A Fast Yet Accurate Message-level Communication Bus Model for Timing Prediction of SDFGs on MPSoC

Hai-Dang Vu, S. Le Nours, S. Pillement
University of Nantes, IETR UMR CNRS 6164, France
{hai-dang.vu}@univ-nantes.fr

Ralf Stemmer, Kim Grüttner
OFFIS e.V., Germany
{ralf.stemmer}@offis.de

ABSTRACT

Fast yet accurate performance and timing prediction of complex parallel data flow applications on multi-processor systems remains a difficult discipline. The reason for it comes from the complexity of the data flow applications and the hardware platform with shared resources, like buses and memories. This combination may lead to complex timing interferences that are difficult to express in pure analytical or classical simulation-based approaches. In this work, we propose a message-level communication model for timing and performance prediction of Synchronous Data Flow (SDF) applications on MPSoCs with shared memories. We compare our work against measurement and TLM simulation-based performance prediction models on two case-studies from the computer vision domain. We show that the accuracy and execution time of our simulation outperforms existing approaches and is suitable for a fast yet accurate design space exploration.

CCS CONCEPTS

• **Computer systems organization** → *Embedded software*; *Multicore architectures*; • **Computing methodologies** → **Modeling methodologies**.

KEYWORDS

System-level modeling, Multi Processor, Timing prediction

ACM Reference Format:
Hai-Dang Vu, S. Le Nours, S. Pillement and Ralf Stemmer, Kim Grüttner. 2021. A Fast Yet Accurate Message-level Communication Bus Model for Timing Prediction of SDFGs on MPSoC. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431418

1 INTRODUCTION

Because of the growing demand in computational efficiency, more and more embedded systems are designed on multi-processor platforms. Validation of software running on such platforms is fundamental to guarantee functional correctness and that timing requirements (*e.g.*, response time, latency) are fully met. Intensive analysis of hardware/software architectures under potential working scenarios is thus mandatory to prevent costly design cycles. However, in the context of software performance prediction on multi-processor platforms, creation of high abstraction level models with good prediction accuracy at affordable analysis times represents a key challenge. One issue that makes creation of efficient performance models a time-consuming effort is due to complex interactions among shared resources (*e.g.*, shared memories, shared communication buses).

In this paper, we present a high abstraction level communication model for fast yet accurate timing prediction of data flow applications on tile-based multi-processor platforms with shared memory for inter tile communication. We aim to reduce the simulation runtime of analyzing multiple FIFO communication channels mapped on a shared bus with a shared memory. We adopt an analytical model to formulate the time dependencies between the elementary communication phases taking into account potential penalty delays due to contention at shared resources. It is used in a message-level simulation model of the communication infrastructure which combination demonstrates then good scalability for performance prediction of different possible mappings. We systematically evaluated the benefits and limitations of our created models for timing prediction of two different data flow applications mapped on a tile-based multi-processor platform. This setup was used for different case-studies and mapping configurations. For each studied mapping, we compared the obtained simulation results of our proposed message-level communication model against detailed transaction-level models and measurements on a real prototype. The evaluation metrics were accuracy of the estimations and analysis time.

The paper is organized as follows. In the next section, we compare to existing system-level modeling approaches for communication resources. In Section 3, we provide an overview of the main concepts associated with our approach. Section 4 presents our contribution to communication modeling. In Section 5, we present our use-cases, the experimental setup, our comparison results and finally a discussion of these results. The paper closes with a summary and an outlook on future work.

2 RELATED WORK

Different simulation-based approaches have been proposed to evaluate multi-processor system performance early in the design process [5]. In the proposed approaches, performance models are formed by considering an application workload model expressing computation and communication loads that an application causes when executed on platform resources. Captured performance models are then translated in a system-level design (SLD) language such as SystemC [1] to provide some estimations through simulation. In this domain, different approaches have been proposed to deliver fast yet accurate models of communication resources. The result-oriented modeling (ROM) approach has been proposed by Shirner

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431418

and Dömer in [12] to optimize the simulation speed-accuracy trade-off. It optimistically predicts the communication durations using available information about resources. The estimated durations are then adapted during simulation in the case of disturbing influences such as pre-emption of shared resources. In our adopted approach, the communication durations are not estimated but computed during simulation. Compared to ROM this allows more simulation events to be saved and potentially better simulation efficiency. In [3], Bobrek et al. propose an approach to raise the abstraction level of simulation while still estimating contention at shared resources such as memories and buses. It interleaves an event-driven simulation to coarsely capture processor execution overhead with an analytical stochastic model used to estimate contention for shared resources. In our work, we also combine simulation with analytical expression of shared resource effects on communication. This approach allows time-consuming simulation events to be significantly reduced while maintaining good level of accuracy. A similar combination is presented in [11] to address the timing accuracy problem in temporally decoupled transaction level models. Delay formulas are proposed according to shared resource usage and availability. They are then used during simulation to adapt the end of each synchronization request. Our presented analytical model is sensitive to the simulated situation and thus our contention delay model can dynamically adapt to communication changes. A similar adaptation approach was considered in [4] but for a different bus protocol. In our case, we concentrate on FIFO channel communications mapped on a shared bus with first-come first served bus arbitration protocol. In previous work [6], this approach was adopted to the modeling of computation resources. In the scope of this paper, we focus on communication resources modeling and we further evaluate the simulation speed and accuracy delivered by the considered modeling approach.

3 PRELIMINARIES

In this section, we provide a brief review of the main concepts, assumptions and constraints associated with our approach. Fig. 1 illustrates our overall system model with a Sobel filter as a running example. We adopt a tile-based platform organization where each tile consists of a processing element (PE) with private instruction and data memories, and a communication interface (IF). A tile can thus execute software without interfering with other tiles as long as the software does not explicitly access shared resources. An execution platform consists of a finite set of tiles, a finite set of shared memories and a shared communication bus. In the scope of our work, we consider a first-come-first-served (FCFS) bus arbitration protocol. Shared memory is used to temporary store channel buffers exchanged between actors of the application. In the example of Fig. 1, the five channels of the modeled application, denoted by C_0 to C_4, are thus communicated through the communication bus and the shared memory. In this paper, communications will be modeled at two different abstraction levels. At transaction level (TL), communications are described with bus transaction granularity and the bus arbitration is expressed for each bus transaction. At message level (ML), communications are described at the application data granularity. Timing is simulated by a single *wait-for-time* statement and arbitration is not explicitly modeled.

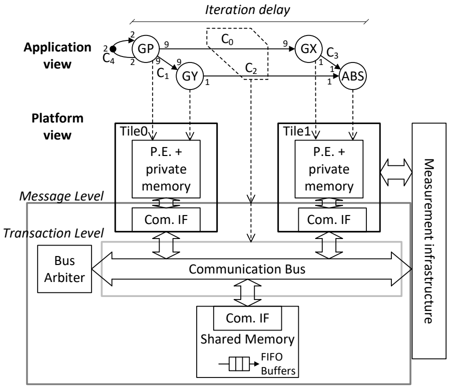

Figure 1: Illustration of studied applications and tile-based platforms with shared communication bus and memory.

We use synchronous data flow (SDF) as model of computation [7]. This model describes the application data flow between *actors* via communication *channels*. The SDF graph of the Sobel filter example given in the upper part of Fig. 1 consists of four actors and five channels. This representation offers a strict separation of computation (*Compute*) and communication (*WriteTokens*, *ReadTokens*) statements. Channels are FIFO buffers that can be mapped to any memory that is accessible by the producer and the consumer. During the read phase of an actor, tokens get read from a channel buffer. During the write phase, tokens get written into a channel. The amount of tokens that a producer writes or a consumer reads is called *token rate*. It is denoted by integer values on the SDF graph of Fig. 1. As FIFO access is blocking, an actor can only switch to its computation phase after it reads all tokens from all incoming channels. In our SDF model, we assume that all tokens have a same size which is exact one data word. After the computation phase, the actor switches into write phase to write all tokens to the outgoing channels. In our work, all channels buffers are mapped onto shared memory, while the actors use the private memory of the tiles they get executed on. This setup is fully composable such that the computation phases of any actor can be considered independent from communication phases.

Communication is done by two functions used in the implemented software: *WriteTokens* that writes tokens onto the shared memory, and *ReadTokens* that reads from shared memory. The state-transition diagram of Fig. 2 gives the observed behavior of the write process of *n* tokens[1]. The states denoted by *polling*, *writing* and *update buffer usage* correspond to the situations when the PE accesses to the shared resources. The states denoted by *initialization*, *inter-polling stage*, *pre-writing*, *inter-writing stage*, *post-writing* correspond to the execution on the PE without interfering with another resource. At the beginning of the tokens write process, once initialization done, the polling step aims to verify the availability of the buffer. If the buffer is ready to write, the actor prepares to copy the data onto the buffer. Otherwise, it continues polling until the condition is satisfied. A delay exists between two consecutive polling states, it corresponds to the duration of the inter-polling

[1] In the behavior description notation, circles correspond to states and thick lines to transitions between states. Rectangles in grey emphasize the states where shared resources are accessed.

state. The tokens write phase finishes when all the data are copied onto the buffer. At the end of the tokens write process, the PE finally updates the usage of the buffer to indicate that the data is ready for the read process. The tokens read process is implemented with a similar sequence of elementary states.

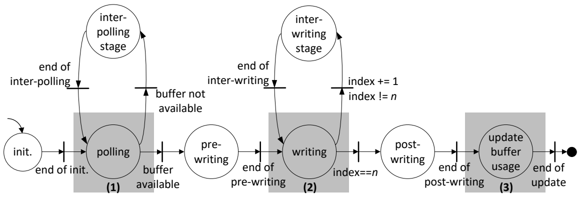

Figure 2: State-transition diagram of the write protocol of n tokens to the shared memory.

For our performance models we need to characterize the duration of the computation phase of each actor as well as the duration of the communication between actors. For the computation phase, we measure the amount of clock cycles a phase needs to be executed. For the communication phases, the adopted measurement infrastructure is able to monitor and store the signals that are relevant to estimate the communication durations. It is used to measure the durations of each elementary states involved in communication, as for example the ones identified in Fig. 2, and delays caused by contention at shared resources. Further details about the measurement infrastructure will be given in Section 5.

4 PROPOSED COMMUNICATION MODEL

4.1 Principle

In this paper, we adopt a modeling approach to allow accurate estimation of communication timings with a limited simulation run-time. This approach consists in associating a message-level simulation model of the mapped communication channels with an analytical expression of the communication durations. Fig. 3 illustrates the main idea of the proposed approach through the execution over time of a n tokens write function.

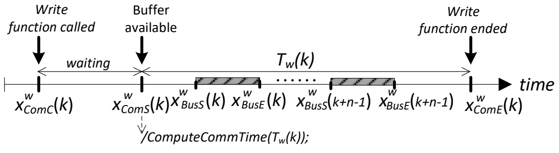

Figure 3: Principle of the proposed message-level model with computation of communication duration.

At message level, three significant instants are exhibited[2]:

- Instant when tokens communication function is called, denoted by x^w_{ComC} in Fig. 3. This is followed with a waiting state which corresponds to the duration until buffer is available for tokens write.
- Instant when buffer is available, denoted by x^w_{ComS}. This would correspond to the instant when data is available in the case of a read tokens function.
- Instant when tokens communication ends, denoted by x^w_{ComE}.

In the trace-driven simulation approach presented in [9], the platform model is driven by the execution of the application through traces that represent computation and communication workloads. During system model simulation, limited capacity of platform resources influences when traces are processed and thus the execution of the application. Our considered hybrid simulation approach differs from trace-driven simulation in the way simulation is performed. As illustrated in Fig. 3, communication durations and instants when shared resources are used are computed during simulation. Computation is denoted by action `ComputeCommTime()`. Communication duration is denoted by T_w and instants denoted by x^w_{BusS} and x^w_{BusE} correspond to the successive instants when the communication bus is accessed during the tokens write process. When multiple tokens communications are simulated, instants when shared resources are used are considered to determine if contention situations occur. In that case, instants are adapted accordingly to appropriately set the communication duration. It is thus possible to limit the number of calls to the simulation kernel but with still accuracy about communication resource usage.

4.2 Application to FCFS arbitration policy

Based on the example of Fig. 1, Fig. 4 represents two simultaneous communications through a shared communication bus and a shared memory. It illustrates the situation of writing n tokens on channel C_1 and reading n tokens from channel C_0 with the protocol described in Fig. 2. In part (a) of Fig. 4, each elementary step of the communications is described with a specific duration called *elementary delay*[3]. Each instant denoted by $x^w_j(k)$ (respectively, $x^r_j(k)$) corresponds to the k-th transition instant between two successive elementary steps of the tokens write process (respectively, the tokens read process)[4]. Two phases especially cause many time-consuming calls to the simulation kernel: the polling phase and the successive elementary reading or writing phases. Contention effects at shared resources are illustrated in part (a) of Fig. 4. The situations denoted by (1), (2), and (3) are related to the states highlighted in grey in Fig. 2. The situation denoted by (1) corresponds to a contention due to two concurrent polling steps. The communication bus arbitration allows the polling step of the tokens read process to execute first and the polling step of the tokens write process has to wait until the end of the read polling phase. The situation denoted by (2) corresponds to the situation where the elementary writing phase is delayed due to a simultaneous elementary reading phase. The situation denoted by (3) corresponds to the situation where the C_1 update usage phase is delayed due to simultaneous C_0 update usage phase.

In the scope of this work, we adopt a message-level communication model which simulation is presented in part (b) of Fig. 4. At

[2] In the following, simulation instants with an upper index w are related to an elementary step in the tokens write process and ones with an upper index r are related to an elementary step of the tokens read process. The same notation is used to denote elementary delays.

[3] For example, the durations of the *polling*, *writing*, *inter-writing*, and *update buffer usage* steps are respectively denoted by t^w_p, t^w_{wr}, t^w_l, and t^w_u.

[4] For example, $x^w_u(k)$ corresponds to the k-th transition instant between the *post-writing* state and the *update buffer usage* state.

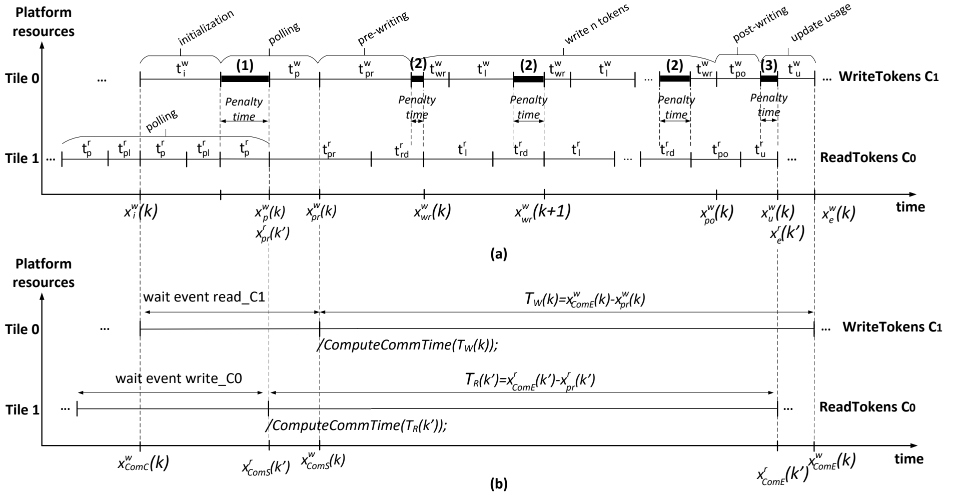

Figure 4: Modeling and simulation of writing n tokens on channel C1 and reading n tokens from channel C0. (a) At transaction level, the elementary delays and penalty delays caused by the contention are exhibited. (b) At the proposed message level, the required communication durations (denoted by T_W and T_R) are computed locally.

message level, the number of simulation events is commonly reduced at the expense of accuracy. To deliver still accurate results, it is needed to appropriately set the durations of the communications, especially in the situations when contentions at shared resources occur. In part (b) of Fig. 4, the successive polling steps are replaced by two synchronization events, denoted by *read_C1* and *write_C0*, that indicate instants when first elementary writing and reading steps start. In the proposed approach, the created simulation model uses instantaneous computation of the whole communication durations. The computation of *ReadTokens* and *WriteTokens* durations is denoted by action `ComputeCommTime`. It is performed at the instants $x_{ComS}^w(k)$ (*i.e.*, $x_{pr}^w(k)$) and $x_{ComS}^r(k')$ (*i.e.*, $x_{pr}^r(k')$) in Fig. 4.

4.3 Definition of the analytical model

The analytical model is built from the knowledge of communication phases and the bus arbitration policy. In our approach, the timed Petri net (TPN) formalism is adopted to formulate the relationships between elementary communication steps. It represents a timed extension of Petri nets for which time is expressed as minimal durations on the sojourn of tokens on places. Fig. 5 represents a part of the TPN description for C_1 *WriteTokens* and C_0 *ReadTokens* through the studied shared communication bus and memory[5].

In this example, a transition q is enabled if each upstream place p contains at least one token. The k-th instant when transition q_i is enabled is denoted by $x_i(k)$. The holding time t in a place is the time a token must spend in the place before contributing to the enabling of the downstream transition. Each holding time corresponds to the duration of the elementary states identified in Fig. 2. In Fig. 5, the place denoted by p_{shared} represents the limited availability of the shared resources and the fact that only some phases of the communication can occur at a time. The previously discussed situations (2) and (3) are exhibited in Fig. 5. Relationships between transition instants can basically be expressed using two operators: addition and maximization [2]. For the situations that cause accesses to communication shared resources (*i.e.*, situations (2), and (3) in Fig. 5) operator maximization is used to express the effect of mutual exclusion on transition instants values. As for example, situation (3) leads to a delay until the end of the step *update buffer usage* of the read process. The instants when the step *update buffer usage* can start is thus expressed as follows:

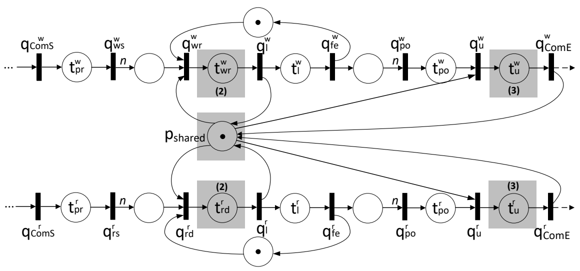

Figure 5: Timed Petri net model of writing n tokens on channel C1 and reading n tokens from channel C0 at transaction level with FCFS arbitration policy.

$$x_u^w(k)=max(x_{po}^w(k)+t_{po}^w(k), x_l^r(k'), x_{ComE}^r(k')),$$

where x_{po}^w is the instant when post-writing starts, t_{po}^w is the duration of the post-writing step, x_l^r is the instant when ongoing elementary read finishes, and x_{ComE}^r is the instant when ongoing tokens read process finishes. Some other transition instants from Fig. 5 are given as examples:

$$x_{ws}^w(k)=x_{ComS}^w(k)+t_{pr}^w(k)$$
$$x_{wr}^w(k)=max(x_{ws}^w(\left\lfloor \frac{k}{n} \right\rfloor), x_{fe}^w(k-1), x_l^r(k'), x_{ComE}^r(k'))$$
$$x_l^w(k)=x_{wr}^w(k)+t_{wr}^w(k), x_{fe}^w(k)=x_l^w(k)+t_l^w(k)$$
$$x_{po}^w(k)=x_{fe}^w(nk), x_{ComE}^w(k)=x_u^w(k)+t_u^w(k)$$

These expressions are used to compute the communication durations in the case of one tokens write and one tokens read for a

[5] In the Petri net notation, circles correspond to places and thick lines to transitions between places. Rectangles in grey emphasize the situations where shared resources are used.

two-tile platform. The adoption of the TPN formalism allows to describe different communication situations with different numbers of tiles and simultaneous read and write processes. It is thus possible to systematize the obtaining of the equations that give the instants when shared resources are accessed. The TPN formalism is suitable in the case of the studied FCFS bus arbitration policy. As it makes possible to express synchronization and mutual exclusion, it could be adopted for different bus protocols and arbitration policies. As an example, this formalism was proposed in [8] to study communication latency for a network on chip infrastructure. In our approach, the content of the `ComputeCommTime` method would be adapted to describe the influence on the communication duration.

5 EXPERIMENTS

5.1 Experiment definition

In this section, two different use-cases and various possible mappings were considered to evaluate the efficiency of the created communication model in terms of accuracy and analysis time. We considered two SDF applications to validate our proposed approach: Sobel filter, and JPEG decoder. The studied Sobel filter application is illustrated in Fig. 1. In Fig. 6, we present (a) the JPEG decoder application and (b) our heterogeneous multiprocessor system that was used for all the experiments. On the upper part of the figure, the platform contains 7 tiles that are connected to a shared memory via a shared interconnect. The measurement infrastructure is presented on the lower part of Fig. 6 (b). The individual tiles used a MicroBlaze soft core that was equipped with private instruction and data memories. Some cores were extended with hardware multipliers (MUL) and floating-point units (FPU). Furthermore one shared memory was used for communication between the tiles. For every considered use-cases and mappings we predicted the average iteration delay of the application. We executed all these experiments on a real platform and measured the actual execution times of each iteration. The obtained predictions were compared between simulation and measured results. We also compared the simulation results from our message-level communication model with the transaction-level model used in [14] as it uses similar assumptions as ours.

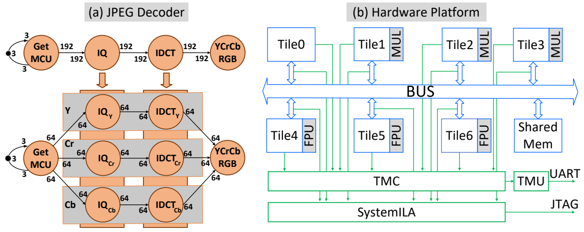

Figure 6: Illustration of (a) JPEG decoder SDF application and (b) heterogeneous platform with 7 tiles connected to a shared memory via a shared interconnect.

Different mappings were applied for each use-case with different number of used tiles. We simulated 3 mappings, denoted by *Sobel1*, *Sobel2* and *Sobel4*, for Sobel Filter respectively with 1, 2 or 4 tiles. For JPEG Decoder, we also considered 3 mappings with 1, 3 or 7 tiles: *Jpeg1*, *Jpeg3* and *Jpeg7*. In the *Jpeg7* experiment, we mapped the actors on 7 tiles to have the highest possible parallelization and to challenge our communication model in terms of accuracy and simulation time. For all the experiments the instruction and local data of an actor were mapped on the private memory of a tile.

5.2 Experiment setup

In our experiment, we considered the Xilinx ZC702 Evaluation Board as the real platform. We used the Xilinx System Integrated Logic Analyzer (SystemILA) to characterize the communication phases. SystemILA connects to the AXI4LITE bus and observes the signals that relate to the communication process. For the writing process, the write address valid signal (*AWVALID*) indicates that the channel is signaling valid write address and control information [10]. The write response valid signal (*BVALID*) indicates that the channel is signaling a valid write response. For the reading process, we observed the signals *ARVALID* and *RVALID*. These signals were sent to the host computer via a JTAG cable. The measurement infrastructure which was used to characterize the computation phase of the actor consists of two main components: *Time Measurement Unit* (TMU) and *Time Measurement Controller* (TMC) which are illustrated in Fig. 6 (b). TMU is basically a counter with the same cycle rate the processors have. The counter can be started and stopped from any tile individually without interference. When the counter got stopped, it sends its counter value via UART to a host computer. Further details about the measurement infrastructure can be found in [13]. In Section 3, we argued that communication and computation do not interfere and they can be considered as independent. To characterize the communication phases we thus implemented applications with no computation parts. Such an implementation corresponds to software that causes the same communication workload as the full application. We implemented different mappings and measured the durations of each elementary steps. One interesting observation is that the elementary delays were constant for all mappings.

For each use-case, the created models have been implemented in the SystemC language. The raw measured computation time of actors were used to validate the proposed communication model. We denote them as *injected data*. During simulation, they get chosen randomly from a text file in each simulated iteration. For the simulation results, we ran 1 000 000 iterations. We considered parallel simulation by splitting our simulations into 20 processes. Each process simulated 50 000 iterations on dedicated processor E5-2630 v4 (2.20 GHz) (at https://ccipl.univ-nantes.fr). For reproduction of our simulations, we provide git repositories with SystemC models[6]

5.3 Results

In this section, we present the results of the experiments considering different criteria: accuracy and analysis time. First we aim to demonstrate the validation of our proposed communication model by comparing the average iteration delay of our message level model with the transaction level model and the measured data. In Tab. 1, we compare the analyzed results of the TL model and ML model with the measured data. Column *Measured* presents the average execution time of 1 000 000 iterations that are the actual measured execution time of the application on real hardware platform. The

[6] Zenodo repository: https://zenodo.org/record/4243071#.X6HY7lB7lhE

Table 1: Comparison of the ML model and the TL model results with the measurement data. The table shows the average execution time (in cycles) and the error.

Experiment	Measured	TL model	ML model
Sobel1	3690	3821.6 (3.57 %)	3756.6 (1.80 %)
Sobel2	2902.5	2960.6 (2.00 %)	2936.6 (1.17 %)
Sobel4	3097.4	2859.7 (−7.70 %)	3153.6 (1.81 %)
Jpeg1	2385860	2387757.6 (0.08 %)	2387489.6 (0.07 %)
Jpeg3	940836	940174.7 (−0.07 %)	941445.3 (0.06 %)
Jpeg7	941059	925969.7 (−1.61 %)	941518.1 (0.05 %)

next two columns show the results of the TL and ML models using the injected data for the computation time representation. We used a noise image as input of the experiments.

In Tab. 1, the results of TL model show over-approximation 3.57 % for *Sobel1* experiment, 2 % for *Sobel2* experiment, while *Sobel4* presents an under-approximation of 7.7 %. The errors in experiments with the JPEG decoder remain lower because this application has a huge computation part in the overall execution of the program. During our experiment, we noticed that the used TL model from [14] did not fully comply to the protocol described in Section 3, especially in the case of multiple contentions. All the experiments of our ML model show a high accuracy result that over-approximate the measured data. In a communication intensive application as Sobel-Filter, the highest error is just 1.81 % in *Sobel4* experiment. In the JPEG decoder application with a huge computation part, the error stays even lower.

In Tab. 2, we first show the duration to measure 1 000 000 iterations of the applications running on the real hardware platform. The simulation time does not include the time it takes to characterize the actors and communication. We then compare the simulation time of our ML model with the TL model. Both models apply the injected data for the computation part. For each experiment, we simulated with 1 000 000 iterations. The column *Speed up* presents the reduction ratio between the two models. In the JPEG Decoder, the simulation time variates from 30 s for *Jpeg1* to 1 h 55 min for the *Jpeg7* experiment. This means that the communication part slowed down the simulation time of the experiments. In our ML model, the simulation time stays less than 10 s for both Sobel-Filter and JPEG Decoder. The simulation time of the ML model was reduced comparing to the TL model simulation. The reduction ratio is 602.25 for the *Jpeg3* experiment and 1731.25 for the *Jpeg7* experiment. *Jpeg7* presented a higher degree of parallel execution but did not show the improvement in the average iteration delay compared to the *Jpeg3* experiment. It demonstrates well the boundedness of the application speed-up due to communication overhead on the shared memory and it demonstrates also the benefit of our approach for design space exploration.

6 CONCLUSION & FUTURE WORK

In this paper, we presented a communication model used for timing prediction of SDF application mapped on a tile-based multiprocessor platform with shared memory. We demonstrated the validation of the message-level simulation approach on the AXI4LITE bus using a FCFS scheduling policy. This simple bus facilitated the characterization of the elementary delays and the communication behaviors to create an analytical model. In the future, we aim at considering more complex bus communication schemes, including DMA transfers and split-burst transactions supported by state-of-the-art COTS MPSoC platforms.

Table 2: Simulation time (HH:MM:SS) with injected data for 1 000 000 iterations.

Experiment	Measured	TL model	ML model	Speed-up
Sobel1	0:07:23	0:00:08	0:00:03	2.67x
Sobel2	0:07:03	0:00:11	0:00:03	3.67x
Sobel4	0:07:13	0:00:13	0:00:02	6.5x
Jpeg1	13:14:31	0:00:30	0:00:06	5x
Jpeg3	5:12:58	0:40:09	0:00:04	602.25x
Jpeg7	5:13:02	1:55:25	0:00:04	1731.25x

ACKNOWLEDGMENT

This work has been partially sponsored by Campus France and the WISE consortium (PETA-MC project under grant agreement 42521PK). It has also been partially sponsored by the DAAD (PETA-MC project under grant agreement 57445418).

REFERENCES

[1] IEEE Standards Association et al. 2012. IEEE Standard for standard SystemC language reference manual. *IEEE Computer Society* (2012).

[2] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. 1992. *Synchronization and linearity, an algebra for discrete event systems.* Wiley & Sons Ltd, New York.

[3] A. Bobrek, J. M. Paul, and D. E. Thomas. 2010. Stochastic Contention Level Simulation for Single-Chip Heterogeneous Multiprocessors. *IEEE Trans. Comput.* 59, 10 (2010), 1402–1418.

[4] S. Chen, C. Chen, and R. Tsay. 2014. An activity-sensitive contention delay model for highly efficient deterministic full-system simulations. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE).* 1–6.

[5] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. P. Stefanov, D. D. Gajski, and J. Teich. 2009. Electronic System-Level Synthesis Methodologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 10 (Oct 2009), 1517–1530. https://doi.org/10.1109/TCAD.2009.2026356

[6] Sébastien Le Nours and Adam Postula. 2018. A Hybrid Simulation Approach for Fast and Accurate Timing Analysis of Multi-Processor Platforms Considering Communication Resources Conflicts. *Journal of Signal Processing Systems* 90, 12 (01 Dec 2018), 1667–1685. https://doi.org/10.1007/s11265-017-1315-x

[7] Edward A Lee and David G Messerschmitt. 1987. Synchronous data flow. *Proc. IEEE* 75, 9 (1987), 1235–1245.

[8] B. Li, J. Zhao, J. Wang, and W. Dou. 2012. A Max-Plus Algebra Approach for Network-on-Chip End-to-End Delay Estimation. In *2012 Eighth International Conference on Semantics, Knowledge and Grids.* 217–220.

[9] P. Lieverse, P. Van Der Wolf, K. Vissers, and E. Deprettere. 2001. A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems. *The Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology* 29 (2001), 197–207.

[10] ARM Limited. [n.d.]. AMBA® AXI™ and ACE™ Protocol Specification. ([n. d.]). AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite.

[11] K. Lu, D. Müller-Gritschneder, and U. Schlichtmann. 2013. Analytical timing estimation for temporally decoupled TLMs considering resource conflicts. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE).* 1161–1166.

[12] G. Schirner and R. Dömer. 2007. Result-oriented modeling - a novel technique for fast and accurate TLM. *IEEE Transactions on computer-aided design of integrated circuits and systems* 26, 9 (September 2007), 1688–1699.

[13] Christof Schlaak, Maher Fakih, and Ralf Stemmer. 2017. Power and Execution Time Measurement Methodology for SDF Applications on FPGA-based MPSoCs. *arXiv preprint arXiv:1701.03709* (2017).

[14] Ralf Stemmer, Hai-Dang Vu, Kim Grüttner, Sebastien Le Nours, Wolfgang Nebel, and Sebastien Pillement. 2019. Experimental Evaluation of Probabilistic Execution-Time Modeling and Analysis Methods for SDF Applications on MPSoCs. In *2019 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS).* Springer, 241–254.