomes waiting for the same memory module. The selection of the processor which will get access first is random with the same probability assigned to all waiting processors. In real systems, there is usually some priority scheme that determines the order in which the waiting processors access the bus. Such a priority scheme could easily be modeled if it is needed (for example, for studying the starvation effect which can be created when the system is overloaded).

In **Figure 14**, the selection of memory modules is random, with the same probabilities for all modules. If this policy is not realistic, a different memory accessing policy can be implemented,

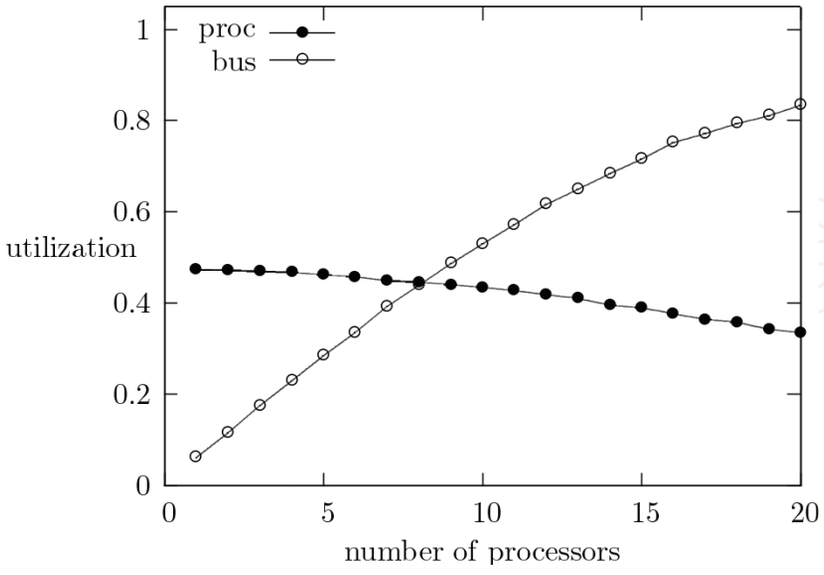

Figure 15. Processor and bus utilization as functions of the number of processors—system with four memory modules and $h_1 = 0.9$, $h_2 = 0.8$, $p_s = 0.2$.

for example, the probabilities of accessing consecutive memory modules by each processor could be used to model sequential processing of large arrays, and so on.

Figure 15 shows the utilization of processors and busses as functions of the number of processors in a system outlined in **Figure 13**.

In **Figure 15**, even for 20 processors, the average utilization of the bus is close to 80%, so the system can accommodate more processors.

5. Concluding remarks

The chapter uses timed Petri nets to model shared-memory bus-based architectures at the level of instruction execution to study the effects of modeling parameters on the performance of the system. The models are rather simple with straightforward representation of modeling parameters.

Performance results presented in this chapter have been obtained by the simulation of developed Petri net models. However, the model shown in **Figure 7** has only 10 states, so its analytical solution (for different values of modeling parameters) can be easily obtained and compared with simulation results to verify their accuracy. **Table 2** shows such a comparison of processor utilization for several combinations of parameters h_1 and h_2. In all cases, the simulation-based results are very close to the analytical ones.

The models of multiprocessor systems are usually composed of many copies of the same submodel of a processor and possibly other elements of the system. Colored Petri nets [17] can significantly simplify such models by eliminating copies of similar subsystems. Analysis of colored Petri nets is, however, much more complex than that of ordinary Petri nets.

Finally, it should be noted that the performance of real-life multiprocessor systems very rarely can be described by a set of parameters that remain stable for any significant period of time. The basic parameters like the hit rates depend upon the executed programs as well as their data, and can change very quickly in a significant way. Consequently, the characteristics presented in this chapter can only be used as some insight into the complex behavior of multiprocessor systems.

h_1	h_2	Simulated results	Analytical results
0.8	0.8	0.3341	0.3333
0.8	0.9	0.3846	0.3846
0.9	0.8	0.4763	0.4762
0.9	0.9	0.5255	0.5263

Table 2. A comparison of simulation and analytical results.

Author details

Wlodek M. Zuberek

Address all correspondence to: wlodek@mun.ca

Department of Computer Science, Memorial University, Canada

References

[1] Hamilton S. Taking Moore's law into the next century. IEEE Computer. 1999;**32**(1):43-48

[2] Patterson DA, Hennessy JL. Computer Architecture—A Quantitative Approach. 4th ed. San Mateo, CA: Morgan Kaufmann; 2006

[3] Wilkes MV. The memory gap and the future of high-performance memories. ACM Architecture News. 2001;**29**(1):2-7

[4] Mutlu O, Stark J, Wilkerson C, Patt YN. Runahead execution: An effective alternative to large instruction windows. IEEE Micro. 2003;**23**(6):20-25

[5] Murata T. Petri nets: Properties, analysis and applications. Proceedings of IEEE. 1989; **77**(4):541-580

[6] Reisig W. Petri nets—An introduction (EATCS Monographs on Theoretical Computer Science. Vol. 4). New York, NY: Springer-Verlag; 1985

[7] Popova-Zeugmann L. Time and Petri Nets. Berlin, Heidelberg: Springer-Verlag; 2013

[8] Zuberek WM. Timed petri nets—Definitions, properties and applications. Microelectronics and Reliability (Special Issue on Petri Nets and Related Graph Models). 1991;**31**(4):627-644

[9] Allen AA. Probability, Statistics and Queueing Theory with Computer Science Applications. 2nd ed. San Diego, CA: Academic Press; 1991

[10] Jain R. The Art of Computer Systems Performance Analysis. New York, NY: Wiley Interscience; 1991

[11] Pooch UW, Wall JA. Discrete Event Simulation. Boca Raton, FL: CRC Press; 1993

[12] Zuberek WM. D-timed petri nets and modelling of timeouts and protocols. Transactions of the Society for Computer Simulation. 1987;**4**(4):331-357

[13] Zuberek WM. M-timed petri nets, priorities, preemptions, and performance evaluation of systems. In: Advances in Petri Nets 1985 (LNCS 222). Berlin, Heidelberg: Springer-Verlag; 1986. pp. 478-498

[14] Ramamoorthy CV, Li HF. Pipeline architecture. ACM Computing Surveys. 1977;**9**(1):61-102

[15] Zuberek WM. Modeling and analysis of simultaneous multithreading. In: Proceedings of the 14th International Conference on Analytical and Stochastic Modeling Techniques and Applications (ASMTA-07), a part of the 21st European Conference on Modeling and Simulation (ECMS'07); Prague, Czech Republic; 2007. pp. 115-120

[16] Suh T, Lee H-HS, Blough DM. Integrating cache coherence protocols for heterogeneous multiprocessor system. Part 2. IEEE Micro. 2004;**24**(5):55-69

[17] Jensen K, Kristensen LM. Coloured Petri Nets—Modeling and Validation of Concurrent Systems. Berlin, Heidelberg: Springer-Verlag; 2009

IntechOpen