A Simplified Executable Model to Evaluate Latency and Throughput of Networks-on-Chip

Leandro Möller [2]
Luciano Ost[1,2] Leandro Soares Indrusiak[2] Sanna Määttä[3]
Fernando G. Moraes[1] Manfred Glesner[2] Jari Nurmi[3]
{ost, moraes}@inf.pucrs.br {moller, indrusiak, glesner}@mes.tu-darmstadt.de {sanna.maatta, jari.nurmi}@tut.fi

[1] PUCRS - FACIN - Av. Ipiranga 6681- Porto Alegre - 90619-900 – Brazil
[2] FG MES - TECHNISCHE UNIVERSITÄT DARMSTADT- Karlstr. 15, 64283 Darmstadt - Germany
[3] DCS - TAMPERE UNIVERSITY OF TECHNOLOGY - P.O.Box 553, FIN-33101 Tampere - Finland

ABSTRACT
This paper proposes a technique that mixes simulation and an analytical method to evaluate the characteristics of Networks-on-Chips (NoCs). The advantage of this technique is to reduce the simulation time by reducing the complexity of the NoC model while still obtaining accurate results for latency and throughput. The basis of this technique is: (*i*) to send the whole payload data at once in the packet header; (*ii*) to reduce the NoC simulation complexity by omitting the flit by flit payload forwarding; (*iii*) to use an algorithm for controlling the release of the packet trailer in order to close the connection at the right time. For the evaluation of this technique, an actor-oriented model of a NoC, JOSELITO, was created. Simulation results show that JOSELITO is in average 2.3 times faster in 88% of the executed case studies than the implementation without using the proposed technique. The worst case simulation results for latency and throughput have, respectively, 5.26% and 0.1% error compared to the corresponding Register Transfer Level (RTL) model.

Categories and Subject Descriptors
B.7.1 [**Integrated Circuits**]: Types and Design Styles – advanced technologies, VLSI (very large scale integration).

General Terms
Design, Experimentation, Performance, Verification.

Keywords
Networks-on-Chip, Modeling, Performance Evaluation.

1. INTRODUCTION
Multiprocessor System-on-Chip (MPSoC) are a trend in SoC design [1]. These systems are characterized by high data rates that continue to go up in several applications (e.g video processing) running on homogeneous or heterogeneous processors [2]. Due to the high communication demands of these applications, new on-chip communication infrastructures are required.

A network-on-chip (NoC) is a potential and efficient infrastructure to handle MPSoC communication requirements. Scalability, energy efficiency, and support to globally asynchronous locally synchronous (GALS) paradigm justify the adoption of this approach [1][3][4]. However, the adoption of NoCs includes new challenges to the MPSoC design flow, such as choosing a suitable routing algorithm, NoC topology, buffering strategy, flow control scheme, or reducing power consumption.

Due to the vast design space alternatives that these challenges may impose to the final application and its required performance, the evaluation of NoCs become a mandatory step in the MPSoCs design flow. The evaluation of NoCs is required to establish a good compromise between the NoC architecture characteristics and the requirements of the given application. An example of this is to investigate the correlation between the packet length and buffer depth [1].

To accelerate the design space exploration of NoCs, high level abstraction modeling of the NoC is needed. The design exploration at register transfer level (RTL) does not provide the required support to the design space exploration of MPSoCs based on NoC communication architecture. The level of details that have to be modeled and the low accessibility and visibility of the components' behavior justify that affirmative.

The issues mentioned above, combined with time to market pressure, demand high abstraction level modeling and more appropriate debugging capabilities. The high level modeling activity is a trade-off between level of details and model confidence. In NoC context, the *level of details* refers to the structure and behavior abstraction of the NoCs' components. The *structural abstraction* refers to: (*i*) the granularity of data storage (e.g. storage for a flit or packet); (*ii*) the number of components that will be considered or abstracted and how they are interconnected (e.g. the number of wires or a channel). The *behavior* abstraction includes how, and more importantly, when such components (e.g. arbiter) update their internal state and concurrently interact with other components (e.g. buffer). The *model confidence* means how useful the model is for a particular purpose regarding the accuracy results between the model and its reference scenario.

SBCCI'08, September 1–4, 2008, Gramado, Brazil.

In the MPSoC context, software engineers have to deal with concurrency issues inherent to applications that require multiprocessing [5]. Thus, more appropriate debugging capabilities to design MPSoCs based on NoCs are demanded in order to increase and simplify the development of these systems.

In this context, this paper proposes the Payload Abstraction Technique (PAT). The proposed technique can be used to implement simplified NoC models, which allows a performance evaluation by combining simulation and an analytical method. The advantage of this technique is high precision latency and throughput results while reducing simulation time when compared to similar high-level simulation models. In order to validate the proposed technique, a model of a well known NoC architecture, described in [4], was implemented.

This paper is organized as follows. Section 2 describes the related work in NoC performance evaluation. The proposed technique is presented in Section 3. Section 4 presents the reference model as well as the proposed model to validate the Payload Abstraction Technique. Section 5 provides a quantitative and qualitative analysis of the proposed model, including results of the proposed high abstraction modeling approach regarding its fidelity to the original RTL model. Finally, conclusions and future works are pointed out, in Section 6.

2. RELATED WORK

As defined in [1][6], modeling NoC interconnects through abstract models is the first means to approach and understand the required architecture and the impact of the traffic within it. In this context, some research groups try to adapt generic network simulators to the intra-chip environment, while others proposed tools and techniques to specify, simulate and generate NoCs.

Examples of generic network simulators used in this context are the NS-2 and the OPNET. Both include the description of the network topology, communication protocol, routing algorithm, and traffic scheme (e.g. random traffic). These network simulators do not consider some particularities inherent to on chip structures (e.g. communication bandwidth between routers), which can be very important to the design decisions.

Bertozzi et al. [7] propose the NetChip synthesis flow, which allows the exploration of different NoCs topologies. The NetChip flow is composed by three phases: (*i*) NoC topology mapping, (*ii*) selection, and (*iii*) generation (SystemC model that can be simulated at a cycle and signal accurate level). Additionally, an input core graph, obtained with the SUNMAP tool, is used in the mapping phase [8]. Then, the SUNFLOOR tool is used to synthesize the most power and performance efficient NoC topology that satisfies the application requirements [9].

Xu et al. [10] present an architecture-level methodology for modeling, analysing, and designing different NoC architectures. Due to its low level abstraction, this approach can accurately estimate the performance, power, and area of several NoC architectures. This methodology employs some available tools like OPNET, Design Compiler, and SPICE, which requires long simulation times.

Kogel et al. [11] propose a modular framework for system level exploration in Transaction-Level Modeling (TLM) of the on-chip interconnection architecture. It allows capturing performance metrics as latency and throughput of different NoC configurations.

The OCCN framework proposed by [12] enables the creation of NoC architectures at different abstraction levels, protocol refinement, design exploration, and NoC component development and verification based on a communication API. The OCCN methodology has been adopted by Dumitrascu et al. [13] in order to analyze the effectiveness of inter-module communication and adaptation components. In this approach, the communication architecture performance evaluation is based on cycle-accurate co-simulation.

Pestana et al. present in [14] a NoC simulator based on user-generated XML files that describe NoC topology, IP to NoC mapping and detailed interconnections. The simulator also allows describing traffic generators to evaluate NoCs.

Most of proposed techniques or tools allow the emulation of the NoC in different levels of abstraction, considering different architectures (e.g topology, router) and traffic conditions. In many cases, the simulation occurs in TLM style, which demands less design and simulation time than RTL description. However, the accuracy of those high level models is influenced by the structural and behavioral abstractions. The major contribution of the proposed work is to present a technique that can be applied to wormhole packet switching NoCs to reduce the simulation time, with high accuracy of latency and throughput estimation.

3. PROPOSED TECHNIQUE

The main idea of the proposed technique is to decrease the simulation time by reducing the number of communication events. The communication events are reduced due to the abstraction of the packet payload. The PAT comprises: (*i*) packet abstraction; (*ii*) buffer abstraction, and (*iii*) an analytical method to estimate the packet trailer release time. Usually packets are composed by a header, a payload, and a trailer. In this technique, the packet structure is abstracted in *header* and *trailer*, as shown in the right bottom part of Figure 1. This abstraction eliminates the *flit by flit* payload transfer.

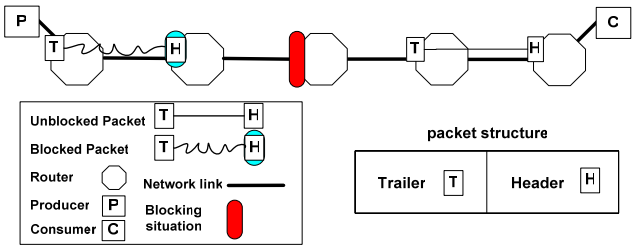

Figure 1 - Packet transmission situations (unblocked and blocked), and packet structure.

In this technique, the NoC routers are modeled using the entities buffer and control. The entity *control* is responsible for input port arbitration, routing algorithm, and input to output port data forwarding. The entity *buffer*, a FIFO structure, is modeled as a finite state machine (FSM), with four states: (*i*) *empty*, (*ii*) *header*, (*iii*) *waiting trailer*, and (*iv*) *trailer*.

In the *empty state*, the buffer is able to receive a packet header. The received header is stored into the buffer in the *header state*. After forwarding the packet header, the buffer goes to the *waiting trailer* state. When the trailer is received, the FSM goes to *trailer state*. Once the trailer is forwarded and removed from the buffer, it returns to the *empty state*.

To ensure correct functionality during packet transfers and to obtain high precision latency and throughput results, a sender (producer or router) releases the packet trailer according to an analytical method. One possible solution is presented in Equation 1, which indicates the packet trailer release time (*ptrt*) including when the header is forwarded (*hft*), the packet size and the number of cycles to transmit one flit (*ctf*) from one hop to another router or consumer. The *ctf* is expected to be one for credit-based and two for handshake control flow (clock cycles).

$$ptrt = hft + pcksize * ctf \quad (1)$$

where: ptrt: *packet trailer release time*
hft: *header forwarding time*
pcksize: *packet size (number of flits)*
ctf: *number of clock cycles to transmit one flit*

The header forwarding time depends on the number of clock cycles required to execute the arbitration, the routing algorithm and the successful reception of the header by the neighbor resource (router or consumer). This parameter is obtained from the RTL-NoC simulation. After reaching the packet trailer release time, defined by Equation 1, the packet trailer is sent, following the same path reserved by the header.

When a header packet arrives in an input buffer, two blocking situations can occur: (*i*) the desired output port is reserved by the control entity to another input port; (*ii)* the target neighbor input buffer is in the trailer state. The packet trailer can be blocked only when the target buffer is in the header state. In this case, after sending the header, the packet trailer is forwarded after *ptrt* clock cycles. The connection between an input and an output port of the router will be closed right after sending the packet trailer to the neighbor input buffer.

According the proposed technique, three transmission scenarios are possible. For the sake of simplicity, the following explanation assumes that the abstracted packet payload requires 4 input buffers (4 hops, including the header position, Figure 2) in the path consumer-producer, i.e., the relationship between the real payload size and the real buffer depth is 4. This means that when the packet header arrives at the consumer, the trailer must be stored into the input buffer of the router located 4 hops before its consumer location. In the following examples, consider arbitration/routing requiring 7 clock cycles, payload size equal to 21 flits and credit-based flow control (*ctf*=1). The packet header (H) arrives at the first router (R1) at time 0.

Scenario (i): ***Blocking-free delivery***
After sending the header, the producer (P) forwards the trailer after 21 cycles (Equation 1). This is the best case scenario, without any resource conflicts, resulting in a blocking-free delivery. In this case, there is no loss of accuracy in the latency evaluation due to: (*i*) the trailer arrives at the consumer after 21 cycles after the header received time; (*ii*) the connection between P and R1 will be closed after 21 cycles of the header forwarded time (*hft* equal 7). After sending the header to the next router (R2) at cycle 7 (*hft*), the Equation 1 is applied (R1) and the trailer is forwarded to the next router (R2) at the cycle 28 (Figure 2).

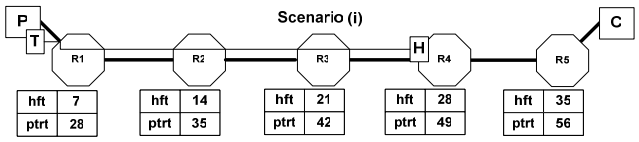

Figure 2 - Estimated release times regarding blocking-free delivery scenario.

According to [4], in a blocking-free delivery scenario, the latency of a packet (*pcklantency*) from producer (P) to consumer (C) is obtained from Equation 2.

$$pcklantency = nhops * arbt + pcksize \quad (2)$$

where: nhops: *number of hops between P and C*
arbt: *arbitration time*
pcksize: *packet size (number of flits)*

Applying Equation 2 using the parameters of the scenario (*i*), the same 56 clock cycles are obtained.

Scenario (*ii*): ***Header Blocking***
The header is sent by the producer, but in the fifth router (R5) a blocking situation is detected, as illustrated in Figure 3 . During the header blocking period (assuming 7 clock cycles), the trailer is forwarded two hops further (P to R2), decreasing the number of hops between the header and trailer (2 instead of 4 hops). Consequently, the connection between P and the first router (R1) is closed without considering the impact of the header blocking, which can lead to loss of accuracy on blocking other packets.

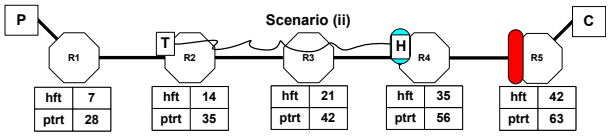

Figure 3 - Packet forwarding situation regarding header blocking.

Scenario (iii): ***Header and Trailer Blocking***
The header and the trailer are sent as described in the previous scenarios. Due to the high blocking time (assuming 14 clock cycles) the trailer (R3) is blocked by its packet header (R4), as shown in Figure 4. When the trailer is blocked by the header, the Equation 1 is applied again and the router R4 will releases the packet trailer after *ptrt* clock cycles of the header forwarded time (*hft* equal 42). In this case, the distance between the header and the trailer is just one hop. Thus, two connections (P to R1 and R1 to R2) are released without considering the impact of the header blocking, increasing the possibility of loss of accuracy on blocking other packets.

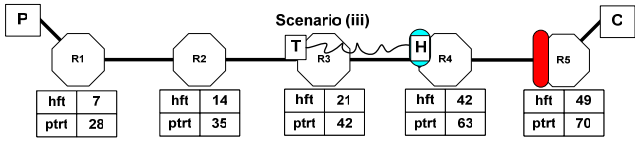

Figure 4 - Packet forwarding situation regarding header and trailer blocking.

It should be clear that the proposed technique is flexible in terms of modification and it is not restricted to the abstractions presented here. For example, different parameters can be added in the Equation 1, improving the quality of the technique.

4. MODEL

4.1 Reference Models

Two models are used as reference in the current work: a RTL-VHDL (HERMES) and a high abstraction model (RENATO). The first is used as reference for latency and throughput performance figures, since it is cycle accurate; the second is used as reference for simulation time, to enable the comparison between two models at a similar abstraction level. It is important to mention that the high-level model proposed in the current work and RENATO are back annotated with timing information from HERMES, thus allowing latency comparison among them.

HERMES is a parameterizable infrastructure specified in VHDL at RT level, aiming to implement low area overhead packet switching NoCs. In such interconnects, all data is partitioned in packets, which are composed by a header and a number of flits (data words). Packets are sent from a processing element to a given destination through routers that have some awareness of the interconnect topology and thus manage to deliver the packet while avoiding deadlocks and livelocks.

HERMES supports either 2D mesh or torus topologies and allows designers to select routing algorithm, control flow mechanism, flit size and buffer depths. Its routers have centralized switching control logic and five bi-directional ports. One port is used to establish the communication between a router and its local processing element, while the others are connected to the neighbor routers. Each input port stores received data on a FIFO buffer. The routing algorithm chosen for this work is the XY, which sends packets in the X direction up to the target X coordinate, and next proceeds in the Y direction until reaching the target router. Multiple packets may arrive simultaneously in a given router. A centralized round-robin arbitration grants access to incoming packets. The priority of an input port is a function of the last input port having a routing request granted. If the incoming packet request is granted by the arbiter, the XY routing algorithm is executed to connect the input port to the correct output port. If the algorithm returns a busy output port, the header flit and all subsequent flits of this packet are blocked. After all flits in a packet are transmitted, the port is released.

Due to the low observability and debugging capabilities provided by HERMES, RENATO [15] was created as a first attempt to increase the design space required for future systems. RENATO is an actor oriented model implemented on Ptolemy II [16]. RENATO allows a designer to change buffer schemes, routing and arbitration algorithms, and even the network topology, then simulate the model in Ptolemy II to obtain performance figures as area, latency, throughput, and power consumption.

4.2 Proposed Model

This Section presents JOSELITO, a NoC model that uses the technique described in Section 3 to simplify the simulation complexity. Three requirements are set to this model: to simulate faster than RENATO; to allow debugging NoC resources and data passing through it; and to provide latency and throughput results as similar as possible to HERMES.

In order to implement JOSELITO to use PAT, the following components of the RENATO were modified: routers, buffers, and packet structure. Routers do not forward data flit by flit since they use the PAT. Buffers have only one position, which can be in one of the four states explained in Section 3. The packet structure of JOSELITO is composed by header and trailer. As JOSELITO is a high level abstraction model of a NoC, some degree of freedom can be applied to it to achieve higher debugging and observing capabilities, e.g. the amount of information added to headers and trailers. JOSELITO's header contains the target router address, the packet size, and the data payload. JOSELITO's trailer contains only evaluation parameters.

5. RESULTS

This Section presents simulation results of the implemented model. The NoC Models (HERMES, RENATO and JOSELITO) are evaluated varying:

- NoC sizes: 2x2, 3x3, and 4x4;
- traffic distribution: uniform (200 Mbps), normal (minimal rate 150Mbps, maximal rate 250Mbps, and standard deviation 10Mbps), and Pareto on-off (200 Mbps, maximum number of bursts set to 10 packets);
- number of transmitted packets per producer: 100 packets (T1), 1000 packets (T2), 10000 packets (T3), and 20000 packets (T4).
- Packet size: 16 and 50 flits.

Figure 5 presents the average latency simulation error (measured in clock cycles) of JOSELITO in comparison to HERMES for 3 different traffic distributions, 3 different NoC sizes and 16 flits per packet. The closer the lines are from the zero latency error, more similar JOSELITO and HERMES latency are obtained. The worst case average latency error presented is 3.4 clock cycles (Figure 5 (c) - 4x4 - 20000 packets). In this specific worst case, JOSELITO average latency is 5.26% lower than the reference model (according to the average absolute latency results presented in Table 1, 64.59 and 61.19 clock cycles for HERMES and JOSELITO, respectively).

Figure 6 presents the JOSELITO latency error in comparison with HERMES when applying packets with 50 flits inside the NoC model. As the model does not yet consider the buffer size and it is calibrated to 16 flits per packet, the worst case error presented by 50 flits is 4.26 clock cycles (Figure 6 (b), 100 packets) in average. In this specific worst case, the absolute average latency to deliver all packets is 164.04 clock cycles in JOSELITO and 168.30 in HERMES. This represents only 2.53% latency error in comparison with the same case study in the reference RTL-VHDL model. It is important to remember that the packet size is usually chosen by network interfaces, which is outside the NoC model and can usually be calibrated to any possible size.

Table 1 presents the absolute average latency and throughput simulation results for HERMES and JOSELITO using 16-flit packets. The worst case throughput error of JOSELITO in comparison to HERMES is 0.1% when sending 10000 packets per producer in a 4x4 Pareto on-off traffic distribution, which reflects multimedia applications behavior. These similar throughput and latency results between both models show that a high-level abstraction model can indeed replicate the behavior of a cycle accurate model.

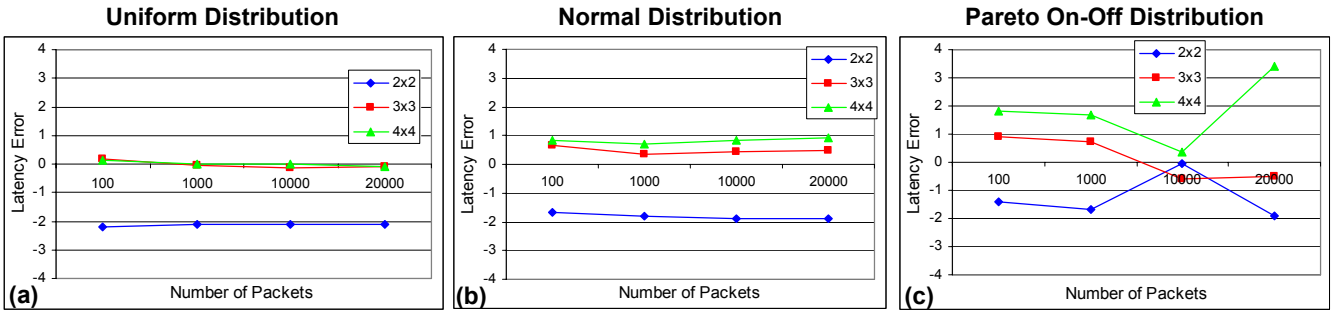

Figure 5 - Latency error between JOSELITO and HERMES (the RTL reference model) for 3 different traffic distributions (uniform, normal, and pareto on-off) and NoC sizes (2x2, 3x3 and 4x4). 16 flits packets.

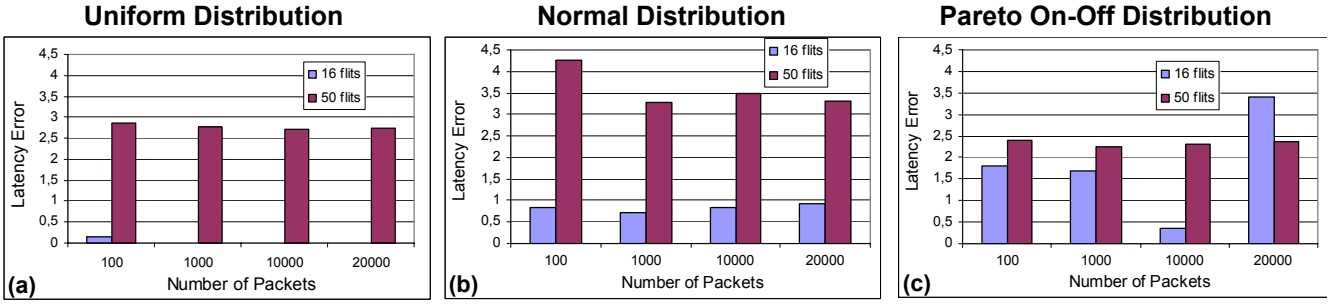

Figure 6 - Latency error between a 4x4 JOSELITO and a 4x4 HERMES for 3 different traffic distributions (uniform, normal and pareto on-off) and 2 different packet sizes (16 and 50 flits).

Table 1 - Average latency ("L" - clock cycles) and throughput ("T" - % of the relative channel bandwidth) for HERMES and JOSELITO, using 16 flits packets.

		Uniform Distribution						Normal Distribution						Pareto On-Off Distribution					
		2x2		3x3		4x4		2x2		3x3		4x4		2x2		3x3		4x4	
		HERMES	JOSELITO	HERMES	JOSELITO	HERMES	JOSELITO	HERMES	JOSELITO	HERMES	JOSELITO	HERMES	JOSELITO	HERMES	JOSELITO	HERMES	JOSELITO	HERMES	JOSELITO
T1	L	58.81	60.99	70.99	70.83	87.68	87.54	52.44	54.12	63.32	62.64	73.96	73.14	49.46	50.86	55.35	54.44	62.62	60.82
	T	12.88	12.86	7.21	7.17	4.37	4.34	13.95	13.99	7.28	7.34	4.37	4.38	12.04	12.03	5.30	5.31	3.16	3.16
T2	L	58.86	60.98	70.75	70.81	85.97	85.96	51.28	53.07	61.85	61.50	73.85	73.14	49.63	51.32	55.96	55.24	62.49	60.80
	T	13.76	13.77	7.29	7.28	4.55	4.57	13.41	13.42	7.13	7.13	4.44	4.45	13.25	13.27	6.80	6.81	4.32	4.32
T3	L	58.93	61.05	71.09	71.22	85.08	85.07	51.65	53.56	61.10	60.66	73.31	72.47	50.66	50.69	54.71	55.30	61.59	61.22
	T	13.69	13.69	7.20	7.20	4.53	4.53	13.4	13.40	7.06	7.06	4.42	4.42	13.39	13.44	7.12	7.11	4.50	4.40
T4	L	58.87	61.00	71.22	71.30	84.93	85.00	51.99	53.88	61.09	60.62	73.33	72.42	48.71	50.60	54.92	55.44	**64.59**	**61.19**
	T	13.69	13.69	7.25	7.25	4.51	4.52	13.47	13.48	7.09	7.09	4.45	4.46	13.53	13.45	7.02	7.08	4.41	4.41

Table 2 shows that JOSELITO is in average 2.3 times faster than RENATO in 88% of the executed case studies. These improvements in simulation time were achieved only by reducing the number of communication events originally caused by the flit by flit forwarding.

Table 2 - Speed up of JOSELITO in comparison to RENATO.

	Uniform			Normal			Pareto On-Off		
	2x2	3x3	4x4	2x2	3x3	4x4	2x2	3x3	4x4
T1	2.51	2.41	2.93	5.95	2.00	2.08	2.45	2.34	2.34
T2	2.52	2.09	2.27	2.81	2.01	2.01	2.54	1.52	1.64
T3	2.53	1.78	1.99	1.82	1.39	1.67	1.78	0.72	0.74
T4	4.07	1.38	1.68	1.70	1.14	1.67	4.71	0.60	0.70

6. CONCLUSIONS AND FUTURE WORKS

The NoC performance evaluation is a mandatory step to obtain the best trade-off between communication architecture and application requirements. This performance evaluation is a complex and increasingly relevant problem in MPSoCs design due the heterogeneity of the applications. Thus, it is important to dispose models which allow the simulation of the NoC in different levels of abstraction, considering different architectures and traffic conditions.

This paper presented a technique based on payload abstraction, permitting to evaluate with high precision the latency and the throughput of wormhole packet switching NoCs. A model implementation applying the proposed technique was evaluated under different traffic conditions. Results show that the proposed model is in average 2.3 times faster than the reference

model (88% of the executed case studies performance). Additionally, the proposed approach was compared to a cycle-accurate simulation (RTL-VHDL), presenting worst case simulation latency and throughput error results of 5.26% and 0.1%.

The high level model presented in the current work can still be optimized to reduce the number of communication events, thus reducing the simulation time. One possibility is to modify the model to not to generate negative acks, but sending an ack just when a resource is available. The arbiter can also be modified in order to reduce the number of times the round robin algorithm is called when it is in idle state. Future work includes exploring the use of other analytical methods to improve the accuracy of the release time computation.

7. ACKNOWLEDGMENTS

This research was supported partially by CNPq (Brazilian Research Agency), projects 142049/2008-5, 300774/2006-0 and 290121/2006-0.

8. REFERENCES

[1] Pande P.P. et al. **Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures**. IEEE Computer, vol. 54(**8**), 2005.

[2] Wolf, W. **The Future of Multiprocessor Systems-on-Chips**. In: Design Automation and Test in Europe (DATE'04), 2004.

[3] Benini L. et al. **Networks on chips: a new SoC paradigm**. IEEE Computer, 35(**1**). 2002, pp. 70-78.

[4] Moraes, F. et al. **Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip**. Integration, the VLSI Journal, v.38(**1**), 2004, pp. 69-93.

[5] Martin, G. **Overview of the MPSoC Design Challenge**. In: Design Automation and (DAC'06), 2006.

[6] Bjerregaard, T.; Mahadevan, S. **A Survey of Research and Practices of Network-on-Chip**. ACM Computing Surveys 38(1), 2006.

[7] Bertozzi, D.; Jalabert, A.; Srinivasan, M.; Tamhankar, R.; Stergiou, S.; Benini, L.; De Micheli, G. **NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems on Chip**. IEEE Transactions on Parallel and Distributed Systems, 16(**2**), 2005.

[8] Murali, G. De Micheli. SUNMAP: **A Tool for Automatic Topology Selection and Generation for NoCs**. In: DAC, 2004.

[9] Meloni, P.; Murali, S.; Carta, S.; Camplani, M.; Raffo, L. De Micheli, G. **Routing Aware Switch Hardware Customization for Networks on Chips**. In: NanoNet, 2006.

[10] Xu, J.; Wolf, W.; Henkel, J.; Chakradhar, S. **A Methodology for Design, Modeling, and Analysis of Network on Chip**. In: ISCAS, 2005.

[11] Kogel, T.; Doerper, M.; Wieferink, A.; Leupers, R.; Ascheid, G.; Meyr, H.; Goossens, S. **A Modular Simulation Framework for Architectural Exploration of On-Chip Interconnection Networks**. In: CODES/ISSS, 2003.

[12] Coppola, M.; Curaba, S.; Grammatikakis, M.; Maruccia, G.; Papariello, F. **OCCN: A Network on Chip Modeling and Simulation Framework**. In: DATE, 2004.

[13] Dumitrascu, F.; Bacivarov, I.; Pieralisi, L.; Bonaciu, M.; Jerraya, **A. Flexible MPSoC Platform with Fast Interconnect Exploration for Optimal System Performance for a Specific Application**. In: DATE, 2006.

[14] Pestana, S.; Rijpkema, E.; Radulescu, A.; Goossens, K.; Gangwal, O. **Cost-Performance Trade-Offs in Networks on Chip: A Simulation-Based Approach**. In: DATE, 2004.

[15] Indrusiak L.S., Ost, L., M¨oller L., Moraes, F., and Glesner M., **Applying UML interactions and actor-oriented simulation to the design space exploration of network-on-chip interconnects.** In: VLSI, 2008.

[16] Liu, J; Eker, J; Janneck, J W.; Liu, X J. and Lee, E A. **Actor-oriented control system design: A responsible framework perspective**. IEEE Transactions on Control Systems Technology, 12(**2**), 2004.