A Petri Net-Based Approach to Modeling Concurrency and Parallelism in Computer Architectures

Ilija Hristoski[1] and Jelena Stojanov[2]

[1] *"St. Kliment Ohridski" University – Bitola / Faculty of Economics – Prilep, Prilepski Braniteli St. 143, 7500 Prilep, North Macedonia*
[2] *University of Novi Sad / Technical Faculty "Mihajlo Pupin" – Zrenjanin, Đure Đakovića St., 23101 Zrenjanin, Serbia*

ilija.hristoski@uklo.edu.mk; jelena.stojanov@tfzr.uns.ac.rs

Abstract:
The paper focuses on concurrency and parallelism, two closely related concepts in computer architecture that deal with how tasks are executed simultaneously, though that they differ in their underlying mechanisms and objectives. Utilizing the class of Generalized Stochastic Petri Nets (GSPNs), this study proposes four distinct Petri Net-based models to capture the behavior and logic of all possible combinations of these two concepts with a single aim to clarify and explain them. Despite their straightforwardness, the resulting generic models can serve as blueprints that can be applied to simulate, design, optimize, and verify systems that rely on concurrency and parallelism, thereby enhancing both theoretical understanding and practical implementations of such systems.

Keywords:
Concurrency, parallelism, Generalized Stochastic Petri Nets (GSPNs), modeling, TimeNET

1. Introduction

The concepts of concurrency and parallelism have long histories in the development of computer science and computer architecture, since both concepts begun developing in the 1960s [1]. Concurrency was driven by the need for time-sharing and multiprogramming systems [2], while parallelism was advanced by the development of multi-threaded and multi-core processors [3]. Both paradigms have led to the development of distributed computing, supercomputers and high-performance computing systems.

In computer science, concurrency and parallelism refer to entirely distinct concepts, even though they are often used interchangeably. Rob Pike, one of the inventors of the Go programming language, in one of his excellent talks said: "*Concurrency is about DEALING with lots of things at once. Parallelism is about DOING lots of things at once.*" According to him, this distinction emphasizes that concurrency is more about the design or structure of a program, whilst parallelism is about the execution of programs, having minded that concurrency enables parallelism and makes it easy [4] [5]. Parallelism involves the use of multiple CPU cores, with each core executing a task independently. Conversely, concurrency allows a program to manage multiple tasks even on a single CPU core by switching between tasks (or threads) without necessarily completing each one before moving to the next. A program or system can exhibit characteristics of parallelism, concurrency, neither, or a combination of both [6].

A system is considered concurrent if it can handle multiple actions simultaneously or in progress at the same time [7], allowing computation to proceed without waiting for others to complete [8]. Concurrency enables a program to manage multiple tasks on a single CPU core by interleaving execution, switching between tasks without finishing each one before moving on. It involves structuring a program so tasks can progress without necessarily being executed at the same time, utilizing techniques like time-slicing. Concurrency is about handling many tasks at once and allows for components of a program or problem to be executed in any order, enhancing speed, particularly in multi-core systems [9]. Even on a single-core CPU, concurrency is achieved through rapid task

switching. It refers to the structure and scheduling of tasks, ensuring they can progress logically at the same time, regardless of actual simultaneous execution [10].

Parallelism refers to the simultaneous execution of multiple processes or threads, improving computational speed by utilizing multiple processors or CPU cores to work on different parts of a task [11]. It requires multicore or multithreaded CPUs to execute tasks or subtasks at the same time [12], and can occur at various levels, such as instruction-level and data-level parallelism [13]. A parallel system supports the execution of multiple actions simultaneously, with each core performing a task independently [7]. The essence of parallelism is about doing tasks simultaneously, requiring multiple CPU cores or processors.

Despite their long history, concurrency and parallelism remain central to modern research and practice in computer science, particularly in computer architecture, due to ongoing challenges and evolving demands. These include the rise of multi-core processors, high-performance computing, scalability, energy efficiency, real-time systems, embedded and distributed system design, cloud computing, and new programming models for AI, machine learning, and deep learning systems.

By revisiting those two paradigms, the aim of the paper is to clarify and emphasize their differences by proposing four generic, yet quite simple simulation models based on the utilization of the class of Generalized Stochastic Petri Nets (GSPNs). However, it is worth noting that beyond Petri Nets and their various extensions (e.g., Colored Petri Nets for modeling data-dependent concurrency, Timed Petri Nets for modeling real-time concurrent systems), various other formalisms, tools, and approaches have been developed to model concurrency and concurrent computing. These alternatives differ in their treatment of concurrency, synchronization, and inter-process communication. Among others, some of the most prominent alternatives include Process Calculi (Calculus of communicating systems, Communicating Sequential Processes – CSPs, π-calculus), the Actor Model, State Machines and Automata (Finite State Machines, Timed Automata), Programming Models (Thread-based Models, Message-Passing Models, and Dataflow Models), the Parallel Random-Access Machine (PRAM), Bulk Synchronous Parallel (BSP) model, as well as various software tools and programming languages, such as Simple Concurrent Object-Oriented Programming (SCOOP) and the programming languages Go (Golang) and Haskell. On the other hand, there are also alternative approaches that are used to model parallelism and parallel computing, which focus on representing, analyzing, and optimizing parallel tasks, communication, and execution in multi-core and distributed environments, such as Parallel Programming Models (Message Passing Interface – MPI, OpenMP, CUDA), Dataflow Models (Kahn Process Networks, Stream Processing), Graph-Based Models (Directed Acyclic Graphs – DAGs, Task Graphs in OpenCL), Formal Parallel Models (Parallel Random-Access Machine – PRAM, Bulk Synchronous Parallel (BSP) Model), and High-Level Parallel Languages and Libraries (Chapel, Cilk, Threading Building Blocks – TBBs).

The structure of the paper is as follows: Section 2 offers a concise overview of the class of Generalized Stochastic Petri Nets (GSPNs). In Section 3, divided into four subsections, we detail the proposed GSPN-based models that integrate both concurrency and parallelism. Section 4 presents a discussion of the proposed solutions. Finally, the concluding section summarizes the key findings and provides recommendations for future research.

2. Generalized Stochastic Petri Nets

Petri Nets are a well-known mathematical and graphical modeling tool widely used to represent and analyze concurrent, distributed, and dynamic systems. Generalized Stochastic Petri Nets (GSPNs) are an advanced class of Petri Nets, representing an extension of the class of Stochastic Petri Nets (SPNs), designed to model systems that exhibit both deterministic and stochastic behaviors. Initially introduced by Ajmone Marsan in 1984, GSPNs significantly extend the capabilities of standard (i.e., non-timed) Petri Nets by incorporating timing information, thus enabling the analysis of performance, reliability, and availability of concurrent and distributed systems [14] [15] [16] [17].

Without any intention to elaborate on the building blocks, their graphical representation, and operational behavior behind GSPNs in a more detailed manner, we hereby simply present their formal definition.

Formally, a GSPN is defined as a tuple $G = (P, T, I, O, M_0, \lambda)$, where:

- P is a finite set of places;
- $T = T_{imm} \cup T_{timed}$ is a set of transitions, with T_{imm} denoting immediate transitions and T_{timed} denoting timed transitions;
- I and O represent the input and output functions, mapping places to transitions and vice versa;
- M_0 is the initial marking of the Petri Net, representing the starting state of the system;
- λ: $T_{timed} \rightarrow \mathbb{R}^+$ maps each timed transition to its associated firing rate.

It is also worth noting that GSPNs can incorporate inhibitor arcs, which greatly enhance their basic functionality, despite not being included in the standard definition of GSPNs.

The stochastic nature of GSPNs allows for performance analysis through the generation of continuous-time Markov chains (CTMCs). Each reachable marking (state) of the GSPN corresponds to a state in the CTMC, and the transitions between states are governed by the firing rates of the timed transitions.

GSPNs are widely used in the modeling of complex systems, such as computer systems, computer components, communication networks, manufacturing systems, various operations, protocols, strategies, etc., where random events, probabilistic choices, and time delays are inherent.

3. GSPN-based Models of Concurrency and Parallelism

The following four subsections present and elaborate on the four possible combinations of concurrency and parallelism found in computer architectures. It is crucial to distinguish between concurrency and parallelism when determining the most effective approach for solving large-scale problems, even though these terms are often used interchangeably in practice. By focusing on how CPU cores manage tasks at the hardware level, the GSPN-based solutions provided here play a vital role in clarifying these two closely related yet fundamentally distinct paradigms.

For simplicity reasons, it is assumed that there are at most two tasks per CPU core and there are at most two CPU cores in the system, which is an absolute minimum. Moreover, in the first two cases, both dealing with the concept of non-parallelism, there are just two tasks in the system, *Task_A* and *Task_B*. In the rest of the cases that deal with the parallelism of tasks, *Task_C* and *Task_D* are added and also taken into account. It is also supposed that each of these tasks is composed of arbitrary number of subtasks (M, N, P, and R), respectively, that can be processed independently by CPU cores. As usual, the arrival rate of subtasks in a CPU core is labeled with the Greek letter λ, whilst the service rate of CPU cores is labeled with the Greek letter μ. Both the arrival times of tasks in CPU cores and CPU cores' service times are supposed to follow exponential distribution.

All hereby presented GSPN-based models are built and verified using TimeNET 4.5, a dedicated software tool suitable for modeling, analysis, and performance evaluation of stochastic systems using extended Petri Nets, including the class of Generalized Stochastic Petri Nets (GSPNs). TimeNET provides a powerful development environment suitable for the simulation and formal analysis of systems that exhibit concurrency, synchronization, and random behavior, making it particularly useful for evaluating systems where timing and probabilistic events play a key role [18] [19] [20] [21].

3.1. Case #1: Non-concurrent, non-parallel tasks

In this case, the system processes all tasks one at a time, sequentially (i.e., one after the other), by the means of a single CPU core (Figure 1).

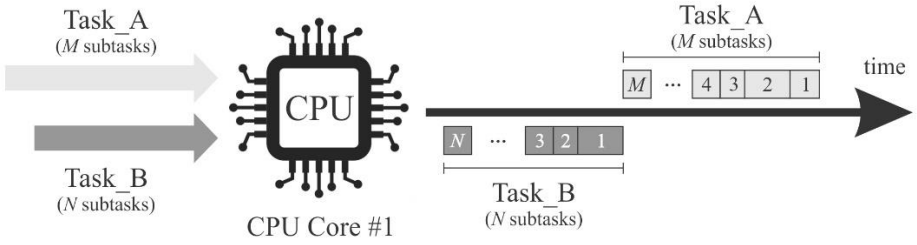


Figure 1: Non-concurrent and non-parallel execution of tasks (Source: The authors)

A single CPU core executes each task sequentially, so that *Task_A* finishes before *Task_B* begins. Task_A consists of *M* subtasks, i.e. tokens held in place *P_task_A*, while *Task_B* is comprised of *N* subtasks, i.e. tokens held in place *P_task_B*, as portrayed in Figure 2. The result of the execution of the GSPN model depicted in Figure 2 is equivalent to what is shown in Figure 1.

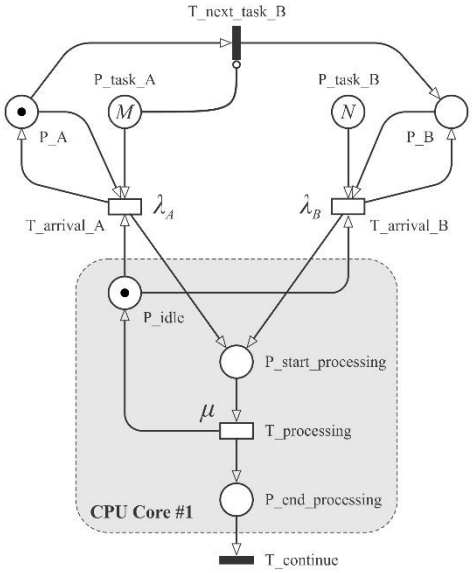
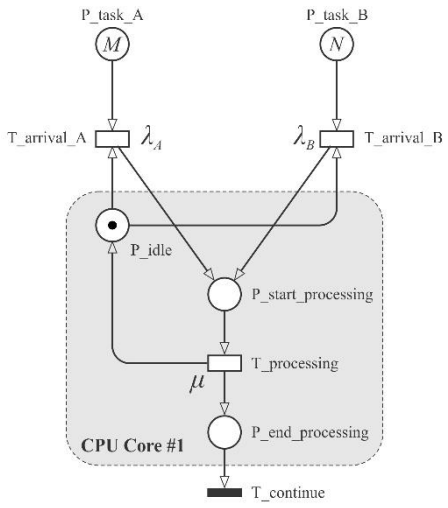


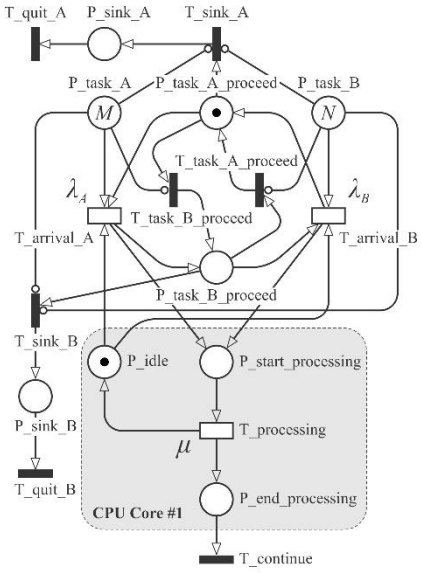
Figure 2: GSPN model resembling Case #1 (Source: The authors)

3.2. Case #2: Concurrent, non-parallel tasks

This scenario occurs when the system handles multiple tasks simultaneously, but no two tasks are executed at the exact same time. A single CPU core processes *Task_A* and *Task_B* concurrently. In the GSPN model presented in Figure 3a, the CPU core can process multiple subtasks from either *Task_A* or *Task_B*, one after the other. Meanwhile, the model in Figure 3b enforces alternating processing, where the CPU must complete the current subtask from *Task_A* before proceeding with the next subtask from *Task_B* and vice-versa, as shown in Figure 4a and Figure 4b, respectively.

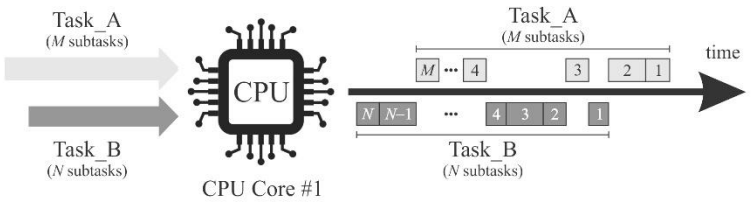


(a) Non-alternating subtasks

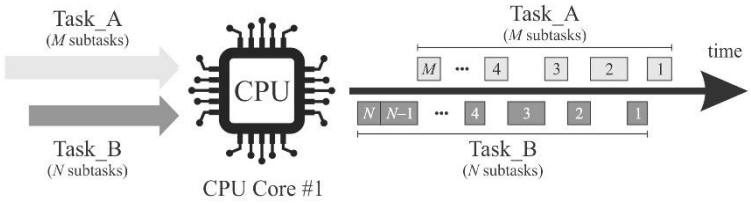


(b) Alternating subtasks

Figure 3: GSPN model resembling Case #2 (Source: The authors)



(a) Non-alternating subtasks



(b) Alternating subtasks

Figure 4: Concurrent and non-parallel execution of tasks (Source: The authors)

3.3. Case #3: Non-concurrent, parallel tasks

In this case, the system processes multiple subtasks of a task in multi-core CPU at the same time. In this case, two CPU cores execute each task in parallel, as shown in Figure 5. The result of the execution of the GSPN model depicted in Figure 5 is equivalent to what is shown in Figure 6.

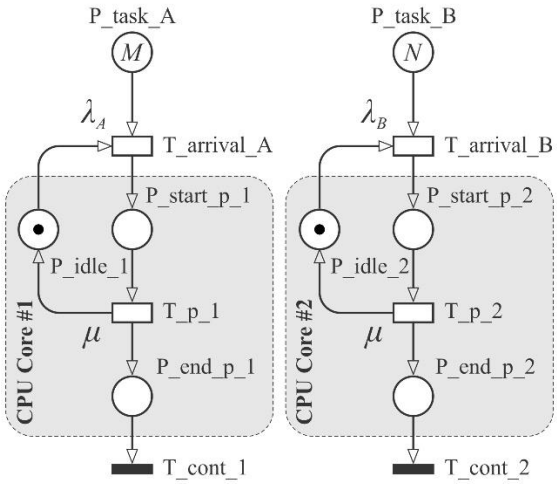


Figure 5: GSPN model resembling Case #3 (Source: The authors)

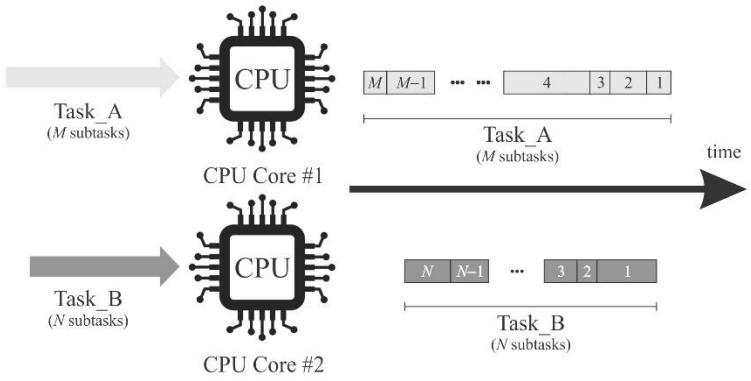


Figure 6: Non-concurrent and parallel execution of tasks (Source: The authors)

3.4. Case #4: Concurrent, parallel tasks

This is the case when the system processes multiple tasks concurrently in a multi-core CPU at the same time. In this particular case, there are two CPU cores; each of them executes two tasks concurrently; *Task_A* and *Task_B* are processed concurrently by CPU Core #1, whilst *Task_C* and *Task_D* are processed concurrently by CPU Core #2. At the same time, both *Task_A* and *Task_B* are processed in parallel with *Task_C* and *Task_D* (Figure 7).

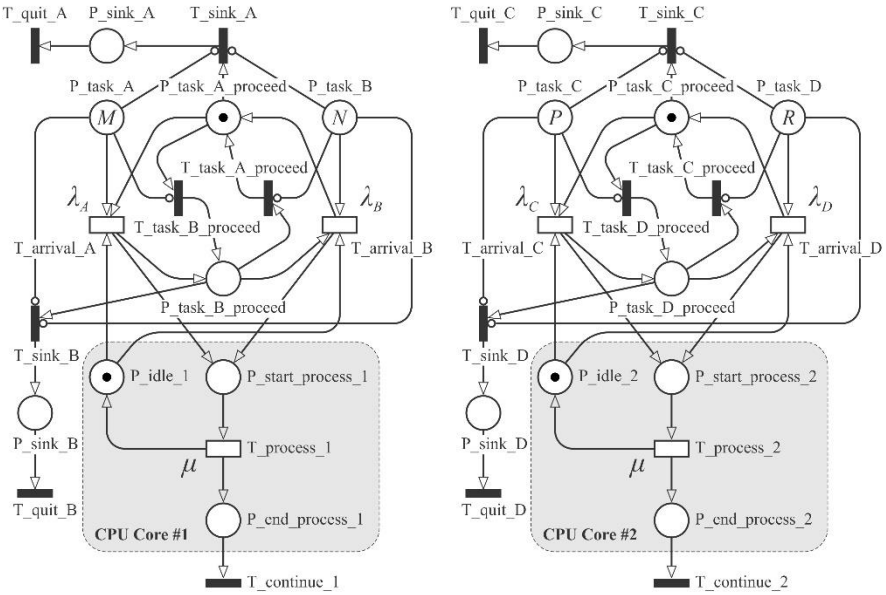


Figure 7: GSPN model resembling Case #4 (Source: The authors)

The result of the execution of the GSPN model depicted in Figure 7 is equivalent to what is shown in Figure 8.

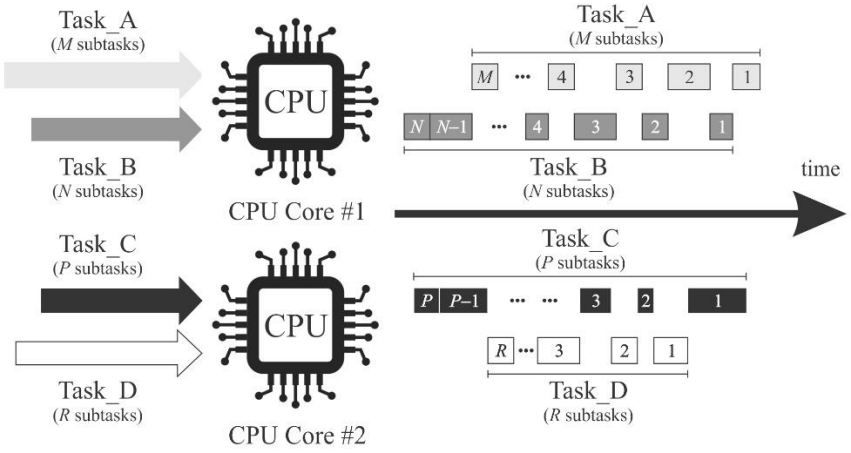


Figure 8: Concurrent and parallel execution of tasks (Source: The authors)

4. Discussion

Despite the fact that GSPNs are a powerful tool for modeling and analyzing systems involving concurrency and parallelism, they face several *limitations* when applied to real-world systems. The most significant limitation is the state space explosion problem, which occurs when the complexity of the modeled system increases by including more concurrent processes (i.e., tasks), larger number of tokens, or transitions. Further on, a serious limitation of this approach can be addressing complexity

intrinsic to modeling real-world systems where the processing of tasks involve non-Markovian time distributions, such as deterministic, fixed delays or other types of time distributions that are not exponential. Yet another important limitation is the difficulty of modeling complex synchronization mechanisms frequently found in many computer architectures that control the access of multiple tasks (i.e., threads or processes) to shared resources, which might require complex and potentially unwieldy Petri Net structures, making the model harder to interpret and analyze, even computationally intractable. At last but not at least, there is always the limitation known as the trade-off between abstraction level and scalability: GSPNs provide a high level of abstraction, which is advantageous for capturing general behaviors of systems, but this feature can become a limitation when low-level details are important. The need to model such intricate details can make GSPN models either too abstract, which can lead to losing important information, or too complex by reducing the simplicity and elegance of the Petri Net approach, thus making the analysis difficult.

The two important processes used to ensure that the model accurately represents the system being studied and behaves as expected are the processes of model verification and validation.

The *verification* of the proposed GSPN models, which refers to the process of ensuring that the GSPN model is correctly implemented according to its formal specification, has been carried out using the TimeNET's Token Game module. It proved that the proposed models are built correctly in terms of their structure, behavior, and conformance to the GSPN rules, by addressing key aspects such as structural correctness, deadlocks and liveness, boundedness, reachability analysis, and internal consistency.

On the other hand, the *validation* of these models, which refers to the process of ensuring that the GSPN model accurately represents the real-world system or process it is intended to simulate, has not been carried out yet due to the lack of real-world data, physical resources (CPUs), as well as suitable tools and measurement methods to convey effective validation. However, in this particular case, this aspect may be considered less crucial, since the proposed models are intended to be used mostly as conceptual designs, having minded their exploratory and theoretical focus.

5. Conclusions

This paper has presented a Petri Net-based approach to modeling concurrency and parallelism in computer architectures, with a focus on the use of Generalized Stochastic Petri Nets (GSPNs). The proposed formal GSPN framework, despite the ultimate straightforwardness of the presented models, offers a flexible and powerful tool for modeling key behaviors in concurrent and parallel execution of tasks in computer systems, with a huge potential in providing insights into performance, resource utilization, and potential bottlenecks. However, despite their modeling strengths, GSPNs come with several limitations that affect their applicability in large-scale, real-world systems, as it was elaborated in the previous section. Despite these limitations, GSPNs remain a valuable tool for quantitative analysis of systems exhibiting features of concurrency and parallelism, and can be applied to obtain metrics such as throughput, utilization, and response time.

The proposed GSPN-based models presented in this paper can be used for simulating and analyzing system performance in computer architecture and related fields, thus helping to assess how different task-handling strategies can affect resource utilization and responsiveness under various conditions, including the number of subtasks, their arrival rates, and CPU processing speeds. They can also help in designing and optimizing hardware or software systems by exploring efficient implementations of concurrency and parallelism, optimizing factors like execution speed and energy consumption. These models can also be used for verification of system behavior, and detecting issues such as deadlocks or bottlenecks. Furthermore, they can allow for the comparison of concurrency and parallelism strategies, guiding decision-makers in choosing the best approach. Ultimately, they can provide a theoretical foundation for studying and teaching these concepts and assist in resource allocation and task scheduling in multi-core or distributed systems for greater efficiency.

Future work could focus on the performance evaluation of the proposed GSPN-based models and their validation vis-à-vis real-world systems. This will ensure that GSPNs remain a practical and effective tool in the ongoing efforts to improve the design and analysis of concurrent and parallel computing systems.

References:

[1] L. Proença, A brief history of modern computers, multitasking and operating systems, 2022. URL: https://dev.to/leandronsp/a-brief-history-of-modern-computers-multitasking-and-operating-systems-2cbn

[2] L. Lamport, The Computer Science of Concurrency: The Early Years, In: Concurrency: The Works of Leslie Lamport (2019) 13–26. doi: 10.1145/3335772.3335775

[3] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, 2nd. ed., Addison-Wesley, Boston, MA, USA, 2003.

[4] R. Pike, Concurrency is Not Parallelism, Speech at Heroku Waza 2012, Video, 2012. URL: https://www.youtube.com/watch?v=oV9rvDllKEg

[5] M. Streeter, Concurrency is Not Parallelism, Rob Spike's Speech at Heroku Waza 2012, Video, 2020. URL: https://www.youtube.com/watch?v=qmg1CF3gZQ0

[6] S. Marlow, Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming, 1st. ed., O'Reilly Media, Sebastopol, CA, USA, 2013.

[7] C. Breshears, The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications, 1st. ed., O'Reilly Media, Sebastopol, CA, USA, 2009.

[8] A. Silberschatz, P. B. Galvin, G. Gagne, Operating System Concepts, 9th. ed., Wiley, Hoboken, NJ, USA, 2012.

[9] L. Lamport, Time, Clocks and the Ordering of Events in a Distributed System, Communications of the ACM 21(7) (1978), 558–565. doi: 10.1145/359545.359563

[10] P. B. Hansen (ed.), The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls, Springer-Verlag, New York, NY, USA, 2002.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 3rd. ed., MIT Press, Cambridge, MA, USA, 2009.

[12] A. C. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, B. Esbaugh, Parallelism via Multithreaded and Multicore CPUs, Computer 43(3) (2010), 24–32. doi: 10.1109/MC.2010.75

[13] D. A. Patterson, J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, 5th. ed., Morgan Kaufmann, Burlington, MA, USA, 2013.

[14] M. Ajmone Marsan, G. Conte, G. Balbo, A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems, ACM Transactions on Computer Systems 2(2) (1984), 93–122. doi: 10.1145/190.191

[15] G. Chiola, M. Ajmone Marsan, G. Balbo, G. Conte, Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications, IEEE Transactions on Software Engineering 19(2) (1993), 89–107. doi: 10.1109/32.214828

[16] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modelling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing, John Wiley and Sons, West Sussex, UK, 1995.

[17] G. Balbo, Introduction to Generalized Stochastic Petri Nets, in: M. Bernardo, J. Hillston (Eds.), Formal Methods for Performance Evaluation (SFM 2007), Volume 4486 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2007, pp. 83–131. doi: 10.1007/978-3-540-72522-0_3

[18] A. Zimmermann, J. Freiheit, R. German, G. Hommel, Petri Net Modelling and Performability Evaluation with TimeNET 3.0, in: B. R. Haverkort, H. C. Bohnenkamp, C. U. Smith (Eds.) Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS 2000), Volume 1786 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2000, pp. 188–202.

[19] A. Zimmermann, M. Knoke, A. Huck, G. Hommel, Towards Version 4.0 of TimeNET, in: Proceedings of the 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2006), Nurnberg, Germany, pp. 477–480, 2006.

[20] A. Zimmermann, M. Knoke, TimeNET 4.0: A Software Tool for the Performability Evaluation with Stochastic and Colored Petri Nets – User Manual, Faculty of EE&CS Technical Report 2007-13, Technische Universität Berlin, Berlin, Germany, 2007.

[21] A. Zimmermann, Modelling and Performance Evaluation with TimeNET 4.4, in: N. Bertrand, L. Bortolussi, (Eds.), Quantitative Evaluation of Systems (QEST 2017), Volume 10503 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2017, pp. 300–303.