View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by IntechOpen

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

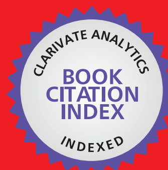
122,000
International authors and editors

135M
Downloads

154
Countries delivered to

Our authors are among the
TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 10

Fluid Stochastic Petri Nets: From Fluid Atoms in ILP Processor Pipelines to Fluid Atoms in P2P Streaming Networks

Pece Mitrevski and Zoran Kotevski

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/50615

1. Introduction

Fluid models have been used and investigated in queuing theory [1]. Recently, the concept of fluid models was used in the context of Stochastic Petri Nets, referred to as *Fluid Stochastic Petri Nets* (FSPNs) [2-6]. In FSPNs, the fluid variables are represented by fluid places, which can hold fluid rather than discrete tokens. Transition firings are determined by both discrete and fluid places, and fluid flow is permitted through the enabled timed transitions in the Petri Net. By associating exponentially distributed or zero firing time with transitions, the differential equations for the underlying stochastic process can be derived. The dynamics of an FSPN are described by a system of first-order hyperbolic *partial differential equations* (PDEs) combined with initial and boundary equations. The general system of PDEs may be solved by a standard discretization approach. In [6], the problem of immediate transitions has also been addressed in relation to the fluid levels, by allowing fluid places to be connected to immediate transitions. The transportation of fluid in zero time is described by appropriately chosen boundary conditions.

In a typical multiple-issue processor, instructions flow through pipeline and pass through separate pipeline stages connected by buffers. An open multi-chain queuing network can present this organization, with each stage being a service center with a limited buffer size. Considering a machine that employs multiple execution units capable to execute large number of instructions in parallel, the service and storage requirements of each individual instruction are small compared to the total volume of the instruction stream. Individual instructions may then be regarded as *atoms of a fluid* flowing through the pipeline. The objective of this approach is to approximate large buffer levels by continuous fluid levels and decrease state-space complexity. Thus, in the first part of this chapter, we employ an

INTECH
open science | open minds

© 2012 Mitrevski and Kotevski, licensee InTech. This is an open access chapter distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

analytical model based on FSPNs, derive the state equations for the underlying stochastic process and present performance evaluation results to illustrate its usage in deriving measures of interest. The attempt to capture the dynamic behavior of an ILP processor with aggressive use of prediction techniques and speculative execution is a rare example that demonstrates the usage of this recently introduced formalism in modeling actual systems. Moreover, we take into consideration numerical transient analysis and present numerical solution of a FSPN with more than three fluid places. Both the application of finite-difference approximations for the partial derivatives [7,8], as well as the discrete-event simulation of the proposed FSPN model [9,10], allow for the evaluation of a number of performance measures and lead to numerous conclusions regarding the performance impact of predictions and speculative execution with varying parameters of both the microarchitecture and the operational environment. The numerical solution makes possible the probabilistic analysis of the dynamic behavior, whereas the advantage of the discrete-event simulation is the much faster generation of performance evaluation results. Since the modeling framework is implementation-independent, it can be used to estimate the performance potential of branch and value prediction, as well as to assess the operational environment influence on the performance of ILP processors with much more aggressive, wider instruction issue.

Another challenging task in the application of FSPNs is the modeling and performance analysis of *Peer-to-Peer* (P2P) live video streaming systems. Web locations offering live video content increasingly attract more and more visitors, which, if the system is based on the client/server architecture, leads to sustainability issues when clients rise above the upload capabilities of the streaming servers. Since IP Multicast failed to satisfy the requirements of an affordable, large scale live video streaming, in the last decade the science community intensively works in the field of P2P networking technologies for live video broadcast. P2P live video streaming is a relatively new paradigm that aims for streaming live video content to a large number of clients with low cost. Even though many such applications already exist, these systems are still in their early stages and prior to creation of such a system it is necessary to analyze performance via representative model that provides significant insight into the system's behavior. Nevertheless, modeling and performance analysis of P2P live video streaming systems is a complex combinatorial problem, which requires addressing many properties and issues of such systems. Inspired by several research articles concerned with modeling of their behavior, in the second part of this chapter, we present how FSPNs can be used for modeling and performance analysis of a mesh based P2P live video streaming system. We adopt fluid flow to represent bits as *atoms of a fluid* that travel through fluid pipes (network infrastructure). If we represent peers with discrete tokens and video bits as fluid, then we have numerous possibilities to evaluate the performance of the system. The developed model is simple and quite flexible, providing performance evaluation of a system that accounts for a number of system features, such as: network topology, peer churn, scalability, peer average group size, peer upload bandwidth heterogeneity, video buffering, control traffic overhead and admission control for lesser contributing peers. In this particular case, discrete-event simulation (DES) is carried out using SimPy

(http://simpy.sourceforge.net) – an object-oriented, process-based discrete-event simulation language based on standard Python (http://www.python.org), which provides the modeler with components of a simulation model including processes (for active components) and resources (for passive components) and provides monitor variables to aid in gathering statistics.

2. Part A: Fluid atoms in ILP processor pipelines

Most of the recent microprocessor architectures assume *sequential programs* as input and use a *parallel execution* model. The hardware is expected to extract the parallelism out of the instruction stream at run-time. The efficiency is highly dependent on both the hardware mechanisms and the program characteristics, i.e. the *instruction-level parallelism* (ILP) the programs exhibit. Many ILP processors *speculatively* execute control-dependent instructions before resolving the branch outcome. They rely upon *branch prediction* in order to tolerate the effect of *control dependences*. A branch predictor uses the current fetch address to predict whether a branch will be fetched in the current cycle, whether that branch will be taken or not, and what the target address of the branch is. The predictor uses this information to decide where to fetch from in the next cycle. Since the branch execution penalty is only seen if the branch was mispredicted, a highly accurate branch predictor is a very important mechanism for reducing the branch penalty in a high performance ILP processor.

A variety of branch prediction schemes have been explored [11] – they range between *fixed*, *static displacement-based*, *static with profiling*, and various dynamic schemes, like *Branch History Table with n-bit counters*, *Branch Target Address Cache*, *Branch Target Instruction Cache*, *mixed*, *two-level adaptive*, *hybrid*, etc. Some research studies have also proposed concepts to implement high-bandwidth instruction fetch engines based on *multiple branch prediction*. Such concepts include *trace cache* [12] or the more conventional *multiple-block fetching* [13].

On the other hand, given that a majority of static instructions exhibit very little variations in values that they produce/consume during the course of a program's execution [14], *data dependences* can be eliminated at run-time by predicting the outcome values of instructions (*value prediction*) and by executing the true data dependent instructions. In general, the outcome value of an instruction can be assigned to registers, memory locations, condition codes, etc. The execution is *speculative*, as it is not assured that instructions were fed with correct input values. Since the correctness of execution must be maintained, speculatively executed instructions retire only if the prediction was proven correct – otherwise, they are discarded.

Several architectures have been proposed for value prediction [15] – *last value predictor*, *stride predictor*, *context predictors* and *hybrid approaches* in order to get good accuracy over a set of programs due to the different data value locality characteristics that can be exploited only by different schemes. Based on instruction type, value prediction is sometimes identified as prediction of the outcome of *arithmetic instructions* only, and the prediction of the outcome of *memory access instructions* as a different class, referred to as *memory prediction*.

2.1. Model definition

A model should always have a form that is more concise and closer to a designer's intuition about what a model should look like. In the case of a processor pipeline, the simplest description would be that the instructions flow and pass through separate pipeline stages connected by buffers. Control dependences stall the inflow of useful instructions (fluid) into the pipeline, whereas true data dependences decrease the aperture of the pipeline and the outflow rate. The buffer levels always vary and affect both the inflow and outflow rates. Branch prediction techniques tend to eliminate stalls in the inflow, while value prediction techniques help keeping outflow rate as high as possible.

Representing the dynamic behavior of systems subject to randomness or variability is the main concern of *stochastic modeling*. It relies on the use of random variables and their distribution functions [16]. We assume that the distribution of the time between two consecutive occurrences of branch instructions in the fluid stream is exponential with rate λ. The rate depends on the instruction fetch bandwidth, as well as the program's average *basic block size*. Branches vary widely in their dynamic behavior, and predictors that work well on one type of branches may not work as well on others. A set of hard-to-predict branches that comprise a fundamental limit to traditional branch predictors can always be identified [17]. We assume that there are two classes: *easy-to-predict* and *hard-to-predict branches*, and the expected branch prediction accuracy is higher for the first, and lower for the second. The probabilities to classify a branch as either easy- or hard-to-predict depend on the program characteristics.

When the instruction fetch rate is low, a significant portion of data dependences span across instructions that are fetched consecutively [18]. As a result, these instructions (a producer-consumer pair) will eventually initiate their execution in a sequential manner. In this case, the prediction becomes useless due to the availability of the consumer's input value. Hence, in each cycle, an important factor is the number of instructions that consume results of *simultaneously* initiated producer instructions. We assume that the distribution of the time between two consecutive occurrences of consuming instructions in the fluid stream is exponential with rate μ. The rate depends on the number of instructions that simultaneously initiate execution at a functional unit, as well as the program's average *dynamic instruction distance*. We assume that there are two classes of consuming instructions: (1) instructions that consume *easy-to-predict values* and (2) instructions that consume *hard-to-predict values*. The expected value prediction accuracy is higher for the first and lower for the second. The probability to classify a value as either easy- or hard-to-predict depends on the program's characteristics, similarly to the branch classification.

The set of programs executed on the machine represent the *input space*. Programs with different characteristics are executed randomly and independently according to the *operational profile*. We partition the input space by grouping programs that exhibit as nearly as possible homogenous behavior into *program classes*. Since there are a finite number of partitions (classes), the upper limits of λ and μ, as well as the probabilities to classify a branch/value as either easy- or hard-to-predict are considered to be discrete random variables and have different values for different program classes.

2.2. FSPN representation

We assume that the pipeline is organized in four stages: Fetch, Decode/Issue, Execute and Commit. Fluid places P_{IC}, P_{IB}, $P_{RS/LSQ}$, P_{ROB}, P_{RR}, P_{EX} and P_{REG}, depicted by means of two concentric circles (Figure 1), represent buffers between pipeline stages: *instruction cache, instruction buffer, reservation stations and load/store queue, reorder buffer, rename registers, instructions that have completed execution* and *architectural registers*. Five of them have limited capacities: Z_{IBmax}, $Z_{RS/LSQmax}$, Z_{RRmax}, Z_{ROBmax} and Z_{EXmax}. We prohibit both an overflow and a negative level in a fluid place. The fluid place P_{TIME} has the function of an hourglass: it is constantly filled at rate 1 up to the level 1 and then flushed out, which corresponds to the machine clock cycle. $Z_{TIME}(t)$ denotes the fluid level in P_{TIME} at time t. Fluid arcs are drawn as double arrows to suggest a pipe. Flow rates are piecewise constant, i.e. take different values at the beginning of each cycle and are limited by the fetch/issue width of the machine (W). Rates depend on the vector of fluid levels $\mathbf{Z}(t)$ and change when T_{CLOCK} fires and the fluid in P_{TIME} is flushed out. The flush out arc is drawn as thick single arrow.

Let $Z_{IC_0}, Z_{IB_0}, Z_{RS/LSQ_0}, Z_{RR_0}, Z_{ROB_0}$ and Z_{EX_0} be the fluid levels at the beginning of the clock cycle, i.e. $Z_{IC_0} = Z_{IC}(t_0)$, $Z_{IB_0} = Z_{IB}(t_0)$, $Z_{RS/LSQ_0} = Z_{RS/LSQ}(t_0)$, $Z_{RR_0} = Z_{RR}(t_0)$, $Z_{ROB_0} = Z_{ROB}(t_0)$ and $Z_{EX_0} = Z_{EX}(t_0)$, where $t_0 = \lfloor t \rfloor$ and $Z_{TIME}(t_0) = 0$.

A high-bandwidth instruction fetch mechanism fetches up to W instructions per cycle and places them in the instruction buffer. The fetch rate is given by:

$$r_{FETCH} = \min(Z_{IB_{\max}} - Z_{IB_0} + r_{ISSUE}, Z_{IC_0}, W) \tag{1}$$

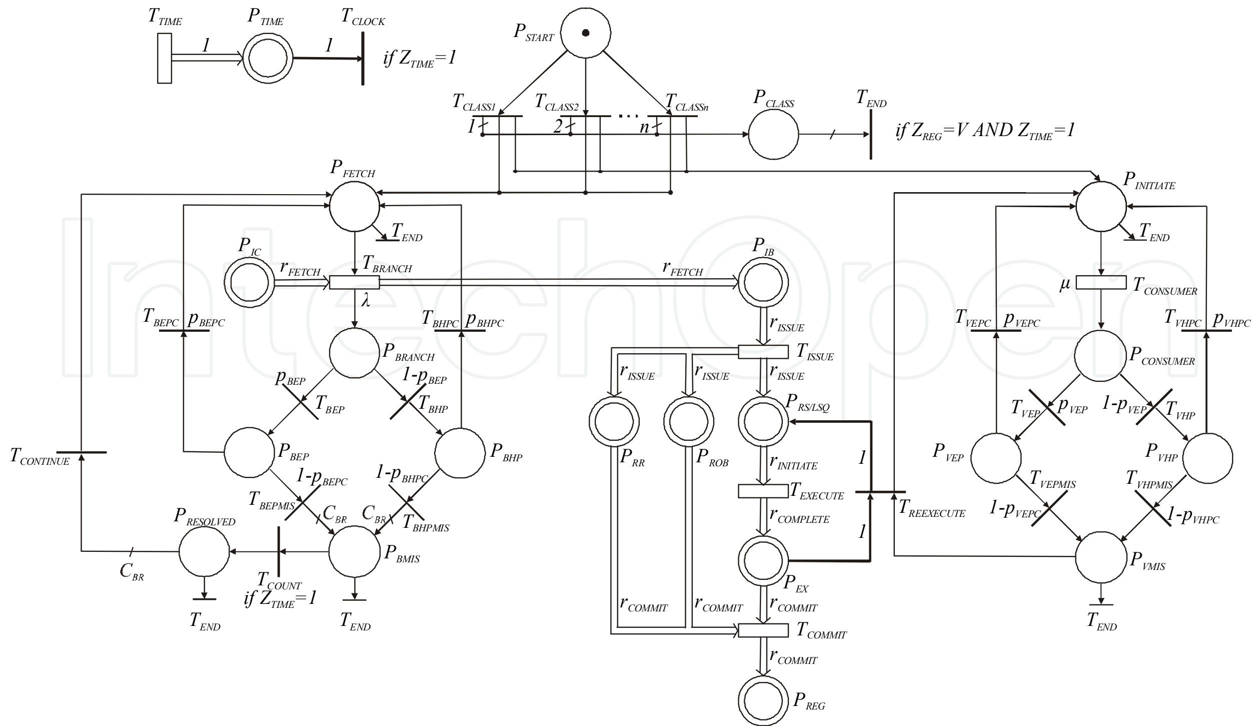


Figure 1. A Fluid Stochastic Petri Net model of an ILP processor

In the case of a branch misprediction, the fetch unit is effectively stalled and no useful instructions are added to the buffer. Instruction cache misses are ignored.

Instruction issue tries to send *W* instructions to the appropriate reservation stations or the load/store queue on every clock cycle. Rename registers are allocated to hold the results of the instructions and reorder buffer entries are allocated to ensure in-order completion. Among the instructions that initiate execution in the same cycle, speculatively executed consuming instructions are forced to retain their reservation stations. As a result, the issue rate is given by:

$$r_{ISSUE} = \min(Z_{RR\max} - Z_{RR0} + r_{COMMIT}, Z_{ROB\max} - Z_{ROB0} + r_{COMMIT}, Z_{RS/LSQ\max} - Z_{RS/LSQ0}, Z_{IB0}, W) \quad (2)$$

Up to *W* instructions are *in execution* at the same time. With the assumptions that functional units are always available and out-of-order execution is allowed, the instructions *initiate* and *complete* execution with rate:

$$r_{INITIATE} = r_{COMPLETE} = \min(Z_{RS/LSQ_0}, W) \quad (3)$$

During the execute stage, the instructions first check to see if their source operands are available (predicted or computed). For simplicity, we assume that the execution latency of each instruction is a single cycle. Instructions execute and forward their own results back to subsequent instructions that might be waiting for them (no result forwarding delay). Every reference to memory is present in the first-level cache. With the last assumption, we eliminate the effect of the memory hierarchy.

The instructions that have completed execution are ready to move to the last stage. Up to *W* instructions may commit per cycle. The results in the rename registers are written into the register file and the rename registers and reorder buffer entries freed. Hence:

$$r_{COMMIT} = \min(Z_{EX_0}, W) \quad (4)$$

In order to capture the relative occurrence frequencies of different program classes, we introduce a set of weighted immediate transitions in the Petri Net. Each program class is assigned an immediate transition T_{CLASS_i} with weight w_{CLASS_i}. The operational profile is a set of weights. The probability of firing the immediate transition T_{CLASS_i} represents the probability of occurrence of a class *i* program, given by:

$$\hat{w}_{T_{CLASS_i}} = \frac{w_{T_{CLASS_i}}}{\sum_{k=1}^{n} w_{T_{CLASS_k}}} \quad (5)$$

A token in P_{START} denotes that a new execution is about to begin. The process of firing one of the immediate transitions randomly chooses a program from one of the classes. The firing of transition T_{CLASS_i} puts *i* tokens in place P_{CLASS}, which identify the class. At the same time instant, tokens occur in places P_{FETCH} and $P_{INITIATE}$, while the fluid place P_{IC} is filled with fluid with volume V_i equivalent to the total number of useful instructions (*program volume*).

Firing of exponential transition T_{BRANCH} corresponds to a branch instruction occurrence. The parameter λ changes at the beginning of each clock cycle and formally depends on both the number of tokens in P_{CLASS} and the fetch rate:

$$\lambda = f\left(\# P_{CLASS}\right)\frac{r_{FETCH}}{W} = f\left(i\right)\frac{r_{FETCH}}{W} = \lambda_i \frac{r_{FETCH}}{W} \tag{6}$$

where λ_i is its upper limit for a given program class *i* at maximum fetch rate ($r_{FETCH}=W$). The branch is classified as easy-to-predict with probability p_{BEP}, or hard-to-predict with probability $1-p_{BEP}$. In either case, it is correctly predicted with probability p_{BEPC} (p_{BHPC}), or mispredicted with probability $1-p_{BEPC}$ ($1-p_{BHPC}$). These probabilities are included in the FSPN model as weights assigned to immediate transitions T_{BEP}, T_{BHP}, T_{BEPC}, T_{BHPC}, T_{BEPMIS} and T_{BHPMIS}, respectively. This approach is known as *synthetic branch prediction.* Branch mispredictions stall the fluid inflow for as many cycles as necessary to resolve the branch (C_{BR} tokens in place P_{BMIS}). Usually, a branch is not resolved until its execution stage ($C_{BR}=3$). With several consecutive firings of T_{CLOCK}, these tokens are consumed one at a time and moved to $P_{RESOLVED}$. As soon as the branch is resolved, transition $T_{CONTINUE}$ fires, a token appears in place P_{FETCH} and the inflow resumes.

Similar to this, firing of exponential transition $T_{CONSUMER}$ corresponds to the occurrence of a consuming instruction among the instructions that initiated execution. The parameter μ changes at the beginning of each clock cycle and formally depends on both the number of tokens in P_{CLASS} and the initiation rate:

$$\mu = g\left(\# P_{CLASS}\right)\frac{r_{INITIATE}}{W} = g\left(i\right)\frac{r_{INITIATE}}{W} = \mu_i \frac{r_{INITIATE}}{W} \tag{7}$$

where μ_i is its upper limit for a given program class *i* when maximum possible number of instructions simultaneously initiate execution ($r_{INITIATE}=W$). The consumed value is classified as easy-to-predict with probability p_{VEP}, or hard-to-predict with probability $1-p_{VEP}$. In either case, it is correctly predicted with probability p_{VEPC} (p_{VHPC}), or mispredicted with probability $1-p_{VEPC}$ ($1-p_{VHPC}$). These probabilities are included in the FSPN model as weights assigned to immediate transitions T_{VEP}, T_{VHP}, T_{VEPC}, T_{VHPC}, T_{VEPMIS} and T_{VHPMIS}, respectively. Whenever a misprediction occurs (token in place P_{VMIS}), the consuming instruction has to be *rescheduled* for execution. The firing of immediate transition $T_{REEXECUTE}$ causes transportation of fluid in zero time. Fluid jumps have deterministic height of 1 (one instruction) and take place when the fluid levels in P_{RS} and P_{EX} satisfy the condition $Z_{RS}(t) \le Z_{RS_{\max}} - 1$ and $Z_{EX}(t) \ge 1$. Jumps that would go beyond the boundaries cannot be carried out. The arcs connecting fluid places and immediate transitions are drawn as thick single arrows. The fluid flow terminates at the end of the cycle when all the fluid places except P_{REG} are empty and T_{END} fires.

2.3. Derivation of state equations

When executing a class *i* program, the nodes m_i of the reachability graph (Figure 2) consist of all the tangible discrete markings, as well as those in which the enabling of immediate

transitions depends on fluid levels and cannot be eliminated, since they are of mixed tangible/vanishing type (Table 1). It is important to note that the number of discrete markings does not depend on the machine width in any way.

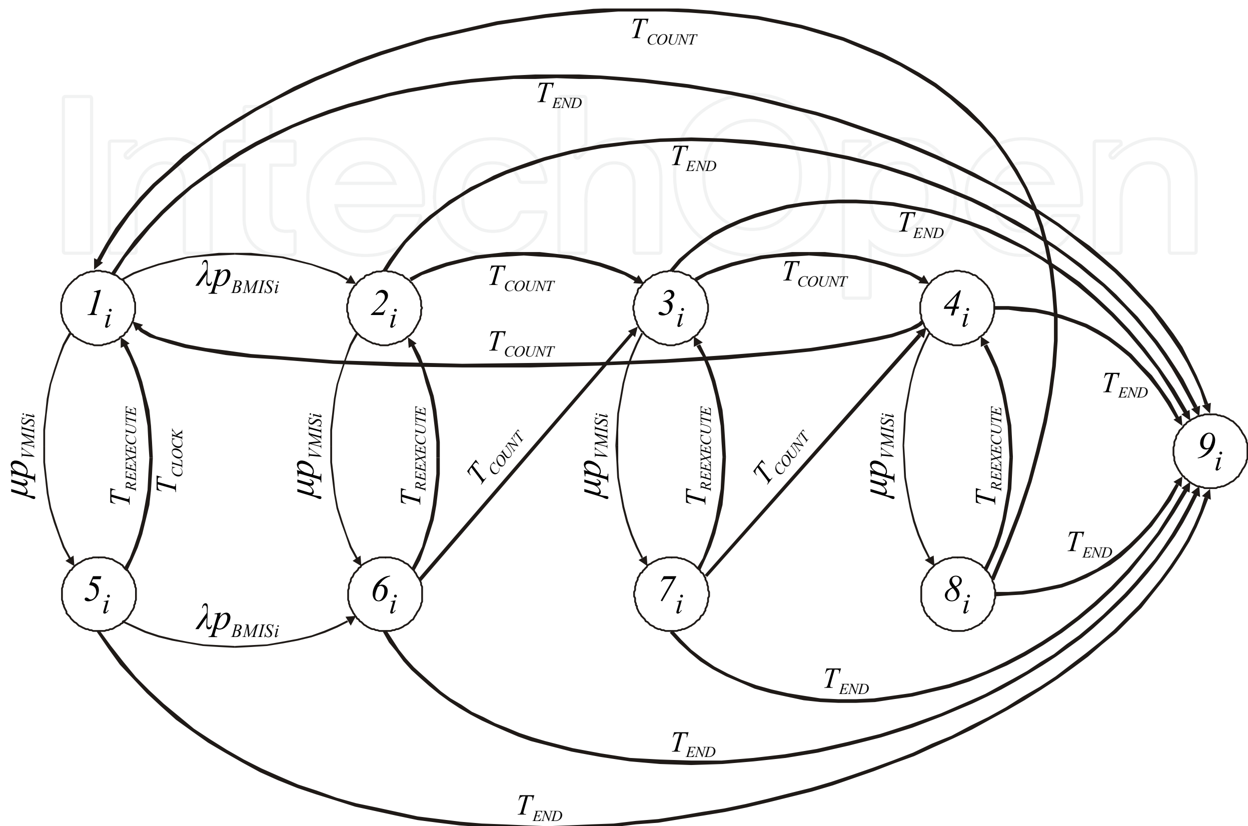


Figure 2. Reachability graph of the FSPN model

Number of tokens / Marking (m_i)	$\#P_{FETCH}$	$\#P_{BMIS}$	$\#P_{INITIATE}$	$\#P_{VMIS}$
1_i	1	0	1	0
2_i	0	3	1	0
3_i	0	2	1	0
4_i	0	1	1	0
5_i	1	0	0	1
6_i	0	3	0	1
7_i	0	2	0	1
8_i	0	1	0	1
9_i	0	0	0	0

Table 1. Discrete markings of the FSPN model (C_{BR}=3)

A vector of fluid levels supplements discrete markings. It gives rise to a stochastic process in continuous time with continuous state space. The total amount of fluid contained in P_{IC}, P_{IB}, $P_{RS/LSQ}$, P_{EX} and P_{REG} is always equal to V_i, and the amount of fluid contained in P_{RR} (as well as

P_{ROB}) is equal to the total amount of fluid in $P_{RS/LSQ}$ and P_{EX}. Therefore, only the fluid levels Z_{IB}(t), $Z_{RS/LSQ}$(t), Z_{EX}(t) and Z_{REG}(t) are identified as four supplementary variables (components of the fluid vector $\mathbf{Z}(t)$), which provide a full description of each state.

The instantaneous rates at which fluid builds in each fluid place are collected in diagonal matrices:

$$\begin{aligned}
&\mathbf{R}_{IB} = \mathbf{diag}\left(r_{FETCH} - r_{ISSUE}, -r_{ISSUE}, -r_{ISSUE}, -r_{ISSUE}, r_{FETCH} - r_{ISSUE}, -r_{ISSUE}, -r_{ISSUE}, -r_{ISSUE}, 0\right)\\
&\mathbf{R}_{RS/LSQ} = \mathbf{diag}\left(r_{ISSUE} - r_{INITIATE}, \ \ldots, r_{ISSUE} - r_{INITIATE}, 0\right)\\
&\mathbf{R}_{EX} = \mathbf{diag}\left(r_{COMPLETE} - r_{COMMIT}, \ \ldots, r_{COMPLETE} - r_{COMMIT}, 0\right)\\
&\mathbf{R}_{REG} = \mathbf{diag}\left(r_{COMMIT}, \ \ldots, r_{COMMIT}, 0\right)
\end{aligned} \tag{8}$$

The matrix of *transition rates* of exponential transitions causing the state changes is:

$$\mathbf{Q}_i = \begin{bmatrix}
-(\lambda p_{BMIS_i} + \mu p_{VMIS_i}) & \lambda p_{BMIS_i} & 0 & 0 & \mu p_{VMIS_i} & 0 & 0 & 0 & 0\\
0 & -\mu p_{VMIS_i} & 0 & 0 & 0 & \mu p_{VMIS_i} & 0 & 0 & 0\\
0 & 0 & -\mu p_{VMIS_i} & 0 & 0 & 0 & \mu p_{VMIS_i} & 0 & 0\\
0 & 0 & 0 & -\mu p_{VMIS_i} & 0 & 0 & 0 & \mu p_{VMIS_i} & 0\\
0 & 0 & 0 & 0 & -\lambda p_{BMIS_i} & \lambda p_{BMIS_i} & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

where:

$$\begin{aligned}
&p_{BMIS_i} = p_{BEP_i}\left(1 - p_{BEPC}\right) + \left(1 - p_{BEP_i}\right)\left(1 - p_{BHPC}\right)\\
&\text{and}\\
&p_{VMIS_i} = p_{VEP_i}\left(1 - p_{VEPC}\right) + \left(1 - p_{VEP_i}\right)\left(1 - p_{VHPC}\right)
\end{aligned} \tag{9}$$

Let π_{m_i} be an abbreviation for the *volume density* $\pi_{m_i}(t, z_{IB}, z_{RS/LSQ}, z_{EX}, z_{REG})$ that is the transient probability of being in discrete marking m_i at time t, with fluid levels in an infinitesimal environment around $\mathbf{z} = [z_{IB} \quad z_{RS/LSQ} \quad z_{EX} \quad z_{REG}]$. If $\boldsymbol{\pi}_i = [\pi_{1_i} \quad \pi_{2_i} \quad \ldots \quad \pi_{9_i}]$, according to [4-6] the evolution of the process is described by a coupled system of nine *partial differential equations* in four continuous dimensions plus time:

$$\frac{\partial \boldsymbol{\pi}_i}{\partial t} + \frac{\partial(\boldsymbol{\pi}_i \cdot \mathbf{R}_{IB})}{\partial z_{IB}} + \frac{\partial(\boldsymbol{\pi}_i \cdot \mathbf{R}_{RS/LSQ})}{\partial z_{RS/LSQ}} + \frac{\partial(\boldsymbol{\pi}_i \cdot \mathbf{R}_{EX})}{\partial z_{EX}} + \frac{\partial(\boldsymbol{\pi}_i \cdot \mathbf{R}_{REG})}{\partial z_{REG}} = \boldsymbol{\pi}_i \cdot \mathbf{Q}_i \tag{10}$$

If $\mathbf{z_0} = \begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}$ is the vector of initial fluid levels, the *initial conditions* are:

$$\begin{aligned}
&\pi_{1_i}(0, \mathbf{z}) = \delta(\mathbf{z} - \mathbf{z_0})\\
&\pi_{m_i}(0, \mathbf{z}) = 0 \quad (2 \le m \le 9)
\end{aligned} \tag{11}$$

Since fluid jumps shift probability mass along the continuous axes (in addition to discrete state change), firing of transition *TREEXECUTE* at time *t* can be seen as a *jump* to another location in the four-dimensional hypercube defined by the components of the fluid vector. It can be described by the following *boundary conditions*:

$$\begin{aligned}
&\pi_{1_i}(t^+,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})=\pi_{1_i}(t^-,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})+\pi_{5_i}(t^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\\
&\pi_{2_i}(t^+,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})=\pi_{2_i}(t^-,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})+\pi_{6_i}(t^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\\
&\pi_{3_i}(t^+,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})=\pi_{3_i}(t^-,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})+\pi_{7_i}(t^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\\
&\pi_{4_i}(t^+,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})=\pi_{4_i}(t^-,z_{IB},z_{RS/LSQ}+1,z_{EX}-1,z_{REG})+\pi_{8_i}(t^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\\
&\pi_{m_i}(t^+,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})=0 \quad (\text{ if } z_{RS/LSQ}\le Z_{RS/LSQ_{\max}}-1,\ z_{EX}\ge 1,\ 5\le m\le 8)
\end{aligned} \tag{12}$$

The firing of transitions *TCLOCK* and *TCOUNT* at time t_0 causes switching from one discrete marking to another. Therefore:

$$\begin{aligned}
\pi_{1_i}(t_0{}^+,z_{IB},z_{RS/LSQ},z_{EX},z_{REG}) &= \pi_{1_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})+\pi_{4_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})+\\
&\quad+\pi_{5_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})+\pi_{8_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\\
\pi_{4_i}(t_0{}^+,z_{IB},z_{RS/LSQ},z_{EX},z_{REG}) &= \pi_{3_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})+\pi_{7_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\\
\pi_{3_i}(t_0{}^+,z_{IB},z_{RS/LSQ},z_{EX},z_{REG}) &= \pi_{2_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})+\pi_{6_i}(t_0{}^-,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\\
\pi_{m_i}(t_0{}^+,z_{IB},z_{RS/LSQ},z_{EX},z_{REG}) &= 0 \quad (\ m\in\{2,5,6,7,8\}\ )
\end{aligned} \tag{13}$$

Similarly, the firing of transition *TEND* when all the fluid places except *PREG* are empty, causes switching from any discrete marking to 9_i:

$$\pi_{9_i}(t_0{}^+,0,0,0,V_i)=\sum_{m=1}^{9}\pi_{m_i}(t_0{}^-,0,0,0,V_i) \qquad \pi_{m_i}(t_0{}^+,0,0,0,V_i)=0 \quad (m\le 8) \tag{14}$$

The *probability mass conservation law* is used as a normalization condition. It corresponds to the condition that the sum of all state probabilities must equal one. Since no particle can pass beyond barriers, the sum of integrals of the volume densities over the definition range evaluates to one:

$$\sum_{m=1}^{9}\int_0^{Z_{IB_{\max}}}\int_0^{Z_{RS/LSQ_{\max}}}\int_0^{Z_{EX_{\max}}}\int_0^{V_i}\pi_{m_i}\left(t,z_{IB},z_{RS/LSQ},z_{EX},z_{REG}\right)dz_{IB}dz_{RS/LSQ}dz_{EX}dz_{REG}=1 \tag{15}$$

Let $M_i(t)$ be the state of the discrete marking process at time *t*. The *probabilities of the discrete markings* are obtained by integrating volume densities:

$$\Pr\{M_i(t)=m_i\}=\int_0^{Z_{IB_{MAX}}}\int_0^{Z_{RS/LSQ_{MAX}}}\int_0^{Z_{EX_{MAX}}}\int_0^{V_i}\pi_{m_i}\left(t,z_{IB},z_{RS/LSQ},z_{EX},z_{REG}\right)dz_{IB}dz_{RS/LSQ}dz_{EX}dz_{REG} \quad (m\le 9) \tag{16}$$

The *fluid levels* at the beginning of each clock cycle are computed as follows:

$$Z_{IB_0}=E\big(Z_{IB}(t_0)\big)=\int_0^{Z_{IB_{\max}}} z_{IB}\underbrace{\left(\int_0^{Z_{RS/LSQ_{\max}}}\int_0^{Z_{EX_{\max}}}\int_0^{V_i}\left(\sum_{m=1}^{9}\pi_{m_i}(t_0,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\right)dz_{RS/LSQ}dz_{EX}dz_{REG}\right)}_{\text{marginal density for } Z_{IB}}dz_{IB}$$

$$Z_{RS/LSQ_0}=E\big(Z_{RS/LSQ}(t_0)\big)=\int_0^{Z_{RS/LSQ_{\max}}} z_{RS/LSQ}\underbrace{\left(\int_0^{Z_{IB_{\max}}}\int_0^{Z_{EX_{\max}}}\int_0^{V_i}\left(\sum_{m=1}^{9}\pi_{m_i}(t_0,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\right)dz_{IB}dz_{EX}dz_{REG}\right)}_{\text{marginal density for } Z_{RS/LSQ}}dz_{RS/LSQ} \quad (17)$$

$$Z_{EX_0}=E\big(Z_{EX}(t_0)\big)=\int_0^{Z_{EX_{\max}}} z_{EX}\underbrace{\left(\int_0^{Z_{IB_{\max}}}\int_0^{Z_{RS/LSQ_{\max}}}\int_0^{V_i}\left(\sum_{m=1}^{9}\pi_{m_i}(t_0,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\right)dz_{IB}dz_{RS/LSQ}dz_{REG}\right)}_{\text{marginal density for } Z_{EX}}dz_{EX}$$

$$Z_{REG_0}=E\big(Z_{REG}(t_0)\big)=\int_0^{V_i} z_{REG}\underbrace{\left(\int_0^{Z_{IB_{\max}}}\int_0^{Z_{RS/LSQ_{\max}}}\int_0^{Z_{EX_{\max}}}\left(\sum_{m=1}^{9}\pi_{m_i}(t_0,z_{IB},z_{RS/LSQ},z_{EX},z_{REG})\right)dz_{IB}dz_{RS/LSQ}dz_{EX}\right)}_{\text{marginal density for } Z_{REG}}dz_{REG}$$

$Z_{IC_0}=V_i-(Z_{IB_0}+Z_{RS/LSQ_0}+Z_{EX_0}+Z_{REG_0})$ and $Z_{RR_0}=Z_{ROB_0}=Z_{RS/LSQ_0}+Z_{EX_0}$.

Finally, the flow rates and the parameters λ and μ are computed as indicated by Eqs. 1-4, 6 and 7, respectively.

2.4. Performance measures

Let τ be a random variable representing the time to absorb into $A=\{m_i \,|\, \pi_{m_i}(t,0,0,0,V_i)=1\}$. The *distribution of the execution time* of a program with volume V_i is:

$$F_{t_{EX_i}}(t)=\Pr\{\tau\le t\wedge M_i(t)\in A\}=\Pr\{M_i(t)=9_i\}=$$
$$=\int_0^{Z_{IB_{MAX}}}\int_0^{Z_{RS/LSQ_{MAX}}}\int_0^{Z_{EX_{MAX}}}\int_0^{V_i}\pi_{9_i}\left(t,z_{IB},z_{RS/LSQ},z_{EX},z_{REG}\right)dz_{IB}dz_{RS/LSQ}dz_{EX}dz_{REG}=\pi_{9_i}(t,0,0,0,V_i) \quad (18)$$

with *mean execution time*:

$$t_{EX_i}=\int_0^{\infty}\left(1-F_{t_{EX_i}}(t)\right)dt \quad (19)$$

Consequently, the *sustained number of instructions per cycle* (*IPC*) is given by:

$$IPC_i=V_i/t_{EX_i} \quad (20)$$

When the input space is partitioned, *IPC* is the ratio between the average volume and the average execution time of all the programs of different classes, as indicated by the operational profile:

$$IPC=\sum_{k=1}^{n}V_k\widehat{w}_{T_{CLASS_k}}\Big/\sum_{k=1}^{n}t_{EX_k}\widehat{w}_{T_{CLASS_k}} \quad (21)$$

The sum of probabilities of the discrete markings that do not carry a token in place P_{FETCH} gives the *probability of a stall* in the instruction fetch unit at time t:

$$P_{STALL_i}(t) = \Pr\{M_i(t) \neq 1_i \wedge M_i(t) \neq 5_i\} = \\ = \sum_{\substack{m \neq 1 \\ m \neq 5}} \int_0^{Z_{IB_{MAX}}} \int_0^{Z_{RS/LSQ_{MAX}}} \int_0^{Z_{EX_{MAX}}} \int_0^{V_i} \pi_{m_i}\left(t, z_{IB}, z_{RS/LSQ}, z_{EX}, z_{REG}\right) dz_{IB} dz_{RS/LSQ} dz_{EX} dz_{REG} \tag{22}$$

Because of the discrete nature of pipelining, additional attention should be given to the probability that no useful instructions will be added to the instruction buffer in the cycle beginning at time t_0 (complete stall in the instruction fetch unit that can lead to an effectively empty instruction buffer) due to branch misprediction. It can be obtained by summing up the probabilities of the discrete markings that still carry one or more tokens in place P_{BMIS} immediately after firing of T_{CLOCK}:

$$P_{NO_FETCH_i}(t_0) = \Pr\{M_i(t_0) = 3_i \vee M_i(t_0) = 4_i\} = \\ = \sum_{m=3}^{4} \int_0^{Z_{IB_{MAX}}} \int_0^{Z_{RS/LSQ_{MAX}}} \int_0^{Z_{EX_{MAX}}} \int_0^{V_i} \pi_{m_i}\left(t_0, z_{IB}, z_{RS/LSQ}, z_{EX}, z_{REG}\right) dz_{IB} dz_{RS/LSQ} dz_{EX} dz_{REG} \tag{23}$$

In addition, the *execution efficiency* is introduced, taken as a ratio between the number of useful instructions and the total number of instructions executed during the course of a program's execution:

$$\eta_{EX_i} = \frac{V_i}{V_i + \begin{pmatrix} \text{instructions reexecuted} \\ \text{due to value misprediction} \end{pmatrix}} \approx \frac{V_i}{V_i + \bar{\mu} \cdot p_{VMIS_i} \cdot t_{EX_i}} = \frac{1}{1 + \dfrac{\mu_i \cdot \bar{r}_{INITIATE_i} \cdot p_{VMIS_i}}{W \cdot IPC_i}} \tag{24}$$

where $\bar{r}_{INITIATE}$ is the *average initiation rate*.

2.5. Numerical experiments and performance evaluation results

We have used *finite difference approximations* to replace the derivatives that appear in the PDEs: *forward* difference approximation for the time derivative and first-order *upwind* differencing for the space derivatives, in order to improve the stability of the method [7,8]:

$$\frac{\partial \pi_{m_i}(t, z_1, ..., z_n)}{\partial t} \approx \frac{\pi_{m_i}(t + \Delta t, z_1, ..., z_n) - \pi_{m_i}(t, z_1, ..., z_n)}{\Delta t} \\ r \frac{\partial \pi_{m_i}(t, z_1, ..., z_k, ..., z_n)}{\partial z_k} \approx r \cdot \operatorname{sgn}(r) \cdot \frac{\pi_{m_i}(t, z_1, ..., z_k, ..., z_n) - \pi_{m_i}(t, z_1, ..., z_k - \operatorname{sgn}(r)\Delta z_k, ..., z_n)}{\Delta z_k} \tag{25}$$

The explicit discretization of the right-hand-side coupling term allows the equations for each discrete state to be solved separately before going on to the next time step. The discretization is carried out on a hypercube of size $Z_{IB_{\max}} \times Z_{RS/LSQ_{\max}} \times Z_{EX_{\max}} \times V_i$ with step

size Δz in direction of z_{IB}, z_{RS}, z_{EX} and z_{REG}, and step size Δt in time. The computational complexity for the solution is

$$O\left(8 \cdot \frac{t}{\Delta t} \cdot \frac{Z_{IB_{\max}} \cdot Z_{RS/LSQ_{\max}} \cdot Z_{EX_{\max}} \cdot V_i}{\Delta z^4}\right) \text{ floating-point operations,}$$

since for each of $t/\Delta t$ time steps we must increment each solution value in the four-dimensional grid for eight of the nine discrete markings. The storage requirements of the algorithm are at least

$$8 \cdot \frac{Z_{IB_{\max}} \cdot Z_{RS/LSQ_{\max}} \cdot Z_{EX_{\max}} \cdot V}{\Delta z^4} \cdot 4 \text{ bytes,}$$

since for eight of nine discrete markings we must store a four-dimensional grid of floating-point numbers (solutions at successive time steps can be overwritten).

Unless indicated otherwise, $Z_{IB_{\max}} = W$, $Z_{RS/LSQ_{\max}} = Z_{RR_{\max}} = Z_{ROB_{\max}} = Z_{EX_{\max}} = 2W$ and $\Delta t = \Delta z/(n \cdot W)$, where n=4 is the number of continuous dimensions. With these capacities of fluid places, virtually all name dependences and structural conflicts are eliminated. Step size Δz is varied between $\Delta z = 1/2$ (coarser grid, usually when the prediction accuracy is high) and $\Delta z = 1/6$ (finer grid, usually when the prediction accuracy is low).

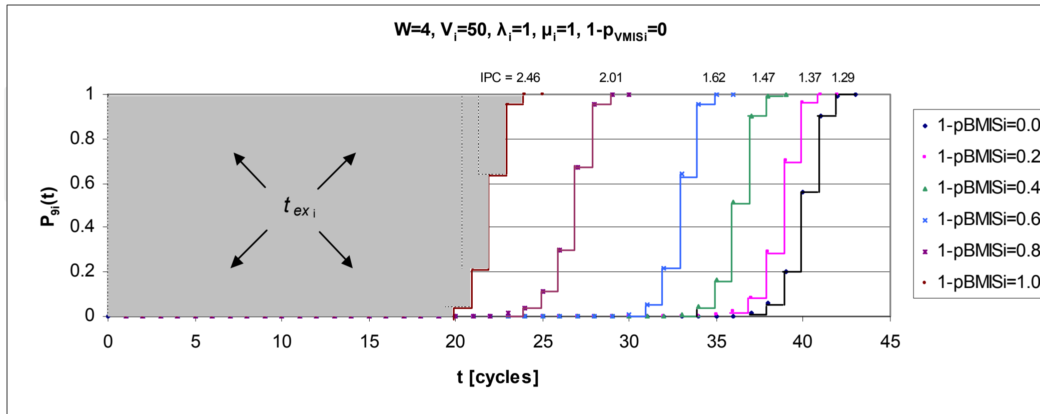
Considering a low-volume program (V_i=50 instructions) executed on a four-wide machine (W=4), we investigate:

- The influence of branch prediction accuracy on the distribution of the program's execution time, when value prediction is not involved (Figure 3a),
- The influence of branch prediction accuracy on the probability of a complete stall in the instruction fetch unit (Figure 3b), and
- The influence of value prediction accuracy on the distribution of the program's execution time, when *perfect* branch prediction is involved (Figure 4).

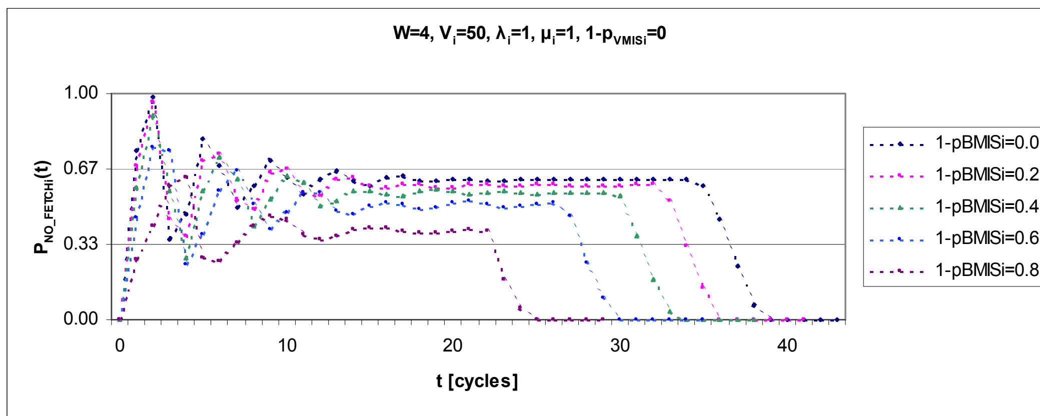
It is indisputably clear that both branch and value prediction accuracy improvements reduce the mean execution time of a program and increase performance. As an illustration, the size of the shaded area in Figure 3a is equal to the mean execution time when perfect branch prediction is involved, and IPC is computed as indicated by Eq. (20). In addition, looking at Figure 3b one can see that the probability of a complete stall in the instruction fetch unit, which can lead to an empty instruction buffer in the subsequent cycle, decreases with branch prediction accuracy improvement. As a result, both the utilization of the processor and the size of dynamic scheduling window increase as branch prediction accuracy increases.

The correctness of the discretization method is verified by comparing the numerical transient analysis results with the results obtained by discrete-event simulation, which is specifically implemented for this model and not for a general FSPN. The types of events that need to be scheduled in the event queue are either *transition firings* or the *hitting of a threshold* dependent on fluid levels. We have used a *Unif[0,1]* pseudo-random number generator to

generate samples from the respective cumulative distribution functions and determine transition firing times via inversion of the *cdf* ("Golden Rule for Sampling"). Discrete-event simulation alone has been used to obtain performance evaluation results for wide machines with much more aggressive instruction issue (*W*>>1).



(a)



(b)

Figure 3. Influence of branch prediction accuracy on (a) the distribution of the program's execution time and (b) the probability of a complete stall in the instruction fetch unit

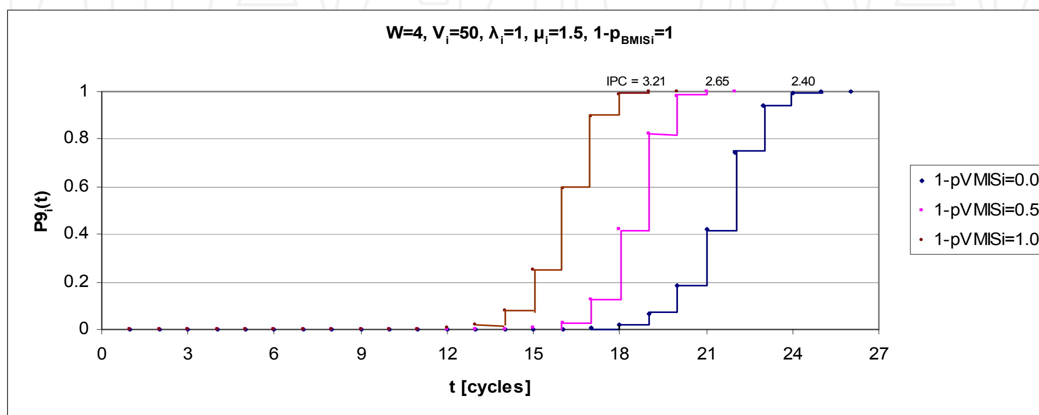


Figure 4. Influence of value prediction accuracy on the distribution of the program's execution time

It takes quite some effort to tune the numerical algorithm parameters appropriately, so that a sufficiently accurate approximation is obtained. Various discretization and convergence errors may cancel each other, so that sometimes a solution obtained on a coarse grid may agree better with the discrete-event simulation than a solution on a finer grid – which, by definition, should be more accurate. In Figures 5a-b, a comparison of discretization results and results obtained using discrete-event simulation for a four-wide machine is given. Furthermore, Figure 5c shows the performance of several machines with realistic predictors executing a program with an average basic block size of eight instructions, given that about 25% of the instructions that initiate execution in the same clock cycle are consuming instructions.

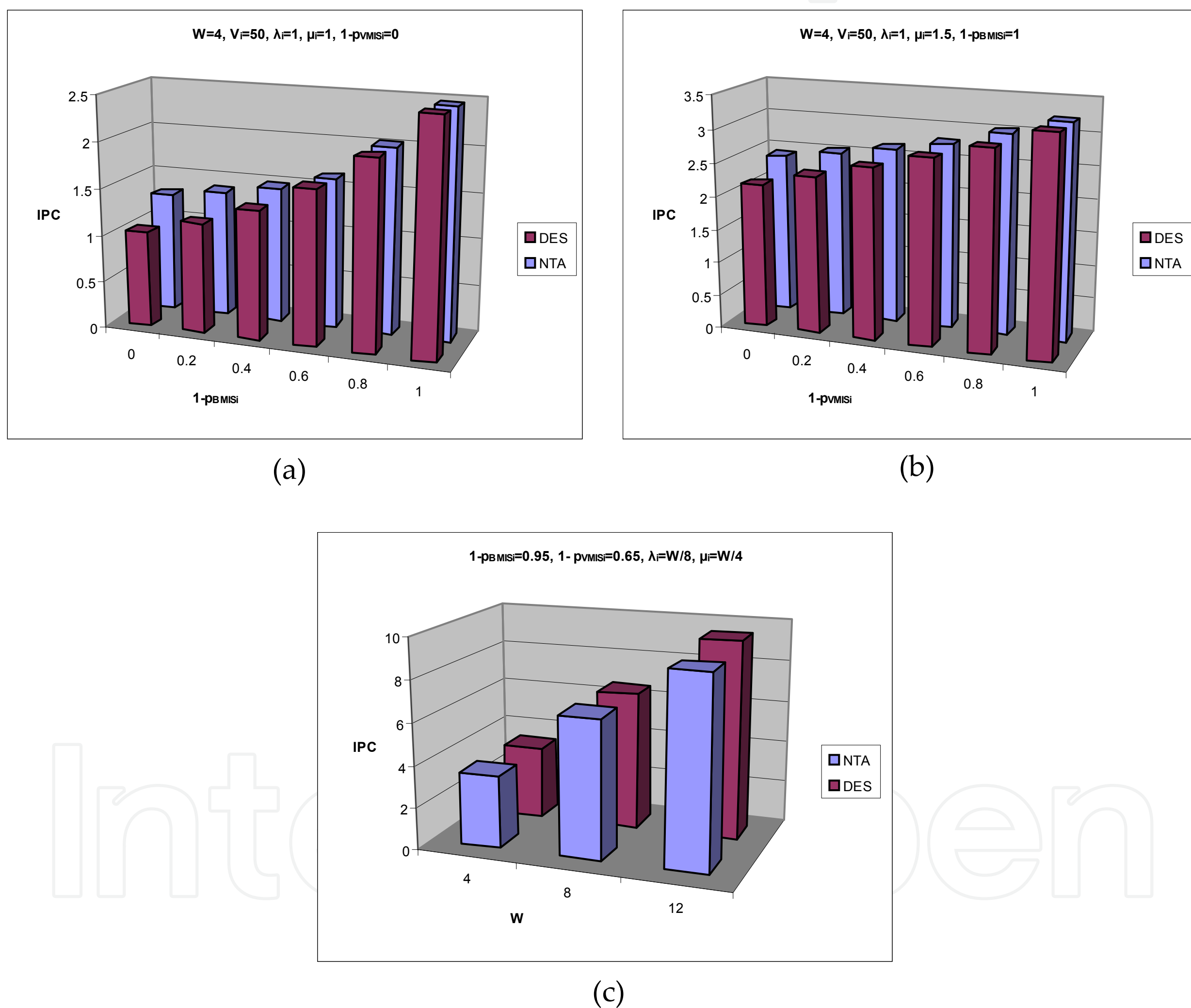


Figure 5. Comparison of numerical transient analysis results (NTA) and results given by discrete-event simulation (DES)

Since the conservation of probability mass is enforced, the differences between the numerical transient analysis and the discrete-event simulation results arise only from the improper distribution of the probability mass over the solution domain. Due to the inherent dissipation error of the first-order accurate numerical methods, the solution at successive

time steps is more or less dissipated to neighboring grid nodes. The phenomenon is emphasized when the number of discrete state changes is increased owing to the larger number of mispredictions.

The results are satisfactorily close to each other, especially when the prediction accuracy is high, which is common in recent architectures. Yet, we believe that much work is still uncompleted and many questions are still open for further research in the field of development of strategies for reducing the amount of memory needed to represent the volume densities, as well as efficient discretization schemes for numerical transient analysis of general FSPNs. *Alternating direction implicit* (ADI) methods [19] in order to save memory, and parallelization of the numerical algorithms to reduce runtime have been suggested.

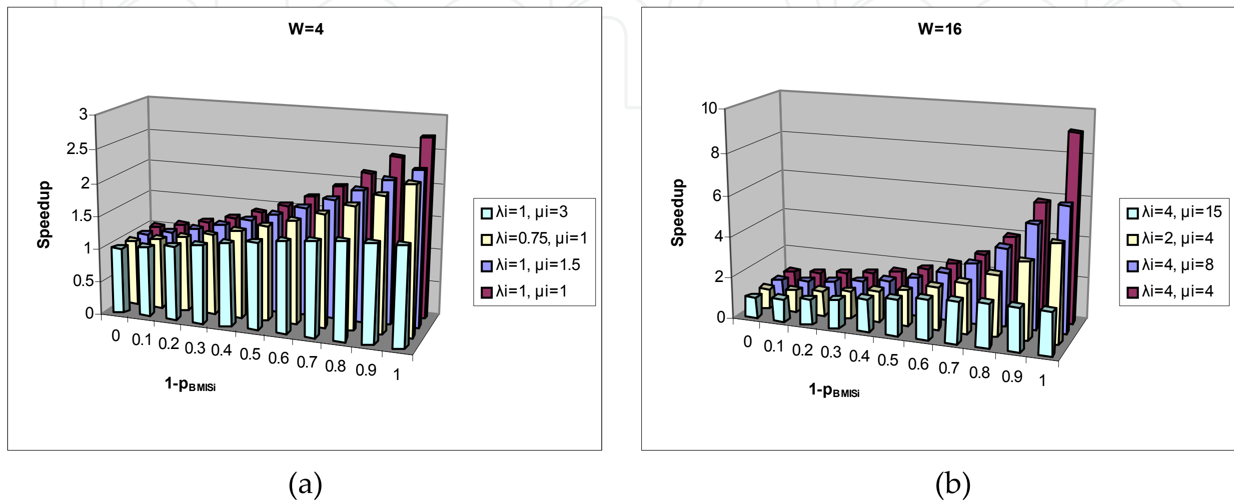
In the remainder of this part, we do not distinguish the numerical transient analysis results from the results given by discrete-event simulation of the FSPN model. Initially we analyze the efficiency of branch prediction by varying branch prediction accuracy. Value prediction is not involved at all. The speedup is computed by dividing the IPC achieved with certain branch prediction accuracy over the IPC achieved without branch prediction ($1-p_{BMIS_i}=0$). For the moment, the input space is not partitioned and program volume is set to V_i=10^6 instructions.

It is observed that, looking at Figures 6a-b, branch prediction curves have an exponential shape. Therefore, building branch predictors that improve the accuracy just a little bit may be reflected in a significant performance increase. The impact of a given increment in accuracy is more noticeable when it experiences a slight improvement beyond the 90%. Another conclusion drawn from these figures is that one can benefit most from branch prediction in programs with relatively short basic blocks (high λ_i / W) and which do not suffer excessively from true data dependences (low μ_i / W). When the ratio μ_i / W is high, true data dependences overshadow control dependences. As a result, the amount of ILP that is expected without value prediction in a machine with extremely aggressive instruction issue is far below the maximum possible value, even with perfect branch prediction. Value prediction has to be involved to go beyond the limits imposed by true data dependences.

Next, we analyze the efficiency of value prediction by varying value prediction accuracy (Figures 7a-b). The speedup is computed by dividing the IPC achieved with certain value prediction accuracy over the IPC achieved without value prediction ($1-p_{VMIS_i}=0$). With perfect branch prediction, it seems clear that the value prediction curves have a linear behavior. Therefore, it is worthwhile to build a predictor that significantly improves the accuracy. Only a small improvement on the value predictor accuracy has a little impact on ILP processor performance, regardless of the accuracy range. Another conclusion drawn from these figures is that the effect of value prediction is more noticeable when a significant number of instructions consume results of simultaneously initiated producer-instructions during execution (high μ_i / W), i.e. when true data dependences have a much higher influence on the program’s total execution time.

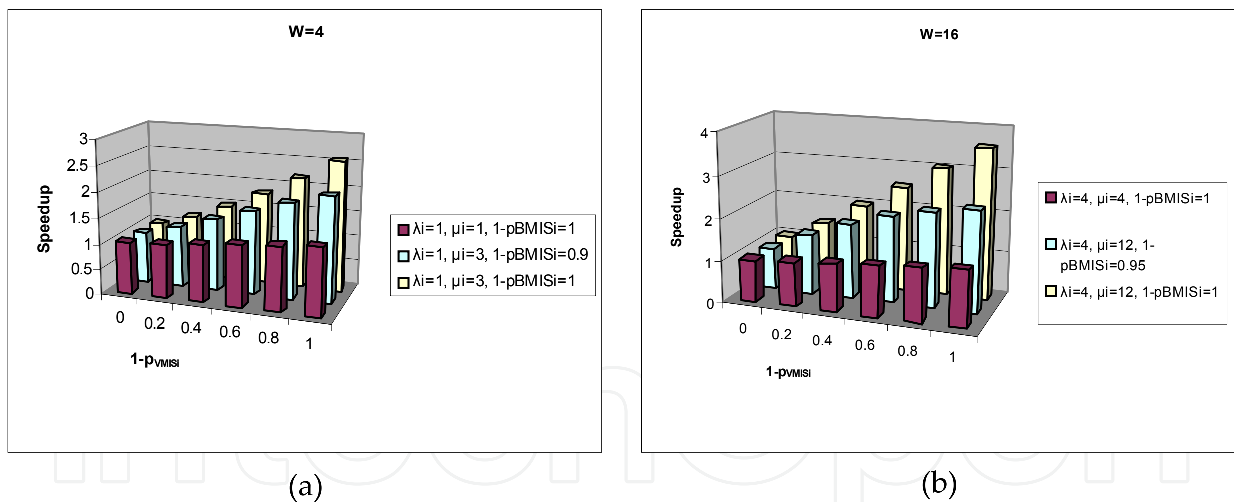
Branch prediction has a very important influence on the benefits of value prediction. One can see that the performance increase is less significant when branch prediction is realistic.

Because mispredicted branches limit the number of useful instructions that enter the instruction window, the processor is able to provide almost the same number of instructions to leave the instruction window, even with lower value prediction accuracy. As a result, graphs tend to flatten out. Correct value predictions can only be exploited when the fetch rate is quite high, i.e. when mispredicted branches are infrequent. Branch misprediction becomes a more significant performance limitation with wider processors (Figure 7b).



(a) (b)

Figure 6. Speedup achieved by branch prediction with varying accuracy

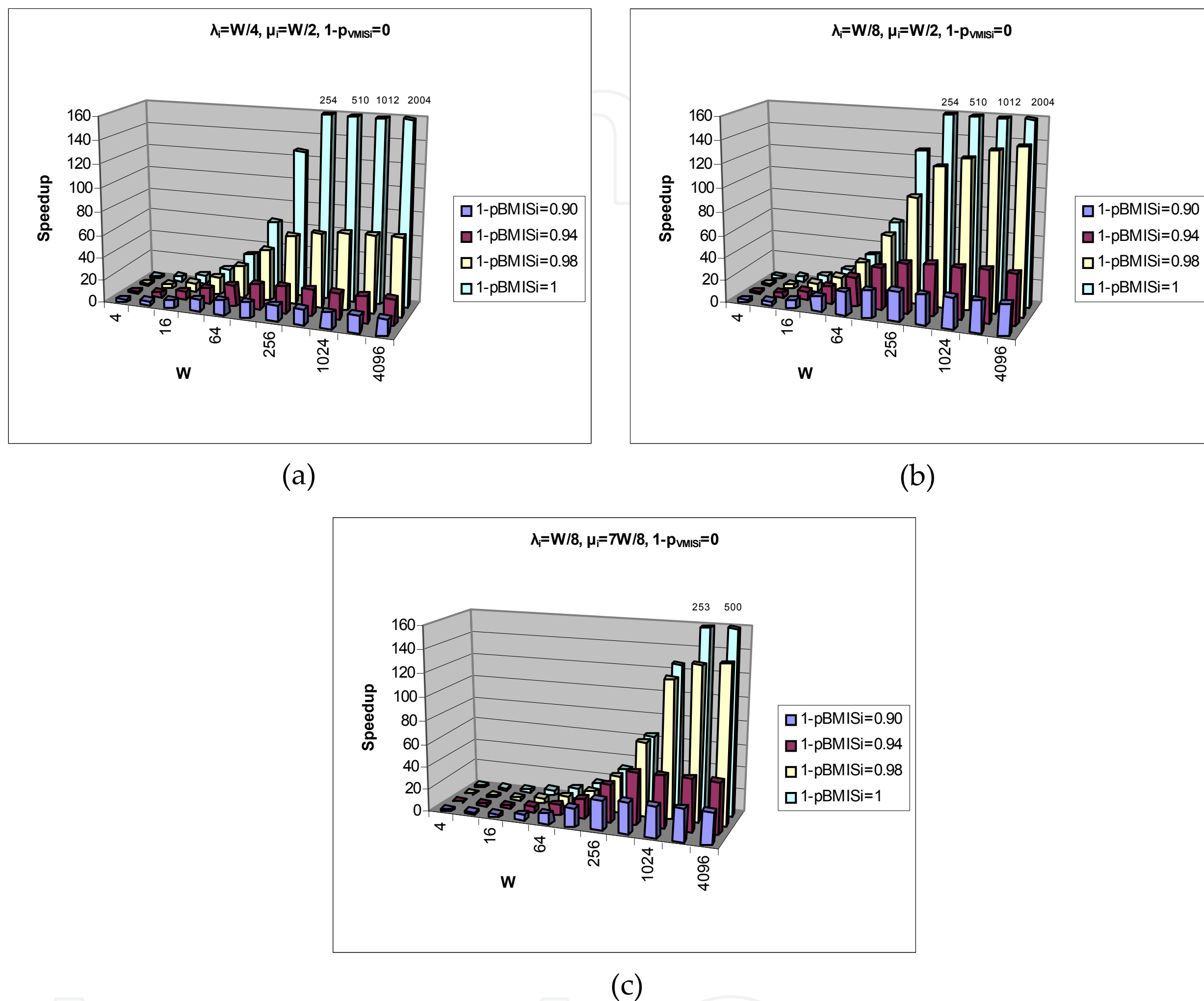


(a) (b)

Figure 7. Speedup achieved by value prediction with varying accuracy

In addition, we investigate branch and value prediction efficiency with varying machine width (Figures 8a-c). The speedup in this case is computed by dividing the IPC achieved in a machine over the IPC achieved in a *scalar* counterpart (W=1, μ_i=0). The speedup due to branch prediction is obviously higher in wider machines. With perfect branch prediction, the speedup unconditionally increases with the machine width. For a given width, the speedup is higher when there are a smaller number of consuming instructions (low μ_i / W). With realistic branch prediction, there is a threshold effect on the machine width: below the

threshold the speedup increases with the machine width, whereas above the threshold the speedup is close to a limit – machine width is by far larger than the average number of instructions provided by the fetch unit. The threshold decreases with increasing the number of mispredicted branches.



(a) (b)

(c)

Figure 8. Speedup achieved by branch prediction with varying machine width

The maximum *additional speedup* that value prediction can provide is computed by dividing the IPC achieved with perfect value prediction over the IPC achieved without value prediction (Figures 9a-c). With perfect branch prediction, some true data dependences can always be eliminated, regardless of the machine width. Actually, the maximum additional speedup is predetermined by the ratio $W/(W-\mu_i)$. However, with realistic branch prediction, the additional speedup diminishes when the machine width is above a threshold value. It happens earlier when there are a smaller number of consuming instructions and/or a larger number of mispredicted branches. In either case, the number of independent instructions examined for simultaneous execution is sufficiently higher than the number of fetched instructions that enter the instruction window. Again, branch prediction becomes more important with wider processors.

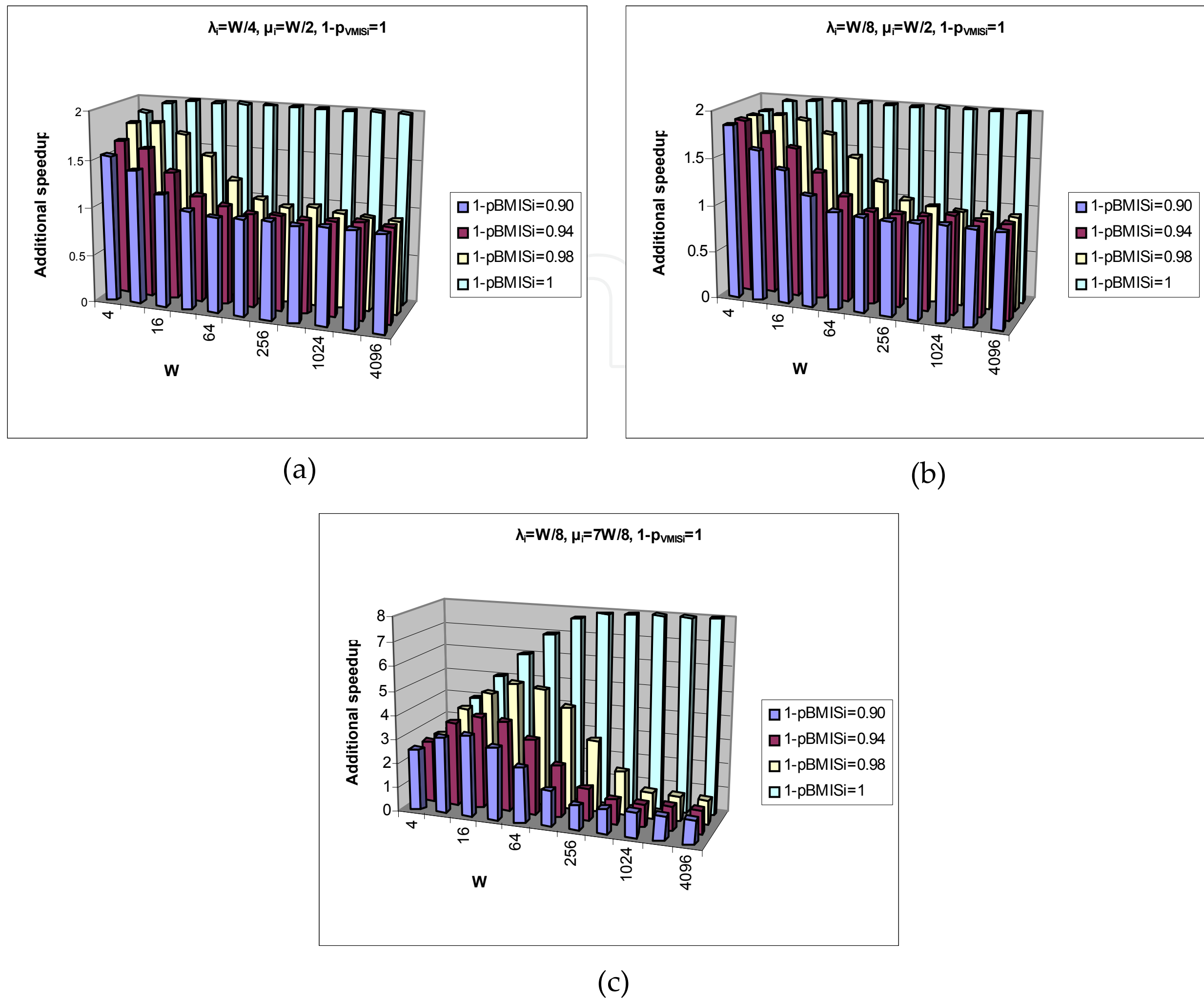
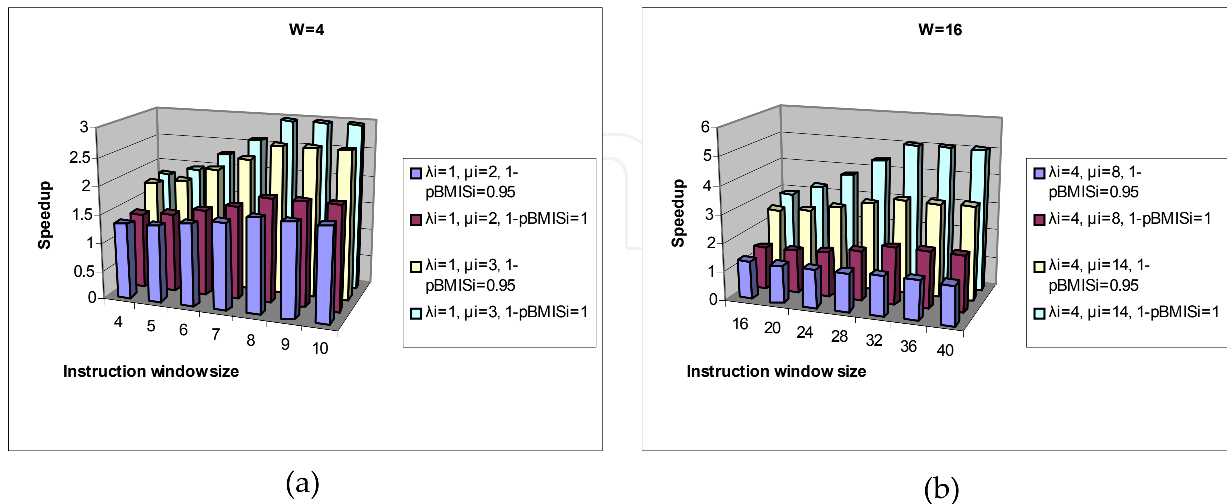


Figure 9. Additional speedup achieved by perfect value prediction with varying machine width

The rate at which consuming instructions occur depends on the initiation rate. Therefore, we also investigate the value prediction efficiency with varying instruction window size (varying capacity $Z_{RS/LSQ_{MAX}}$ of the fluid place $P_{RS/LSQ}$) (Figures 10a-b). The speedup is computed in the same way as in the previous instance. It increases with the instruction window size in [*W*, 2*W*], but the increase is more moderate when there are a smaller number of consuming instructions (low μ_i / W) and/or branch prediction is not perfect. As the instruction window grows larger, performance without value prediction saturates, as does the performance with perfect value prediction. The upper limit value emerges from the fact that in each cycle up to *W* new instructions may enter the fluid place $Z_{RS/LSQ_{MAX}}$ and up to *W* consuming instructions may be forced to retain their reservation stations. One should also note that the speedup for *W*>>1 and realistic branch prediction is almost constant with increasing instruction window size. Two scenarios arise in this case: (1) the number of consuming instructions is large – the speedup is constant but still noticeable as there are not enough independent instructions in the window without value prediction, and (2) the number of consuming instructions is small – there is no speedup as there are enough

independent instructions in the window even without value prediction, regardless of the window size. Again, the main reasons for this behavior are the small number of consuming instructions and the large number of mispredicted branches.



(a) (b)

Figure 10. Speedup achieved by perfect value prediction with varying instruction window size

In order to investigate the operational environment influence, we partitioned the input space into several program classes, each of them with at least one different aspect: branch rate, consuming instruction rate, probability to classify a branch as easy-to-predict or probability to classify a value as easy-to-predict. We concluded that the set of programs executed on a machine have a considerable influence on the *perceived IPC*. Since the term *program* may be interchangeably used with the term *instruction stream*, these observations give good reason for the analysis of the time varying behavior of programs in order to find simulation points in applications to achieve results representative of the program as a whole. From a user perspective, a machine with more sophisticated prediction mechanisms will not always lead to a higher *perceived performance* as compared to a machine with more modest prediction mechanisms but more favorable operational profile [20,21].

3. Part B: fluid atoms in P2P streaming networks

In P2P live streaming systems every user (peer) maintains connections with other peers and forms an application level *logical network* on top of the *physical network*. The video stream is divided in small pieces called *chunks* which are streamed from the source to the peers and every peer acts as a client as well as a server, forwarding the received video chunks to the next peer after some short buffering. The peers are usually organized in one of the two basic types of logical topologies: *tree* or *mesh*. Hence, the tree topology forms structured network of a single tree as in [22], or multiple multicast trees as in [23], while mesh topology is unstructured and does not form any firm logical construction, but organizes peers in *swarming* or gossiping-like environments, as in [24]. To make greater use of their complementary strengths, some protocols use combination of these two aspects, forming a hybrid network topology, such as [25]. Hence, members are free to join or leave the system at their own free will (*churn*), which leads to a certain user driven dynamics resulting in

constant disruptions of the streaming data delivery. This peer churn has high influence on the quality of offered services, especially for P2P systems that offer live video broadcast. Also, P2P network members are heterogeneous in their upload bandwidth capabilities and provide quite different contribution to the overall system performance. Efficient construction of P2P live video streaming network requires data latency reduction as much as possible, in order to disseminate the content in a live manner. This latency is firstly introduced by network infrastructure latency presented as a sum of serialization latency, propagation delay, router processing delay and router queuing delay. The second type of delay is the initial start-up delay required for filling the peer's buffer prior to the start of the video play. The buffer is used for short term storage of video chunks which often arrive out of sequence in manner of order and/or time, and resolving this latency issue requires careful buffer modeling and management. Thus, buffer size requires precise dimensioning because even though larger buffers offer better sequence order or latency compensation, they introduce larger video playback delay. Contrary, small buffers offer smaller playback delay, but the system becomes more error prone. Also, since the connections between participating peers in these P2P logical networks are maintained by the means of control messages exchange, the buffer content (buffer map) is incorporated in these control messages and it is used for missing chunks acquisition. Chunk requesting and forwarding is controlled by a chunk scheduling algorithm, which is responsible for on-time chunk acquisition and delivery among the neighboring peers, which is usually based on the available content and bandwidth of the neighboring peers. A lot of research activities are strictly focused on designing better chunk scheduling algorithms [26,27] that present the great importance of carefully composed scheduling algorithm which can significantly compensate for churn or bandwidth/latency disruptions. Beside the basic coding schemes, in latest years an increasing number of P2P live streaming protocols use *Scalable Video Coding* (SVC) technologies. SVC is an emerging paradigm where the video stream is split in several sub-streams and each sub-stream contributes to one or more characteristics of video content in terms of temporal, spatial and SNR/quality scalability. Mainly, two different concepts of SVC are in greater use: *Layered Video Coding* (LVC) where the video stream is split in several dependently decodable sub-stream called *Layers*, and *Multiple Description Coding* (MDC) where the video stream is split in several independently decodable sub-stream called *Descriptions*. A number of P2P video streaming models use LVC [27] or MDC [23,28] and report promising results.

3.1. Model definition

As a base for our modeling we use the work in [29,30], where several important terms are defined. One of them is the maximum achievable rate that can be streamed to any individual peer at a given time, which is presented in Eq. (26).

$$r_{MAX} = \min\left\{ r_{SERVER}, \frac{r_{SERVER} + \sum_{i=1}^{n} r_{Pi}}{n} \right\} \quad (26)$$

where:
r_{MAX} – maximum achievable streaming rate
r_{SERVER} – upload rate of the server
r_{Pi} – upload rate of the ith peer
n – number of participating peers.

Clearly, r_{MAX} is a function of r_{SERVER}, r_{Pi} and n, i.e. $r_{MAX} = \phi(r_{SERVER}, r_P, n)$. This maximum achievable rate to a single peer is further referred to as the *fluid function*, or $\phi()$. The second important definition is of the term *Universal Streaming*. Universal Streaming refers to the streaming situations when each participating peer receives the video stream with bitrate no less than the video rate, and in [29] it is achievable if and only if:

$$\varphi() \geq r_{VIDEO} \tag{27}$$

where r_{VIDEO} is the rate of the streamed video content.

Hence, the performance measures of the system are easily obtained by calculating the *Probability for Universal Streaming* (P_{US}).

Now, we add one more parameter to the previously mentioned to fulfill the requirements of our model. We define the *stream function* $\psi()$ which, instead of the maximum, represents the *actual* streaming rate to any individual peer at any given time, and $\psi()$ satisfies:

$$\psi() \leq \varphi() \tag{28}$$

3.2. FSPN representation

The FSPN representation of the P2P live streaming system model that accounts for: network topology, peer churn, scalability, peer average group size, peer upload bandwidth heterogeneity, video buffering, control traffic overhead and admission control for lesser contributing peers, is given in Figure 11. We assume asymmetric network settings where peers have infinite download bandwidths, while stream delay, peer selection strategies and chunk size are not taken into account.

Similar as in [29] we assume two types of peers: high contributing peers (HP) with upload bitrate higher than the video rate, and low contributing peers (LP) with upload bitrate lower than the video rate. Different from the fluid function $\phi()$, beside the dependency to r_{SERVER}, r_P, and n, the stream function $\psi()$ depends on the level of fluid in the unique fluid place P_B as well:

$$\psi() = f\left(r_{SERVER}, \# P_{HP}, \# P_{LP}, r_{HP}, r_{LP}, Z_B\right) \tag{29}$$

where Z_B represents the level of fluid in P_B.

The FSPN model in Figure 11 comprises two main parts: the discrete part and the continuous (fluid) part of the net. Single line circles represent discrete places that can contain discrete tokens. The tokens, which represent peers, move via single line arcs to and

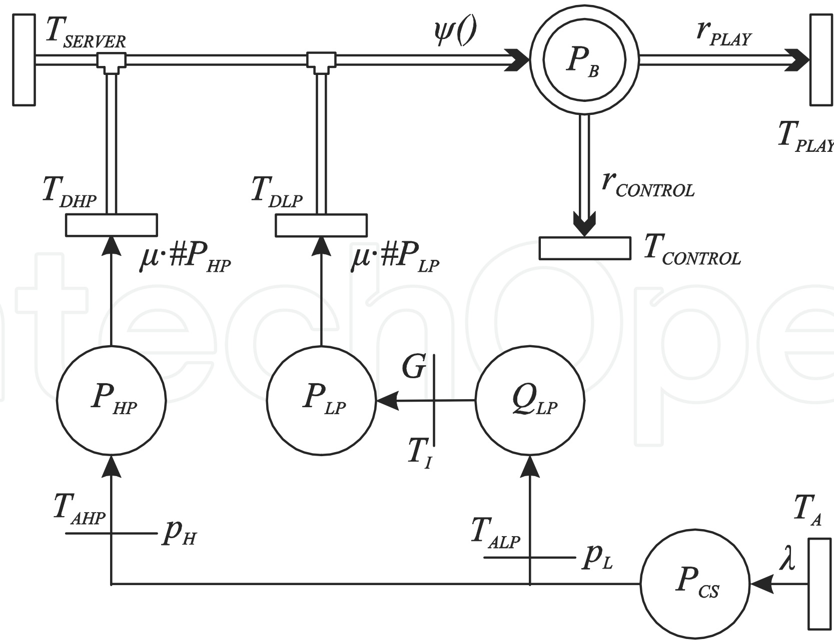


Figure 11. FSPN model of a P2P live video streaming system

out of the discrete places. Fluid arcs, through which fluid is pumped, are drawn as double lines to suggest a pipe. The fluid is pumped through fluid arcs and is streamed to and out of the unique fluid place P_B which represents a single peer buffer. The rectangles represent timed transitions with exponentially distributed firing times, and the thin short lines are immediate transitions. Peer arrival, in general, is described as a stochastic process with exponentially distributed interarrival times, with mean $1/\lambda$, where λ represents the arrival rate. We make another assumption that after joining the system peers' sojourn times (T) are also exponentially distributed. Clearly, since each peer is immediately served after joining the system, we have a queuing network model with an infinite number of servers and exponentially distributed joining and leaving rates. Hence, the mean service time T is equal to $1/\mu$, which transferred to FSPN notation leads to the definition of the departure rate as μ multiplied by the number of peers that are concurrently being served. Now, λ represents peer arrival in general, but the different types of peers do not share the same occurrence probability (p_H and p_L). This occurrence distribution is defined by immediate transitions T_{AHP} and T_{ALP} and their weight functions p_H and p_L. Hence, HP arrive with rate $\lambda_H = p_H * \lambda$, and LP arrive with rate $\lambda_L = p_L * \lambda$, where $p_H + p_L = 1$. In this particular case $p_H = p_L = 0.5$, but, if needed, these occurrence probabilities can be altered. This way the model with peer churn is represented by two independent $M/M/\infty$ Poisson processes, one for each of the different types of peers. The average number of peers that are concurrently being served defines the size of the system as a whole (S_{SIZE}) and is derived from the queuing theory:

$$S_{SIZE} = \lambda/\mu \tag{30}$$

T_A is a timed transition with exponentially distributed firing times that represents peer arrival, and upon firing (with rate λ) puts a token in P_{CS}. P_{CS} (representing the control

server) checks the type of the token and immediately forwards it to one of the discrete places P_{HP} or Q_{LP} (P_{LP}). Places P_{HP} and P_{LP} accommodate the different types of peers in our P2P live streaming system model. Q_{LP} on the other hand, represents queuing station for the LP, which is connected to the place P_{LP} with the immediate transition T_I that is guarded by a *Guard function* G.

The Guard function G is a Boolean function whose values are based on a given condition. The expression of a given condition is the argument of the Guard function and serves as enabling condition for the transition T_I. If the argument of G evaluates to true, T_I is enabled. Otherwise, if the argument of G evaluates to false, T_I is disabled. For the model that does not take admission control into account G is always enabled, but when we want to evaluate the performance of a system that incorporates admission control we set the argument of the guard function as in Eq. (31):

$$G\left\{\frac{r_{SERVER} + \#P_{HP} \cdot r_{HP} + (\#P_{LP} + 1) \cdot r_{LP}}{\#P_{HP} + \#P_{LP}} \geq r_{VIDEO} + r_{CONTROL}\right\} \tag{31}$$

Transitions T_{DHP} and T_{DLP} are enabled only when there are tokens in discrete places P_{HP} and P_{LP}. These are marking dependent transitions, which, when enabled, have exponentially distributed firing times with rate $\mu \cdot \#P_{HP}$ and $\mu \cdot \#P_{LP}$ respectively, where $\#P_{HP}$ and $\#P_{LP}$ represent the number of tokens in each discrete place. Upon firing they take one token out of the discrete place to which they are connected.

Concerning the fluid part of the model, we represent bits as atoms of fluid that travel through fluid pipes (network infrastructure) with rate dependent on the system's state (marking). Beside the stream function as a derivative of several parameters, we identify three separate fluid flows (streams) that travel through the network with different bitrates. The main video stream represents the video data that is streamed from the source to the peers that we refer to as the *video rate* (r_{VIDEO}). The second stream is the play stream which is the stream at which each peer plays the streamed video data, referred to as the *play rate* (r_{PLAY}), and the third stream is the control traffic overhead, referred to as *control rate* ($r_{CONTROL}$), which describes the exchange of control messages needed for the logical network construction and management. As mentioned earlier, transitions T_{DHP} and T_{DLP} are enabled only when there are tokens in discrete places P_{HP} and P_{LP} respectively and beside the fact that they consume tokens when firing, when enabled, they constantly pump fluid through the fluid arc to the fluid place. Flow rates of $\psi()$ are piecewise constant and depend on the number of tokens in the discrete places and their upload capabilities. Continuous place P_B represents single peer's buffer, which is constantly filled with rate $\psi()$ and drained with rate ($r_{PLAY} + r_{CONTROL}$). Z_B is the amount of fluid in P_B and Z_{BMAX} is the buffer's maximum capacity. Transition T_{SERVER} represents the functioning of the server, which is always enabled (except when there are no tokens in any of the discrete places) and constantly pumps fluid toward the continuous place P_B with maximum upload rate of r_{SERVER}. Transition T_{PLAY} represents the video play rate, which is also always enabled and constantly drains fluid from the

continuous place P_B, with rate r_{PLAY}. $T_{CONTROL}$, that represents the exchange of control messages among neighboring peers, is the third transition that is always enabled, has the priority over T_{PLAY}, and constantly drains fluid from P_B with rate $r_{CONTROL}$. For further analysis we derived the rate of $r_{CONTROL}$ from [31] where it is declared that it *linearly* depends on the number of peers in the neighborhood, and for r_{VIDEO} of 128 kbps, the protocol overhead is 2% for a group of 64 users, which leads to a bitrate of 2.56 kbps. Thus, for our performance analysis we assume that peers are organized in neighborhoods with an average size of 60 members where $r_{CONTROL}$ is 2.4 kbps. For the sake of convenience and chart plotting we also define the average upload rate of the participating peers as $r_{AVERAGE}$, which is given in Eq. (32):

$$r_{AVERAGE} = \frac{\# P_{HP} * r_{HP} + \# P_{LP} * r_{LP}}{\# P_{HP} + \# P_{LP}} \tag{32}$$

Since in our model of a P2P live video streaming system we take in consideration $r_{CONTROL}$ as well, Universal Streaming is achievable if and only if:

$$\psi() \geq r_{VIDEO} + r_{CONTROL} \tag{33}$$

3.3. Discrete-event simulation

The FSPN model of a P2P live video streaming system accurately describes the behavior of the system, but suffers from state space explosion and therefore analytic/numeric solution is infeasible. Hence, we provide a solution to the presented model using *process-based discrete-event simulation* (DES) language. The simulations are performed using SimPy which is a DES package based on standard Python programming language. It is quite simple, but yet extremely powerful DES package that provides the modeler with simulation processes that can be used for active model components (such as customers, messages or vehicles), and resource facilities (resources, levels and stores) which are used for passive simulation components that form limited capacity congestion points like servers, counters, and tunnels. SimPy also provides monitor variables that help in gathering statistics, and the random variables are provided by the standard Python random module.

Now, although we deal with vast state space, we provide the solution by identifying four distinct cases of state types. These cases of state types are combination of states of the discrete part and the continuous part of the FSPN, and are presented in Table 2. Hence, the rates at which fluid builds up in the fluid place P_B, in each of these four cases, can be described with linear differential equations that are given in Eq. (34).

case 1	if	$Z_B = Z_{BMAX}$ and $\phi() \geq r_{VIDEO}+r_{CONTROL}$	then	$\psi() = r_{VIDEO}+r_{CONTROL}$ and $r_{PLAY} = r_{VIDEO}$
case 2	if	$0 < Z_B \leq Z_{BMAX}$ *and* $\phi() < r_{VIDEO}+r_{CONTROL}$	then	$\psi() = \phi()$ and $r_{PLAY} = r_{VIDEO}$
case 3	if	$0 \leq Z_{BUF} < Z_{BUFMAX}$ *and* $\phi() \geq r_{VIDEO}+r_{CONTROL}$	then	$\psi() = \phi()$ and $r_{PLAY} = r_{VIDEO}$
case 4	if	$Z_{BUF} = 0$ *and* $\phi() < r_{VIDEO}+r_{CONTROL}$	then	$\psi() = \phi()$ and $r_{PLAY} < r_{VIDEO}$

Table 2. Cases of state types

$$\frac{dZ_B(t)}{dt} = \begin{cases} 0 & case1, \\ \psi() - V_R - C_R & case2, \\ \psi() - V_R - C_R & case3, \\ 0 & case4. \end{cases} \tag{34}$$

In the next few lines (Table 3a-d) we briefly present the definitions of some of the the FSPN model components in SimPy syntax. Algorithm 1 presents the definition of SimPy processes for the different types of tokens. All the FSPN places (as well as r_{PLAY}) are defined as *resource facilities* of the type "Level" and are given in Algorithm 2. The formulation of a "Level" for representing the r_{PLAY} was enforced by the requirement for monitoring and modifying the r_{PLAY} at each instant of time. Algorithm 3 presents T_A combined with T_{AHP} and T_{ALP} where it is defined as two separate SimPy Proceses that independently generate two different types of token processes. Algorithm 4 represents the definition of transitions T_{DHP} and T_{DLP}.

Definition of HP token

```
class tokenHP (Process):
   def join (self):
      yield put, self, Php, 1
```

Definition of LP token with integrated Guard for T_I

```
class tokenLP (Process):
   def join (self):
      if (Php.amount + Plp.amount) == 0:
         yield put, self, Plp, 1
      else:
         yield put, self, Qlp, 1
         def GuardOFF():
                  return (((Rserver + (Plp.amount + 1)*Rlp + Php.amount*Rhp)/((Plp.amount +
                           1) + Php.amount)) >= Rvideo + Rcontrol)
         while True:
            yield waituntil, self, GuardOFF
            yield get, self, Qlp, 1
            yield put, self, Plp, 1
            yield passivate, self
```

(a) Algorithm 1: Definition of tokens in SimPy

```
Pcs = Level (name = 'Control Server', initialBuffered=0, monitored = True)
Php = Level (name = 'Discrete Place Php', initialBuffered=0, monitored = True)
Plp = Level (name = 'Discrete Place Plp', initialBuffered=0, monitored = True)
```

```
Qlp = Level (name = 'Queuing Station', initialBuffered=0, monitored = True)
Pb = Level (name = 'Peer Buffer', initialBuffered=Zbmax, monitored = True)
Pplay = Level (name = 'Play rate', initialBuffered=Rvideo, monitored = True)
```

(b) Algorithm 2: Definition of FSPN places in SimPy

Transition T_A combined with T_{AHP}

```
class HPgenerator (Process):
  def generate (self, end):
    while now() < end:
      yield peerHP = tokenHP ()
      activate (peerHP, peerHP.join())
      yield hold, self, expovariate (pH * Lamda)
```

Transition T_A combined with T_{ALP}

```
class LPgenerator (Process):
  def generate (self, end):
    while now() < end:
      peerLP = tokenLP ()
      activate (peerLP, peerLP.join())
      yield hold, self, expovariate (pL * Lamda)
```

(c) Algorithm 3: Definition of transition T_A combined with T_{AHP} and T_{ALP}

Transition T_{DHP}

```
class HPdeparture (Process):
  def depart (self, end):
    def Condition():
        return (Php.amount > 0)
      while True:
        yield waituntil, self, Condition
        yield hold, self, expovariate (Mi * Php.amount)
        yield get, self, Php, 1
```

Transition T_{DLP}

```
class LPdeparture (Process):
  def depart (self, end):
    def Condition():
        return (Plp.amount > 0)
      while True:
        yield waituntil, self, Condition
        yield hold, self, expovariate (Mi * Plp.amount)
        yield get, self, Plp, 1
```

(d) Algorithm 4: Definition of transitions T_{DHP} and T_{DLP}

Table 3. Definitions of FSPN model components in SimPy

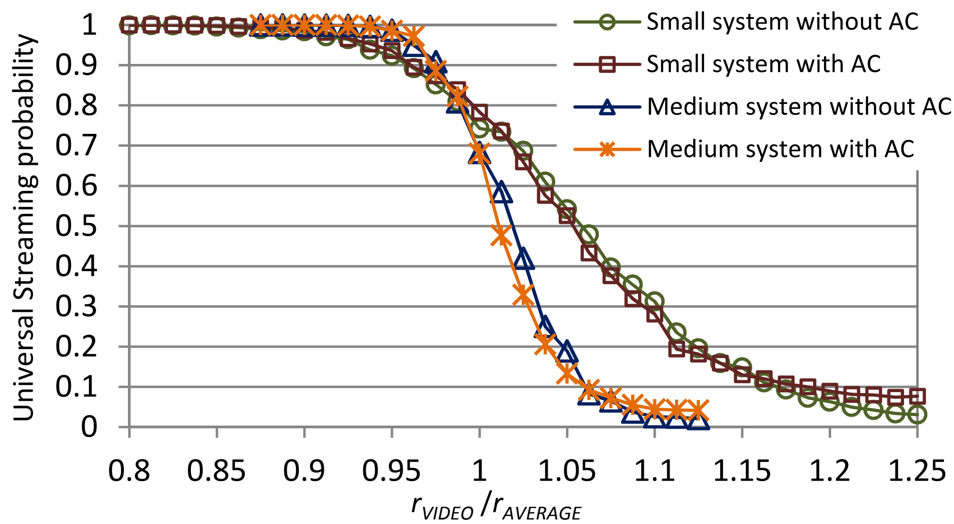


Figure 12. Performance of small and medium systems with and without AC

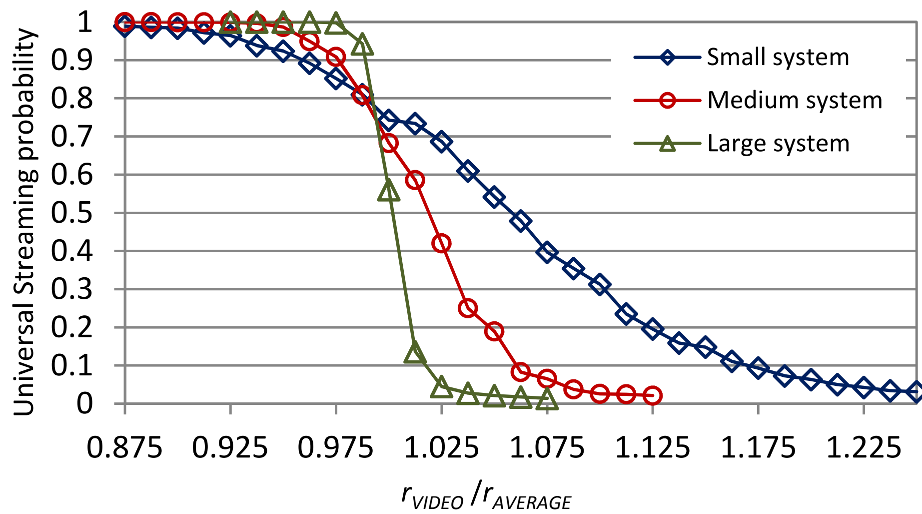


Figure 13. Performance in respect to system scaling

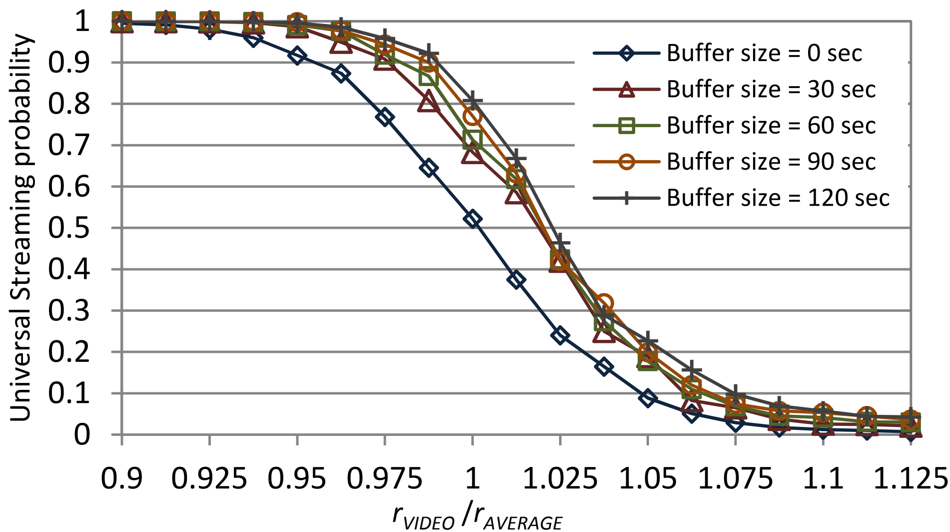


Figure 14. Buffer analysis of medium system without admission control

For simulating the fluid part of the FSPN, time discretization is applied where a SimPy "Stream Processs" checks the system state in small time intervals and consequently makes changes to the level of fluid in the fluid place P_B and r_{PLAY} according to Eq. (34). For gathering the results we use the frequency theory of probability where the probability for Universal Streaming is computed as the amount of time the system spends in Universal Streaming mode against the total simulation time.

3.4. Performance evaluation results and analysis

In this section we make a brief evaluation of three system sizes:

1. Small system with an average of 100 concurrent participating peers
2. Medium system with an average of 500 concurrent participating peers
3. Large system with an average of 5000 concurrent participating peers

The simulation scenario is as follows: $r_{SERVER} = (r_{VIDEO} + r_{CONTROL})*3$, upload bandwidth of HP is r_{HP} = 700kbps, upload bandwidth of LP is r_{LP} = 100kbps, and sojourn time T = 45 minutes. For gathering the performance results we vary the r_{VIDEO} and we plot the P_{US} against the quotient of $r_{VIDEO}/r_{AVERAGE}$, where $r_{AVERAGE}$ for this case is 400kbps. For calculating the P_{US} of a single scenario we calculate the average of 150 simulations for the small system, and an average of 75 simulations for the medium and large system, while each single simulation simulates 10 hours of system activity. Initial conditions are: $Z_{B0} = Z_{BMAX}$, where Z_{B0} is the amount of fluid in P_B in time $t_0 = 0$ and all discrete places are empty.

Comparison of performance of small and medium systems with and without AC is presented in Figure 12, from which an obvious conclusion is inferred that AC almost does not have any direct influence on the performance, but considering the incremented initial delay, incorporation of AC would only have a negative effect on the quality of offered services. Regarding the performance of the system in respect to system scaling, presented in Figure 13, it is obvious that scaling causes increase in performance, but only to a certain point after which performance steeply decreases. Fortunately, the performance decrease is in the region of under capacity which is usually avoided, so it can be concluded that larger systems perform better than smaller ones. Finally, Figure 14 shows that optimal buffer size is about 30 seconds of stored material, and larger buffers only slightly improve performance, but introduce quite large play out delay which leads to diminished quality of user experience.

4. Conclusion

In the first part of this chapter, we have introduced an implementation-independent analytical modeling approach to evaluate the performance impact of branch and value prediction in modern ILP processors, by varying several parameters of both the microarchitecture and the operational environment, like branch and value prediction accuracy, machine width, instruction window size and operational profile. The proposed analytical model is based on recently introduced Fluid Stochastic Petri Nets (FSPNs). We have also

presented performance evaluation results in order to illustrate its usage in deriving measures of interest. Since the equations characterizing the evolution of FSPNs are a coupled system of partial differential equations, the numerical transient analysis poses some interesting challenges. Because of a mixed, discrete and continuous state space, another important avenue for the solution is the discrete-event simulation of the FSPN model. We believe that our stochastic modeling framework reveals considerable potential for further research in this area, needed to better understand speculation techniques in ILP processors and their performance potential under different scenarios.

In the second part of this chapter, we have shown how the FSPN formalism can be used to model P2P live video streaming systems. We have also presented a simulation solution method using process-based discrete-event simulation language whenever analytic/numeric solution becomes infeasible, that is usually a result of state space explosion. We managed to create a model that accounts for numerous features of such complex systems including: network topology, peer churn, scalability, average size of peers' neighborhoods, peer upload bandwidth heterogeneity and video buffering, among which control traffic overhead and admission control for lesser contributing peers are introduced for the first time.

Author details

Pece Mitrevski* and Zoran Kotevski
Faculty of Technical Sciences, University of St. Clement Ohridski, Bitola, Republic of Macedonia

5. References

[1] Rajan R (1995) General Fluid Models for Queuing Networks. PhD Thesis. University of Wisconsin - Madison.
[2] Gribaudo M, Sereno M, Bobbio A (1999) Fluid Stochastic Petri Nets: An extended Formalism to Include non-Markovian Models. Proc. 8th Int. Workshop on Petri Nets and Performance Models. Zaragoza.
[3] Gribaudo M, Sereno M, Horvath A, Bobbio A (2001) Fluid Stochastic Petri Nets Augmented with Flush-out Arcs: Modeling and Analysis. Kluwer Academic Publishers: Discrete Event Dynamic Systems. 11(1/2): 97-117.
[4] Horton G, Kulkarni V, Nicol D, Trivedi K (1998) Fluid Stochastic Petri Nets: Theory, Applications, and Solution. European Journal of Operations Research. 105(1): 184-201.
[5] Trivedi K, Kulkarni V (1993) FSPNs: Fluid Stochastic Petri Nets. In: M. Ajmone Marsan, editor. Lecture Notes in Computer Science: Proc. 14th Int. Conf. on Applications and Theory of Petri Nets. 691: 24-31.
[6] Wolter K, Horton G, German R (1996) Non-Markovian Fluid Stochastic Petri Nets. TU Berlin: TR 1996-13.
[7] Ferziger JH, Perić M (1997) Computational Methods for Fluid Dynamics. Springer-Verlag.

* Corresponding Author

[8] Hoffmann KA, Chiang ST (1993) Computational Fluid Dynamics for Engineers: Volume I & II. Engineering Education System.

[9] Ciardo G, Nicol D, Trivedi K (1997) Discrete-Event Simulation of FSPNs. Proc. 7th Int. Workshop on Petri Nets and performance Models (PNPM'97). Saint Malo. pp. 217-225.

[10] Gribaudo M, Sereno M (2000) Simulation of Fluid Stochastic Petri Nets. Proc. 8th Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. San Francisco. pp. 231-239.

[11] Chang PY, Hao E, Patt Y (1995) Alternative Implementations of Hybrid Branch Predictors. Proc. 28th Annual Int. Symposium on Microarchitecture. Ann Arbor. pp. 252-263.

[12] Rotenberg E, Bennett S, Smith J (1996) Trace Cache: a Low Latency Approach to High Bandwidt Instruction Fetching. Proc. 29th Annual Int. Symposium on Microarchitecture. Paris. pp. 24-35.

[13] Yeh TY, Marr D, Patt Y (1993) Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache. Proc. Int. Conf. on Supercomputing. Tokyo. pp. 67-76.

[14] Lipasti M, Wilkerson C, Shen JP (1996) Value Locality and Load Value Prediction. Proc. 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems. Cambridge. pp. 138-147.

[15] Wang K, Franklin M (1997) Highly Accurate Data Value Prediction using Hybrid Predictors. Proc. 30th Annual Int. Symposium on Microarchitecture. Research Triangle Pk. pp. 281-290.

[16] Milton JS, Arnold JC (1990) Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences (2nd Edition). McGraw-Hill.

[17] Chang PY, Hao E, Yeh TY, Patt Y (1994) Branch Classificaion: a New Mechanism for Improving Branch Predictor Performance. Proc. 27th Annual Int. Symposium on Microarchitecture. San Jose. pp. 22-31.

[18] Gabbay F, Mendelson A (1998) The Effect of Instruction Fetch Bandwidth on Value Prediction. Proc. 25th Int. Symposium on Computer Architecture. Barcelona. pp. 272-281.

[19] Wolter K (1999) Performance and Dependability Modelling with Second Order Fluid Stochastic Petri Nets. PhD Thesis. TU Berlin.

[20] Mitrevski P, Gušev M (2003) On the Performance Potential of Speculative Execution Based on Branch and Value Prediction. Int. Scientific Journal Facta Universitatis. Series: Electronics and Energetics. 16(1): 83-91.

[21] Gušev M, Mitrevski P (2003) Modeling and Performance Evaluation of Branch and Value Prediction in ILP Processors. International Journal of Computer Mathematics. 80(1): 19-46.

[22] Tu X, Jin H, Liao X (2008) Nearcast: A Locality-Aware P2P Live Streaming Approach for Distance Education. ACM Transactions on Internet Technology. 8(2): Article No. 2.

[23] Zezza, S., Magli E, Olmo G, Grangetto M (2009) Seacast: A Protocol for Peer to Peer Video Streaming Supporting Multiple Description Coding. IEEE Int. Conf. on Multimedia and Expo. pp. 1586-1587.

[24] Covino F, Mecella M (2008) Design and Evaluation of a System for Mesh-based P2P Live Video Streaming. ACM Int. Conf. on Advances in Mobile Computing and Multimedia. pp. 287-290.

IntechOpen
IntechOpen

[25] Lu Z, Li Y, Wu J, Zhang SY, Zhong YP (2008) MultiPeerCast: A Tree-mesh-hybrid P2P Live Streaming Scheme Design and Implementation based on PeerCast. 10th IEEE Int. Conf. on High Performance Computing and Communications. pp. 714-719.
[26] Chen Z, Xue K, Hong P (2008) A Study on Reducing Chunk Scheduling Delay for Mesh-Based P2P Live Streaming. In: 7th IEEE Int. Conf. on Grid and Cooperative Computing, pp. 356-361.
[27] Xiao X, Shi Y, Gao Y (2008) On Optimal Scheduling for Layered Video Streaming in Heterogeneous Peer-to-Peer Networks. ACM Int. Conf. on Multimedia. pp. 785-788.
[28] Guo H, Lo KT (2008) Cooperative Media Data Streaming with Scalable Video Coding. IEEE Transactions on Knowledge and Data Engineering. 20(9): 1273-1281.
[29] Kumar R, Liu Y, Ross K (2007) Stochastic Fluid Theory for P2P Streaming Systems. IEEE INFOCOM. pp. 919–927.
[30] Kotevski Z, Mitrevski P (2011) A Modeling Framework for Performance Analysis of P2P Live Video Streaming Systems. In: Gušev M, Mitrevski P, editors. ICT Innovations 2010. Berlin Heidelberg: Springer Verlag. pp. 215-225.
[31] Chu Y, Rao SG, Seshan S, Zhang H (2000) A Case for End System Multicast. IEEE Journal on Selected Areas in Communications. 20(8): 1456–1471.

IntechOpen