Scientific Research Publishing

Journal of Computer and Communications, 2018, 6, 1-14
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

# Timed Petri Net Models of Shared-Memory Bus-Based Multiprocessors

**Wlodek M. Zuberek**

Department of Computer Science, Memorial University, St. John's, Canada
Email: wlodek@mun.ca

**How to cite this paper:** Zuberek, W.M. (2018) Timed Petri Net Models of Shared-Memory Bus-Based Multiprocessors. *Journal of Computer and Communications*, **6**, 1-14.
https://doi.org/10.4236/jcc.2018.610001

**Received:** August 14, 2018
**Accepted:** October 7, 2018
**Published:** October 10, 2018





## Abstract

In shared-memory bus-based multiprocessors, when the number of processors grows, the processors spend an increasing amount of time waiting for access to the bus (and shared memory). This contention reduces the performance of processors and imposes a limitation of the number of processors that can be used efficiently in bus-based systems. Since the multiprocessor's performance depends upon many parameters which affect the performance in different ways, timed Petri nets are used to model shared-memory bus-based multiprocessors at the instruction execution level, and the developed models are used to study how the performance of processors changes with the number of processors in the system. The results illustrate very well the restriction on the number of processors imposed by the shared bus. All performance characteristics presented in this paper are obtained by discrete-event simulation of Petri net models.

## Keyword

Shared-Memory Multiprocessors, Bus-Based Multiprocessors, Timed Petri Nets, Discrete-Event Simulation

## 1. Introduction

More than 50 years ago Gordon Moore predicted that the number of transistors on microprocessor chips will double every 18 to 24 months (the so called Moore's law [1]). This prediction has proven remarkably robust; although the end of Moore's law was supposed to occur several times in the past [2], the trend seems to be holding, resulting in impressive improvement of the performance of microprocessors. The capacity of memory chips has also been doubling every 18 months or so, but their performance has been improving less than 10% per year [3]. The performance gap [4] between the processor and its memory has been

doubling approximately every six 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 [5].

Shared-memory bus-based multiprocessors are typically composed of a number of (identical) processors with their local cache memories and a shared memory (at a higher level of memory hierarchy). As the number of processors grows, the processors spent an increasing amount of time waiting for getting access to the bus (and shared memory) which reduces their performance. The limitations imposed by the bus depend upon many parameters, and different parameters affect the performance in different ways. In order to study the influence of different parameters on the performance of the system, a model of a pipelined processor at the instruction execution level is proposed and is used for performance analysis of shared-memory bus-based multiprocessors. The main objective of this analysis is to study the reduction of processor's performance when the utilization of the (shared) bus approaches 100%.

A flexible formalism that can easily handle concurrent activities as well as synchronization of different events and processes that occur in shared-memory bus-based systems is needed for modeling and performance analysis of bus-based multiprocessors. Petri nets [6] [7] are such formal models. In order to study 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 [8]. In timed Petri nets [9], occurrence times are associated with events, and the events occur in real-time (as opposed to instantaneous occurrences in other models).

In this paper, timed Petri nets are used to model shared-memory bus-based multiprocessor systems at the level of instruction execution. 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 paper.

This paper is an extension of previous work on performance analysis of shared-memory bus-based multiprocessors using timed Petri nets [10]. The new contributions include a refined model of pipelined processors which captures parameters of physical systems in a better way than previously. Also, much simpler models of multiprocessor systems are presented in this paper with performance characteristics that are consistent with previous models.

## 2. Timed Petri Nets

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. So, an occurrence of one event causes some other event (or events) to occur, and so on.

Tokens which can move within a (static) graph-like structure of the net are used in Petri nets to represent concurrent activities. More formally, a marked place/transition Petri net $\mathcal{M}$ is defined as a pair $\mathcal{M}=(\mathcal{N},m_0)$, where the structure $\mathcal{N}$ is a bipartite directed graph, $\mathcal{N}=(P,T,A)$, with two types of vertices, a set of places $P$ (representing conditions) and a set of transitions $T$ (representing events), and a set of directed arcs $A$ connecting places with transitions and transitions with places, $A\subseteq T\times P\bigcup P\times T$. The initial marking function $m_0$ assigns nonnegative numbers of tokens to places of the net, $m_0:P\to\{0,1,\cdots\}$. Marked nets can be equivalently defined as $\mathcal{M}=(P,T,A,m_0)$.

A transition is enabled by a marking if all its input places are marked (*i.e.*, are assigned at least one token by the marking function). Each enabled transition $t$ can occur removing a single token from each of $t$'s input places and adding a single token to each of its output places. This creates a new marking function, a new set of enabled transitions and so on.

A place is shared if it has more than one input transition. A shared place $p$ is free-choice if the sets of output places of 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.

It is assumed that the choice of transition which occurs in each free-choice class and each class of conflicting transitions is random and can be described by a corresponding probability.

In timed nets [9], occurrence times are associated with transitions, and transition occurrences are timed events, *i.e.*, tokens are removed from input places at the beginning of the occurrence period, and they 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 transi-

tions, $c \to [0,1]$, and $t$ is a timing function which assigns an (average) occurrence time to each transition of the net, $f : T \to \mathbf{R}^+$, where $\mathbf{R}^+$ is the set of nonnegative real numbers.

If the occurrence times of transitions are deterministic, the nets are called D-timed nets [11], and their behavior is represented by an embedded Markov chain. If these occurrence times are stochastic with the (negative) exponential distribution function, the nets are called M-timed nets (Markovian nets) [12], and their behavior is represented by a Markov chain. In both cases, the concepts of state and state transitions have been formally defined and used in the derivation of the state space of the model. If this state space is finite and reasonably small, stationary probabilities of states can be determined by standard methods [13] and then the stationary probabilities are used for the derivation of many performance characteristics of the model [14]. In other cases, discrete event simulation [15] is used to find performance characteristics of a timed net.

In timed nets, some transitions may have occurrence times equal to zero, which means that the occurrences are instantaneous; all such transitions are called immediate (while the others are called timed). Since the occurrences of immediate transitions have no tangible effects on the (timed) behavior of the model, for each time instant all occurrences of the (enabled) immediate transitions are performed first, and then (still in the same time instant), when no more immediate transitions are enabled, the occurrences of (enabled) timed transitions are initiated. 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. A detailed description of the behavior or timed nets with immediate and timed transitions is given in [9].

## 3. Pipelined Processors

A timed Petri net model of a pipelined processor [16] at the level of instruction execution is shown in **Figure 1** (as usually, timed transitions are represented by solid bars, and immediate transitions by thin bars). 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 [17]. It is assumed that the first-level cache does not delay the processor, while level-1 cache misses introduce the delay of $t_c$ processor cycles for level-2 cache hits and $t_m$ processor cycles for level-2 cache misses.

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 (*Ts*0 with the choice probability $p_{s0}$), a single-cycle pipeline stall (modeled by the timed transition *Td*1 with the choice probability $p_{s1}$ associated with *Ts*1), and a two-cycle pipeline stall (modeled by *Td*2 and then *Td*1 with the choice probability $p_{s2}$ assigned to *Ts*2). Other pipeline stalls could be represented in a similar way, if needed.

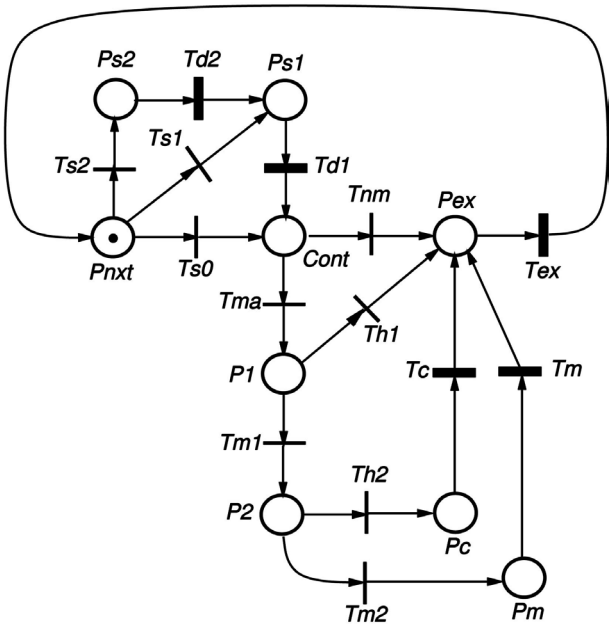


**Figure 1.** Instruction-level Petri net model of a pipelined processor.

*Cont* is a free-choice place which determines if the current instruction accesses memory or not; the probability associated with *Tma* is the probability that an instruction accesses memory (its typical value is 0.4 [3]). Complementary probability is associated with *Tnm*.

Marked place *Pex* 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 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 *Tex*.

*P*1 is another free-choice place which determines if the executing instruction results in a level-1 cache hit or miss. Transition *Th*1 (with the corresponding probability) models first-level cache hits when the processor continues fetching and issuing instructions without any additional delay. Level-1 cache misses are represented by *Tm*1.

*P*2 is another free-choice place; it models the hits and missed of the second-level cache. The probability associated with transition *Th*2 represents the hit ratio of the second-level cache (the occurrence time of *Tc* is the average access time to the second-level cache, $t_c$) while the miss ratio is associated with transition *Tm*2 which represents accesses to the main memory (with the occurrence time $t_m$).

Typical values of modeling parameters used in this paper 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 three combinations of values of the second-level cache

**Table 1.** Modeling parameters and their typical values.

| *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 |
| $p_m$ | prob. that an instruction accesses memory | 0.4 |
| $p_{s1}$ | prob. of one-cycle pipeline stall | 0.1 |
| $p_{s2}$ | prob. of two-cycle pipeline stall | 0.05 |

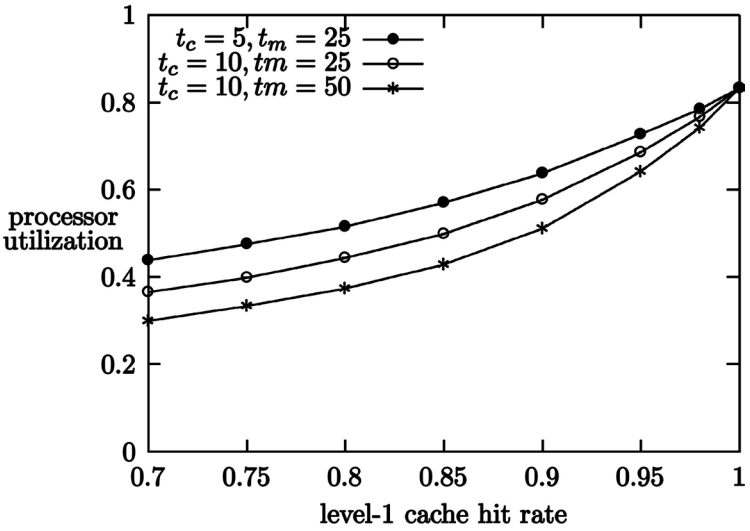


**Figure 2.** Processor utilization as a function of first-level cache hit rate for $h_2 = 0.8, p_s = 0.2$.

access time, $t_c = 5$ and $t_c = 10$, and main memory access time, $t_m = 25$ and $t_m = 50$. It should not be surprising that processor utilization is quite sensitive to the values of $h_1$, but is much less sensitive to the values of $t_c$ and $t_m$.

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

## 4. Shared-Memory Bus-Based Systems

An outline of a shared-memory bus-based multiprocessor is shown in **Figure 4**. 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 [18] which usually increases the miss ratio of accessing caches (and is otherwise not represented in the model).

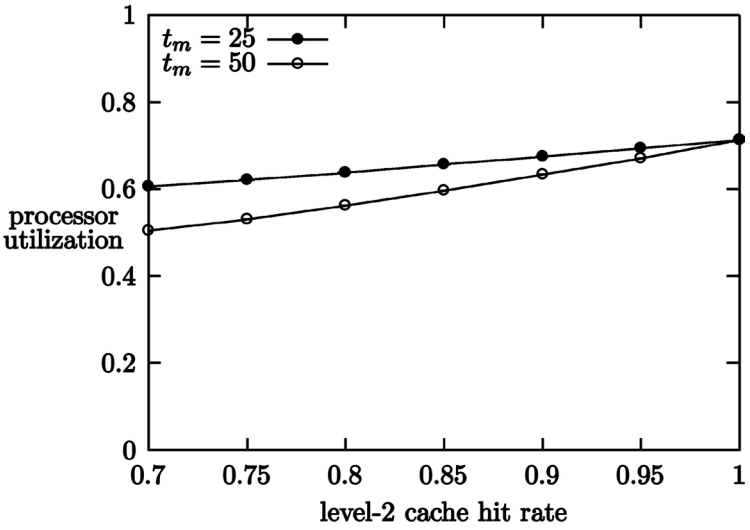


**Figure 3.** Processor utilization as a function of second-level cache hit rate for $h_1 = 0.9, p_s = 0.2$.

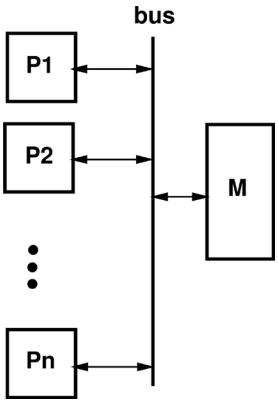


**Figure 4.** A shared-memory buss-based multiprocessor.

A timed Petri net model of a shared-memory bus-based multiprocessor is shown in Figure 5. It is very similar to the model shown in Figure 1 with the only new element that is the place *Bus* controlling access to the shared memory represented by *Tsm*. All *n* processors use the same model and are actually represented by the initial marking (in Figure 5, three processors are represented by the initial marking of place *Pnxt*). The single token assigned to *Bus* serializes accesses to the shared memory; if several processors simultaneously request access to the shared memory, only one access is granted while all other request wait in place *Psm*.

Figure 6 shows the utilization of processors and the bus as functions of the number of processors in a shared-memory system.

In Figure 6, the bus utilization approaches 100% for about 6 processors. Also, the reduction of processors' performance due to increasing waiting times for accessing the bus (and shared memory) is well illustrated in Figure 6.

The average waiting time (in processor cycles) of accessing shared memory (*i.e.*, the average time from requesting memory access to granting this access) is

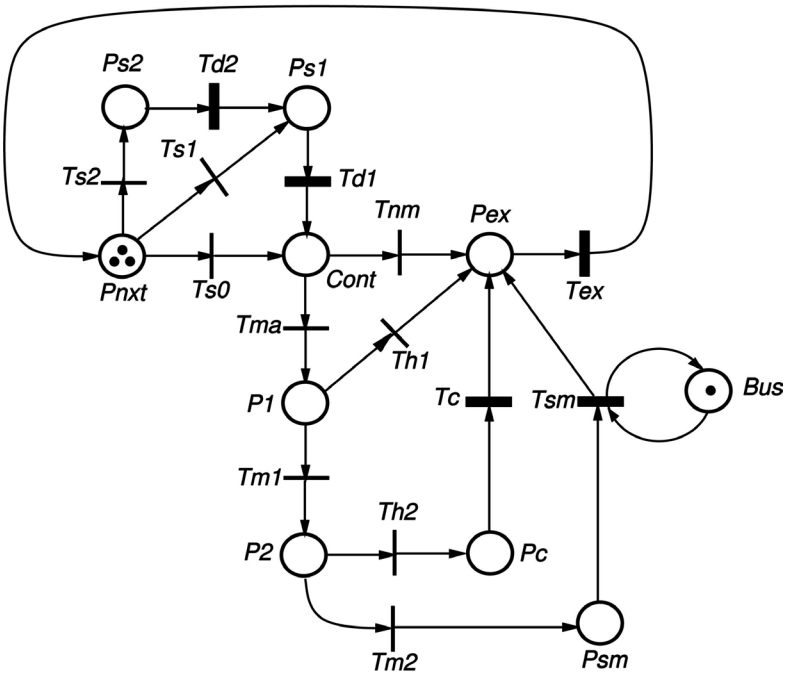


Figure 5. A timed Petri net model of bus-based shared-memory multiprocessor.

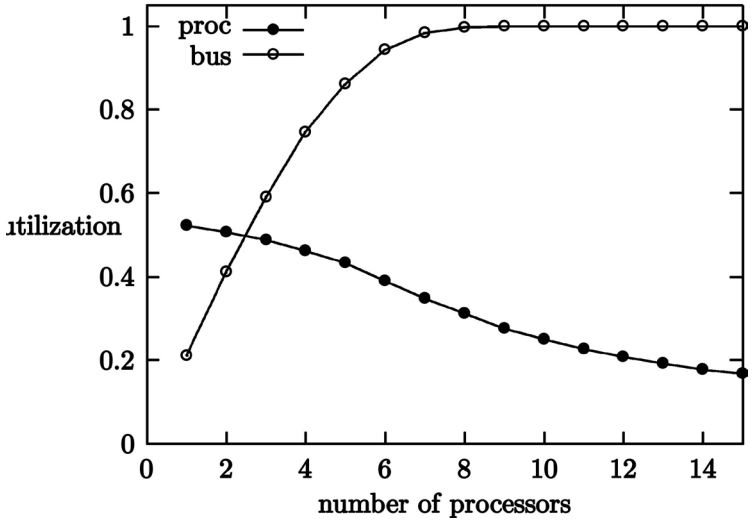


Figure 6. Processor and bus utilization as functions of the number of processors for $h_1 = 0.8$, $h_2 = 0.8$, $p_s = 0.2$. $t_c = 5$, $t_m = 25$.

shown in Figure 7 as a function of the number of processors in the system.

Figure 7 shows that the waiting times increase almost linearly with the number of processors when this number of greater then 6, *i.e.*, when the bus (and shared memory) is utilized in almost 100%. In such a situation each additional processor increases the average length of the queue of processors waiting for access to the bus by one, and then the waiting time of all processors is increased by the access time to shared memory, *i.e.*, by $t_m$.

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 (*i.e.*, it allows two concurrent accesses). Figure 8 outlines a dual bus shared-memory system.

Petri net model of a dual bus system is the same as in Figure 6, and the only difference is the initial marking of place *Bus*, which now requires two tokens to represent the two buses of the system.

Figure 9 shows the utilization of processors and buses as functions of the number of processors in a dual bus system. It should be observed that, for the small number of processors, the utilization of each bus in Figure 9 is one half of that in Figure 6, 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 6).

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

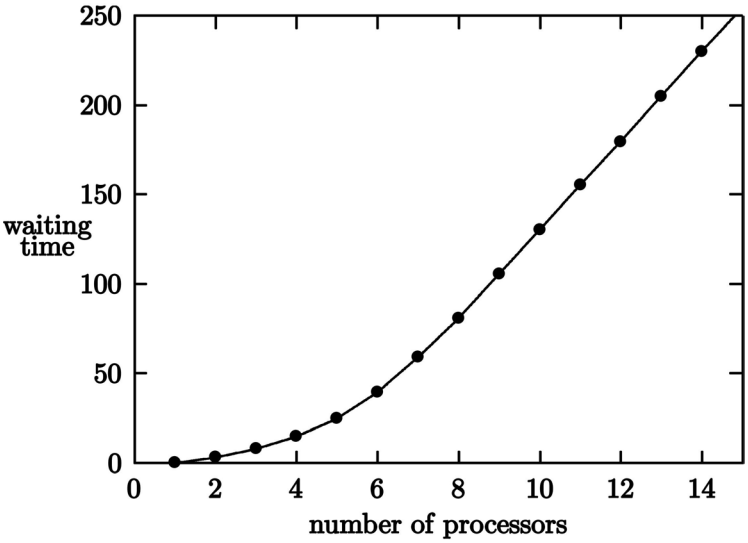


Figure 7. The average waiting time for accessing shared memory as as a function of the number of processors for $h_1 = 0.8$, $h_2 = 0.8$, $p_s = 0.2$. $t_c = 5$, $t_m = 25$.

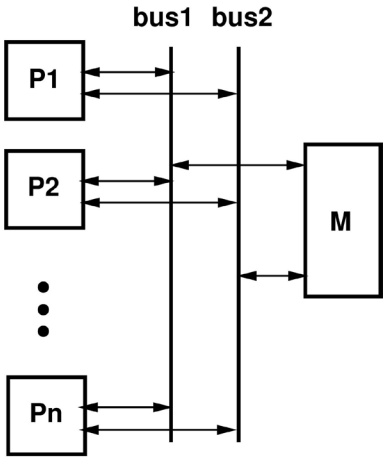


Figure 8. A dual bus shared-memory multiprocessor.

provided that the bus is also split into sections associated with each module, with processors accessing all such sections, as shown in Figure 10 for 4 independent memory modules. The main difference between a multibus system (Figure 8) 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 10) each section of the bus accesses only one memory module. In the system shown in Figure 10, 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 models of a system outlined in Figure 10 is shown in Figure 11.

In Figure 11, the free-choice place *Psm* selects the requested memory module by transitions *Tbj*, $j = 1, 2, 3, 4$, and forwards the memory access request to the

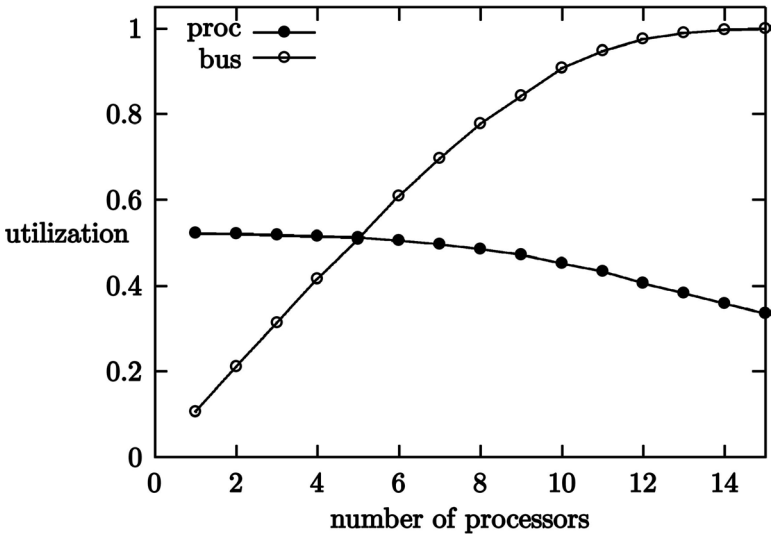


Figure 9. Processor and bus utilization as functions of the number of processors—dual bus system with $h_1 = 0.8$, $h_2 = 0.8$, $p_s = 0.2$. $t_c = 5$, $t_m = 25$.

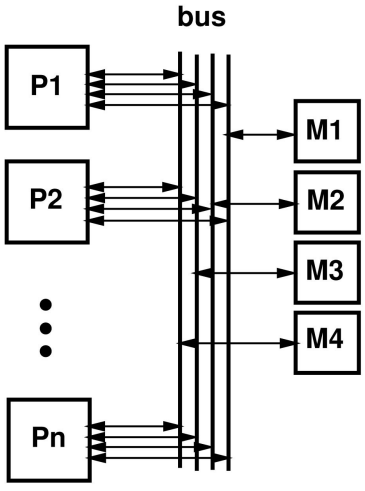


Figure 10. A shared-memory multiprocessor with multiple memory modules.

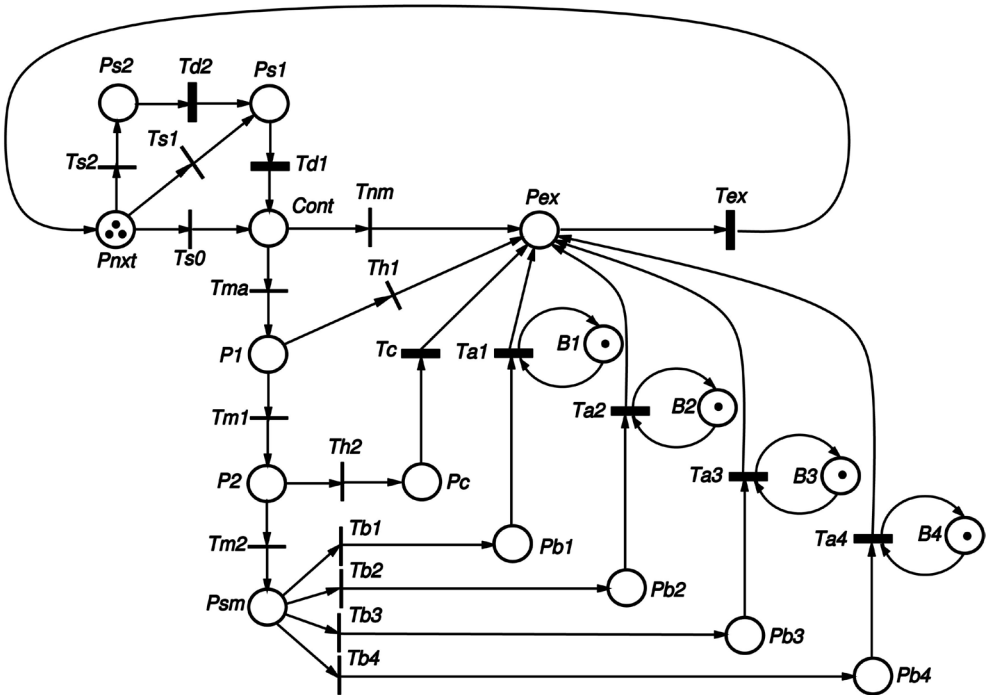


**Figure 11.** Petri net model of a shared-memory multiprocessor with multiple memory modules.

selected memory module (place *Pbj*). If the selected module is available, *i.e.* if place *Bj* is marked, the access to shared memory is initiated. If memory module is not available when it is requested, the memory access is delayed (in *Pbj*) until the requested module becomes available.

If more that one processor is 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 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 11, 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, 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 12 shows the utilization of processors and buses as functions of the number of processors in a system outlined in Figure 10.

In Figure 12, even for 25 processors the average utilization of the bus is close to 85%, so the system can accommodate a few additional processors.

## 5. Concluding Remarks

The paper 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

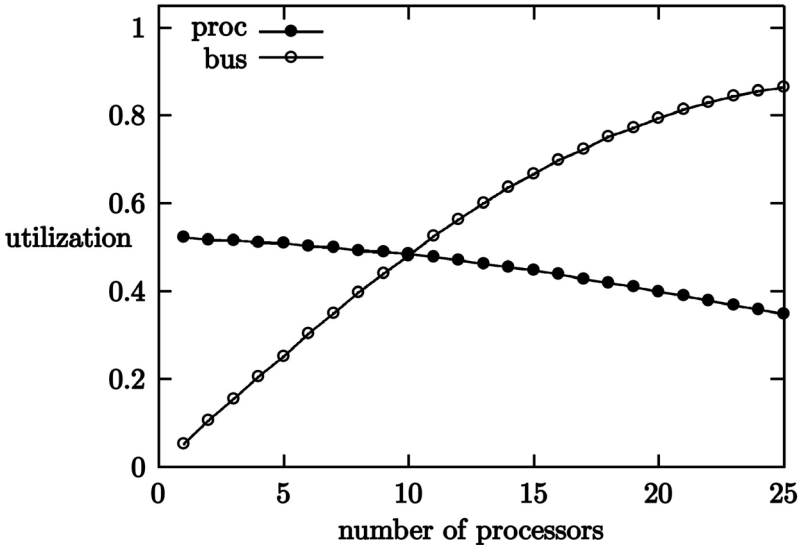


Figure 12. Processor and bus utilization as functions of the number of processors—system with four memory modules and $h_1 = 0.8$, $h_2 = 0.8$, $p_s = 0.2$. $t_c = 5$, $t_m = 25$.

the performance of the system. The models are rather simple with straightforward representation of modeling parameters.

Performance results show that bus-based share-memory systems cannot be used efficiently for large numbers of processors. When the utilization of the bus approaches 100%, the utilization of individual processors as well as the entire system degrades with the processors spending an increasing amount of time waiting for the access to the bus (and shared memory).

The long-latency accesses to the shared memory can be tolerated by using instruction-level multithreading [19], which may result in increased performance of processors. It should be observed, however, that multithreading increases concurrency at the thread level, so improved processor utilization is associated with increases demand for accessing the bus; the utilization of processors cannot be improved without increasing the utilization of the bus. On the other hand, the utilization of the bus can be reduced by improved performance of the cache memory [20].

Performance results presented in this paper have been obtained by the simulation of developed Petri net models. Their accuracy can be verified by a comparison with analytical solution for models with reasonably small state spaces. For example, the model shown in **Figure 1** has only 12 states, so its analytical solution (for different values of modeling parameters) can be easily obtained. **Table 2** shows such a comparison of processor utilization for several values of parameters $h_1$ and $h_2$. In all cases the simulation-based results are very close to the analytical ones.

The results presented in this paper are consistent with the earlier results, presented in [10] (although some parameters need rescaling for comparison because of different modeling of instruction execution). **Table 3** compares the utilization of processors and buses in a system with the split bus, for a selected

**Table 2.** A comparison of simulation and analytical results.

| $h_1$ | $h_2$ | simulated results | analytical results |
|---|---|---|---|
| 0.8 | 0.8 | 0.52178 | 0.52083 |
| 0.8 | 0.9 | 0.57000 | 0.56818 |
| 0.9 | 0.8 | 0.64166 | 0.64103 |
| 0.9 | 0.9 | 0.67682 | 0.67568 |

**Table 3.** A comparison of utilization of processors and buses.

| number of processors | current model | | previous model | |
|---|---|---|---|---|
| | processor | bus | processor | bus |
| 5 | 0.460 | 0.288 | 0.462 | 0.284 |
| 10 | 0.430 | 0.536 | 0.434 | 0.530 |
| 15 | 0.384 | 0.717 | 0.390 | 0.716 |
| 20 | 0.334 | 0.830 | 0.334 | 0.833 |

number of processors in the system.

Although the models of multiprocessors are very different, the performance results are practically the same.

Finally, it should be noted that performance characteristics presented in this paper can only be used as some insight into the complex behavior of multiprocessor systems. The performance of real-life multiprocessors 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. Therefore the performance of multiprocessors is typically described at a higher level of abstraction, for example, in terms of the number of processors.

## Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

## References

[1] Hamilton, S. (1999) Taking Moore's Law into the Next Century. *IEEE Computer*, **32**, 43-48. https://doi.org/10.1109/2.738303

[2] Lundstrom, M. (2003) Moore's Law Forever? *Science*, **299**, 210-211. https://doi.org/10.1126/science.1079567

[3] Patterson, D.A. and Hennessy, J.L. (2006) Computer Architecture—A Quantitative Approach. 4th Edition, Morgan Kaufmann, San Mateo, CA.

[4] Wilkes, M.V. (2001) The Memory Gap and the Future of High-Performance Memories. *ACM Architecture News*, **29**, 2-7. https://doi.org/10.1145/373574.373576

[5] Mutlu, O., Stark, J., Wilkerson, C. and Patt, Y.N. (2003) Runahead Execution: An Effective Alternative to Large Instruction Windows. *IEEE Micro*, **23**, 20-25. https://doi.org/10.1109/MM.2003.1261383

[6] Murata, T. (1989) Petri Nets: Properties, Analysis and Applications. *Proceedings of IEEE*, **77**, 541-580. https://doi.org/10.1109/5.24143

[7] Reisig, W. (1985) Petri Nets—An Introduction (EATCS Monographs on Theoretical Computer Science 4). Springer-Verlag, New York.

[8] Popova-Zeugmann, L. (2013) Time and Petri Nets. Springer-Verlag, Berlin. https://doi.org/10.1007/978-3-642-41115-1

[9] Zuberek, W.M. (1991) Timed Petri Nets—Definitions, Properties and Applications. *Microelectronics and Reliability* (*Special Issue on Petri Nets and Related Graph Models*), **31**, 627-644. https://doi.org/10.1016/0026-2714(91)90007-T

[10] Zuberek, W. (2018) Performance Analysis of Shared-Memory Bus-Based Multiprocessors Using Timed Petri Nets. In: Campos-Rodriguez, R., Ed., *Petri Nets in Science and Engineering*, InTechOpen, London, Chapter 5, 75-91.

[11] Zuberek, W.M. (1987) D-Timed Petri Nets and Modelling of Timeouts and Protocols. *Transactions of the Society for Computer Simulation*, **4**, 331-357.

[12] Zuberek, W.M. (1986) M-Timed Petri Nets, Priorities, Preemptions, and Performance Evaluation of Systems. In: *Advances in Petri Nets* 1985 (*LNCS* 222), Springer-Verlag, Berlin, 478-498. https://doi.org/10.1007/BFb0016227

[13] Allen, A.A. (1991) Probability, Statistics and Queueing Theory with Computer Science Applications. 2rd Edition, Academic Press, San Diego, CA.

[14] Jain, R. (1991) The Art of Computer Systems Performance Analysis. J. Wiley Interscience, New York, NY.

[15] Pooch, U.W. and Wall, J.A. (1993) Discrete Event Simulation. CRC Press, Boca Raton, FL.

[16] Ramamoorthy, C.V. and Li, H.F. (1977) Pipeline Architecture. *ACM Computing Surveys*, **9**, 61-102. https://doi.org/10.1145/356683.356687

[17] Zuberek, W.M. (2007) Modeling and Analysis of Simultaneous Multithreading. 14*th International Conference on Analytical and Stochastic Modeling Techniques and Applications*, *a Part of the* 21*st European Conference on Modeling and Simulation*, Prague, 15-120.

[18] Suh, T., Lee, H.S. and Blough, D.M. (2004) Integrating Cache Coherence Protocols for Heterogeneous Multiprocessor System, Part 2. *IEEE Micro*, **24**, 55-69. https://doi.org/10.1109/MM.2004.50

[19] Jesshope, C. (2003) Multithreaded Microprocessors—Evolution or Revolution. In: *Advances in Computer Systems Architecture*, Springer-Verlag, Berlin Heidelberg, 21-45. https://doi.org/10.1007/978-3-540-39864-6_4

[20] Milenkovic, A. (2000) Achieving High Performance in Bus-Based Shared-Memory Multiprocessors. *IEEE Concurrency*, **8**, 36-44. https://doi.org/10.1109/4434.865891