An Analytical Method for Evaluating Network-on-Chip Performance

Sahar Foroutan[1,2], Yvain Thonnart[2], Richard Hersemeule[1], Ahmed Jerraya[2]
[1] ST-Microelectronics, [2] CEA-Leti
{sahar.foroutan, yvain.thonnart, ahmed.jerraya}@cea.fr
richard.hersemeule@st.com

Abstract

Today, due to the increasing demand for more and more complex applications in the consumer electronic market segment, Systems-on-Chip consist of many processing elements and become larger and larger. While on-chip system designers must be able to get fast and accurate communication performance analysis for such huge systems, the simulation-based approaches are not adequate anymore. Addressing the increasing need for early performance evaluation in NoC-based system design flow, this paper presents a generic analytical method to estimate communication latencies and link-buffer utilizations for a given NoC architecture with a given application mapped on it. The accuracy of our method is experimentally compared with the results obtained from Cycle-Accurate SystemC simulations.

I. INTRODUCTION

The broad interest in Network-on-Chip (NoC) technology has led to the definition of many NoC architectures, implementation strategies, and network performance evaluation methods [1-4]. Due to growing SoC complexity, NoC communication parallelism and the need to tight time-to-market design flows, traditional simulation-based methods don't fulfill anymore all requirements of performance evaluation domains: they are extremely slow, non-exhaustive, non-scalable with growing system size and also they are achieved relatively late in design flow. Mathematical and analytical methods seem to be reliable alternatives for performance evaluation purposes before implementation and very soon in design process.

This paper presents a performance evaluation method based on a numerical analysis approach providing the average buffer utilizations, the mean packet latency and the router port-to-port latency of a given NoC architecture. The average buffer utilization gives helpful insights into optimal link capacity allocation and the distribution of the traffic over network buffers. The packet latency is an essential metric in NoC-based system performance and helps to estimate the application runtime. Expressed as a function of offered-load, packet latency can demonstrate the maximum acceptable throughput (saturation threshold) of a network.

NoC performance evaluation is traditionally based on simulation [5-7]. Nevertheless, analytical techniques using queuing theory have also been proposed to measure the communication latency of NoCs and parallel computer networks (off-chip communication) [8-14]. Some of them target only a particular network topology such as [12, 13] which are restricted to k-ary n-cubes or hypercube network topologies. Some others place limitation on buffer or packet size, like [10] which addresses wormhole routing with exponentially distributed message length, or [14] proposing an analytical model for wormhole routing delay but restricted to networks with single flit buffer capacity. In [14] the end-to-end transfer delay is approximated for packets larger than the buffers along their path. Link acquisition time is ignored by assuming that there are as many virtual channels as the number of flows sharing the same physical link (so every head flit can acquire a VC instantaneously on every link it traverses). Using queuing models, the method approximates the link transmission time by considering the time attributed to other virtual channels (i.e. other flows) that interleave over the same physical link. Actually, the authors analyze a "quasi-"circuit-switching case while our proposed method investigates packet-switching cases dominated by both port acquisition and link transfer delays. Authors of [11] present a more general queuing theory based model addressing wormhole routing with arbitrary size messages and finite buffers under application-specific traffic patterns also supporting arbitrary network topology and deterministic routing. While the framework of our approach relies on the same assumptions, our methodology is different from theirs: our approach is based on numerical analysis and iterative computation, whereas [11] relies on the generalization of the single queue model to multiple queues in the router. Using that approach, however, the backpressure effect due to downstream contention tends to be minimized, and thus the saturation effect arises later than what can be observed in simulations.

The method presented in this paper is generic with respect to the implementation in the sense that it supports arbitrary network topologies and deadlock-free routing algorithms with arbitrary packet and buffer lengths. Regarding the applications, we target known traffics with any repartition on the NoC as long as it can be approximated with Poisson distributions (modeled by mean values). The paper is structured as follows: In section II we discuss the basic concepts and assumptions. Section III explains our analytical method in detail. Section IV includes experimental results and the comparison with the simulation results. Finally section V concludes the paper.

II. PACKET LATENCY

The definition of NoC latency varies from one work to another. It can be measured per data word, header, packet, or message transfer (several packets) while including or excluding the time at the source queue [15].

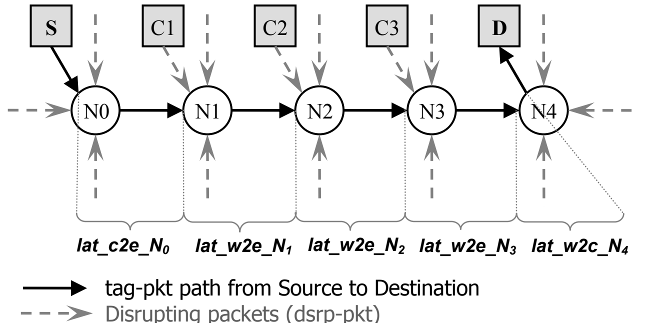
S
C1
C2
C3
D
N0
N1
N2
N3
N4
lat_c2e_N0
lat_w2e_N1
lat_w2e_N2
lat_w2e_N3
lat_w2c_N4
tag-pkt path from Source to Destination
Disrupting packets (dsrp-pkt)

Fig.1: At each node of the path disrupting packets appear probabilistically in front of tagged packet.

The term *packet latency* is interpreted in this paper as the mean latency of a *tagged packet* traversing the NoC from a given source to a given destination. Packets are considered atomic (i.e. they cannot be divided into shorter pieces by the network) so when the header of a packet arrives to a port the packet body arrives immediately after it. In fig.1 the black arrows represent the tagged packet's path from its source to its destination, while dashed arrows demonstrate "*disrupting packets*" competing with the tagged packet for the same output at each node. The proposed method computes separately the mean latency of each node of the path by considering the probabilities and delays of contention between the tagged packet and all other disrupting packets arriving to that node. Node mean latencies are then summed up to give the mean latency of the entire path. For computing the node mean latency we propose an iterative approach that enables us to consider the reciprocal impact of all incoming flows to that node, on each other. Since the latency of a node depends on the latency of its following node and so on, the order for computing node-latencies is important and must follow the reverse order of dependencies. We use a recursive algorithm that regarding the NoC topology, routing algorithm and traffic pattern finds all latency dependencies and returns the node latencies and average buffer utilizations according to the proper computation order.

III. Node Latency

The term "***node latency***" is used in this paper as the mean latency that the tagged packet header needs to cross a node and the buffer after the node. It is labeled with "*lat-in2out-N_p*" when the tagged packet arrives from *"in"* port and targets *"out"* ports of node N_p, and includes two following components (fig.2):

1. **The port-acquisition delay (*dly_port_acq*):** is the delay needed for the header flit to acquire its output port and is counted from the moment the packet header appears at the head of the input buffer of N_p until the moment its output port is allocated and so it can be written in the input buffer of the next node (N_{p+1}).
2. **The link transfer delay *(dly_link_xfr)*:** The time duration a packet header takes for being transferred through the buffer of link $N_p \rightarrow N_{p+1}$ and is counted from the moment the packet header is written to the buffer until the moment it arrives to the head of that buffer (at input port of N_{p+1}).

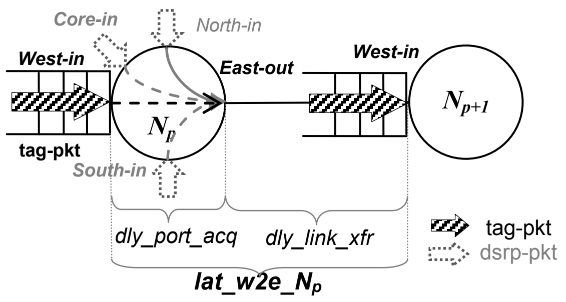

Fig.2: *lat-w2e-N_p = dly_port_acq + dly_link_xfr*

A. Port Acquisition Delay

Port acquisition delay includes the router service time for the packet header (*dly_hdr_serv)* and also the delay caused by contentions with other disrupting packets competing for the same output port (*dly_hdr_cont*):

$$dly\_port\_acq(io) = dly\_hdr\_cont(io) + dly\_hdr\_serv \quad \textbf{(Eq.1)}$$

Router service time is a function of router architecture and considered as an input parameter for the method. The header contention delay (*dly_hdr_cont)* depends on the probabilities of contention (*pr_cont*) between the tagged packet and all other competing inputs and also the delay each disrupting packet causes until it releases the shared output port (*dly_cont*). We assume that arbitration mechanism is starvation-free and fair in the sense that it is symmetric in terms of probability of selection of simultaneous packets on the inputs. Also we assume that at most one packet from each demanding input is allowed to pass after arbitration (never two consecutive packets from the same input are routed at a single allocation when there is other demanding inputs). With this assumption we have:

$$dly\_hdr\_cont(io) = \sum_{j \in I,\, j \neq i} pr\_cont_O(ij).dly\_cont_O(ij) \quad \textbf{(Eq.2)}$$

Where I is the set of all input ports competing for the output port $\boldsymbol{o}$, assuming the tagged-packet is present at input port $\boldsymbol{i}$.

1) Contention Delay

Contention delay *$dly\_cont_o(ij)$* is the mean delay a disrupting packet coming from input port ***j*** produces in front of the tagged packet from input port ***i***, before releasing the shared output port ***o***. The tagged packet may arrive on anytime of the presence of a disrupting packet: it may arrive when the disrupting packet is stalled by the rest of previous flits accumulated in buffer (*impact-buf(o)*) or when its header is waiting on router input port ***j*** to be served (*dly_hdr_cont(jo))* or finally the tagged packet may arrive uniformly on each body-flit (of a disrupting packet) when it is crossing the router in a pipeline way (mean packet length ***L***). Regarding the probability of contention with the header or body of a disrupting packet, *$dly\_cont_o(ij)$* is obtained from equation 3:

$$dly\_cont_O(ij) = F(dly\_hdr\_cont(jo)) = A * B \quad \textbf{(Eq.3)}$$

$$A = \frac{L + impact\_buf(o)}{L + impact\_buf(o) + dly\_hdr\_cont(jo)}$$

$$B = (\frac{L + impact\_buf(o) + 1}{2} + dly\_hdr\_cont(jo))$$

Where *dly_hdr_cont(jo)*is the delay met by the header of disrupting packet in contention with other packets, which are similarly considered as disrupting packets for the first one. The *"impact-buf"* is the back pressure impact which arises when the contention in the network is very high. When the number of flits accumulated in the buffer exceeds the buffer size, packets have to wait until this extra flits, retro-propagated in preceding input buffers, be transferred. In section *III.C.2* we explain how we compute the average link-buffer utilization. It depends on the latency of the following node which is supposed to be computed in advance regarding the computation order.

$$impact\_buf = \begin{cases} 0 & Avr\_used\_buf \leq (buf\_size - 1) \\ Avr\_used\_buf - (buf\_size - 1) & \text{otherwise} \end{cases}$$

As expressed in equations 2 and 3 there is a reciprocal dependency between header contention delays of different packets competing for output ***o***. E.g. *dly_hdr_cont(io)* in (Eq.2) is a function of *$dly\_cont_o(ij)$* which itself is a function of *dly_hdr_cont(jo)*and vice versa. We propose an iterative computation (*III.B*) that for a given node determines latencies between all competing inputs and the associated output port. The iteration starts with *dly_hdr_cont(jo)=0* in Eq.3 for all ***j***.

2) Contention Probability

At a given router N the probability of contention between the tagged packet coming from input ***i*** and a disrupting packet coming from ***j*** for obtaining output ***o*** is called ***$pr\_cont_o(ij)$*** and in a general way we have:

$$pr\_cont_o(ij) = P_{io}P_{jo}$$

P_{jo} depicts the probability of the presence of data on link ***j*** addressing link ***o***. In our method we assume that the tagged packet is already present, so $P_{io} = 1$ and therefore the contention probability is equal to P_{jo} ($pr\_cont_o(ij)=p_{jo}$). In quasi zero loads when the probability of contention is very weak, the forwarding rate λ_{jo} (the data rate from input link j to output link o of the router) is a good approximation for $pr\_cont_o(ij)$. But in practice when the load increases, all delays imposed to the header of a disrupting packet (i.e. delay header contention and impact buffer) result in increasing the probability of contention and must be taken into account: **(Eq.4)**

$$pr\_cont_o(ij) = G(dly\_hdr\_cont(jo)) =$$
$$\lambda_{jo}\frac{L + impact\_buf + dly\_hdr\_cont(jo)}{L}$$

Where λ_{jo} is the forwarding rate and is determined from distribution traffic matrix (T) and routing algorithm. Similar to $dly\_cont_o(ij)$, $pr\_cont_o(ij)$ is a function of $dly_hdr_cont(jo)$and obtained from iterations. In the same way the first iteration begins with $dly\_hdr\_cont(jo)=0$ in Eq.4 for all j.

B. Iterating Refinement

To compute and refine the reciprocal impact of different flows to each other we perform iterations on all inputs competing for the same output of a router. For each iteration one of the competing inputs is marked as the tagged packet (i_{Tag}) and others are marked as disrupting packet inputs. For example in fig.3 four inputs i_1, i_2, i_3 and i_4 of router N_p compete for the output o. Each iteration aims to find the header contention delay between input port i_{Tag} and output port o ($dly\_hdr\_cont(i_{Tag}o)$). The probability and delay of contention with disrupting input i ($pr\_cont_o(i_{tag}i)$, $pr\_cont_o(i_{tag}i)$) are computed according to equations 4 and 3 respectively in which the $dly\_hdr\_cont(io)$ is obtained from the previous iteration. The first iteration starts with $dly\_hdr\_cont(io)=0$ and returns the header contention delay of i_{Tag} ($dly\_hdr\_cont(i_1o)$) which is used in the second iteration for finding $dly\_hdr\_cont(i_2o)$. We note that i_1 is the tagged packet port in iteration 1 while it is considered as a disrupting packet in iteration 2. More we progress in iterations more the result becomes accurate. Clearly iterations make a loop (for example in the fifth iteration i_1 is again considered as i_{Tag}). As soon as two successive iterations for the same i_{Tag} is less than a constant ε (determined by the desired accuracy) we can stop the iteration for that input. The iteration approach provides the latencies of reaching a given output port from all possible input ports of a given node.

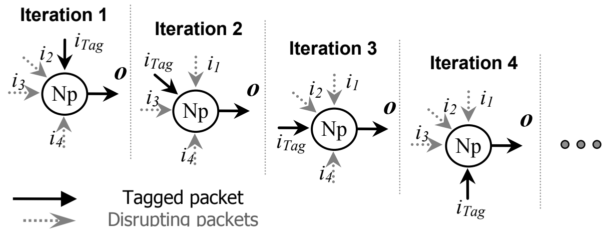

Fig.3 Iterating Latency calculation for different inputs (i_1, i_2, i_3, i_4) competing for obtaining the same output (o)

C. Link Transfer Delay

The link transfer delay (dly_link_xfr) is the time duration a packet header takes to be transferred through the buffer of link $N_p \rightarrow N_{p+1}$ after acquisition of the output port at N_P. This delay includes the buffer latency (dly_buf_const) which is a constant delay required for traversing the buffer even when it is empty and depends on buffer architecture, and the pipeline delay ($dly\_buf\_pip_o$) caused by flits previously accumulated in the buffer: $dly\_link\_xfr_o = dly\_buf\_const + dly\_buf\_pip_o(i)$

In fact $dly\_buf\_pip_o(i)$ is the average number of flits accumulated in front of the packet from input i, immediately when its output port o is allocated:

$$dly\_buf\_pip_o(i) =$$
$$\sum_{j\in I, j\neq i} pr\_cont_o(ij)\cdot dly\_buf_o(j) + \sum_{j\in I} pr\_buf_o(j)\cdot\left(\frac{dly\_buf_o(j)}{2}\right)$$

Where $dly\_buf_o(j)$ is the buffer space occupied after the transmission of a disrupting packet coming from input j in contention with i. The first sigma covers the cases in which the tagged packet is in contention with a disrupting packet and therefore meets all the buffer space occupied by that disrupting packet $(dly\_buf_o(j))$ while the second sigma covers the cases in which it arrives just after a transmission (no contention for the obtaining of the output port) and in this case the tagged packet meets the mean buffer space taken by previous transmissions $(dly\_buf_o(j)/2)$ which could be even from its own input.

$dly\_buf_o(j)$ is equal to the pipelined delay $dly\_buf\_pip_o(j)$ plus the time the header of that disrupting packet waits before being routed at N_{P+1}($stop\_next\_node$). Similar to dly_hdr_cont for computing $dly\_buf\_pip_o$, we consider the reciprocal impact of all incoming flows addressing output o of N_p, on each other by performing iterations.

$$dly\_buf_o(j) = dly\_buf\_pip_o(j) + stop\_next\_node$$
$$pr\_buf_o(j) = \lambda_{jo}\frac{dly\_buf_o(j)}{L}$$

1) Stop at Next-Node and recursive computation

Clearly while the header is waiting at input port of N_{P+1} the body flits are accumulated behind it and since the packet header can be routed to different output links of N_{p+1}, *stop-next-node* is equal to the average latency needed for acquiring any of these outputs. If O be the set of all possible outputs of N_{P+1} to which a packet from i (of N_{P+1}) may be routed and f_{io}, $o \in O$ the forwarding probability then we have:

$$stop\_next\_node = \sum_{o\in O} f_{io}^{N_{P+1}}\cdot(dly\_port\_acq_{N_{p+1}}(io) + impact\_buf_{N_{p+1}}(o))$$

$$f_{io}^{N_{p+1}} = \frac{\lambda_{io}^{N_{p+1}}}{\sum_{i\in O}\lambda_{io}^{N_{p+1}}}$$

As stated $stop\_next\_node$ is a function of acquisition delays and impact buffers of all output $o \in O$ of N_{P+1} and recursively triggers the same computation for all output port of node N_{p+1} which are similarly dependent on their following nodes. As mentioned before this dependency to following nodes makes a graph of dependency that must be covered from its tails (i.e. cores) and in a reverse order. So by implementing a recursive algorithm and calling it for a desired node we are able to cover the whole graph. The recursive algorithm traverses the graph until the cores and then it computes the mean latency of each node of the graph and uses it for computing the mean latency of its backward dependent node and continues until the latency of the desired node obtained.

2) FIFO utilization

The average utilization of the fifo connected to output port o is equal to the data forwarding rate of port o (i.e. the sum of all of forwarding rates λ_{io}) plus the average rate of the occupied buffer space after the transfer of packets for all possible input port i. Note that $dly\text{-}buf_o$ is the average occupied buffer space after the transfer of a specific packet and to obtain the rate of occupation it must be multiplied by the forwarding rate.

$$mean\_buf\_tilzation(o) = \sum_{i \in I} \lambda_{iO}(1 + dly\_buf_O(i))$$

IV. Experimental results

In order to experimentally evaluate our method we have targeted a NoC architecture arranged in a two dimensional mesh topology with an x-first (X-Y) routing algorithm [16]. 2D-mesh topology and x-first routing algorithm are adequate choices for regular and homogeneous NoC architectures[7]. We choose a uniform random traffic pattern which according to [17] is the most common traffic pattern used for NoC performance evaluations. In this experiment we assume that cores (local subsystems) consume the incoming packets with the constant rate of one flit per cycle. The results for a 5x5 2D-mesh NoC are presented. Mean packet length set to 16 flits which is typically the length of a cash-line request. Offered-load is expressed with the average number of flits per 100 cycles per core. Latency vs. offered-load curve for path $N_{5,1} \rightarrow N_{1,5}$ (the diagonal path of the NoC) is presented in fig. 4. Obtained result is compared with the result of a corresponding SystemC CABA simulation platform according to which the service time of header flits (*dly_hdr_serv*) is set to 1 cycle and *dly_const_buf* to 2 cycles. In simulations for each offered-load we have taken the average latency of 1000 received packets per destination. As it can be observed, the inaccuracy of our approach (compared with the simulation results) is less than 5% for offered-loads less than 35%. After this the offered-load the NoC begins to be saturated and the latency tends to infinity. Our method predicts a saturation threshold of 37% for this NoC architecture which is very close to what the simulation demonstrates. Comparing to results of the method proposed in [11] our method seems to provide a more accurate approximation for the saturation threshold.

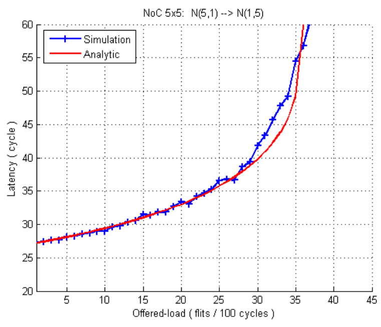

Fig.4 latency/load curve for $N_{5,1} \rightarrow N_{1,5}$

Fig.5 shows average buffer utilizations for all buffers along the path $N_{5,1} \rightarrow N_{1,5}$. Buffers are numerated from source to destination (1=buffer $N_{5,1} \rightarrow N_{4,1}$ etc.) at offered-load=30%. Table.1 compares simulation and analytic run-time for analyzing the diagonal path (the longest path form bottom-right to top-left) in a 2D-mesh NoC with different dimensions.

V. Conclusion

In this paper a novel numerical analysis method has been proposed for computing the mean packet latency between two nodes of a given NoC. It also provides average buffer utilization and port-to-port router latency, dominated by both port-acquisition and link-transfer delays. It finds as well the saturation threshold of the network within a good accuracy (comparing to simulation). The experimentally evaluation of the method and comparison with the corresponding CABA SystemC simulations have shown that the method provides quite accurate results.

The methodology is generic and adaptable to arbitrary NoC topology and deterministic deadlock free routing algorithms. It supports arbitrary packet and buffer length with a worm-hole routing policy and works for applications that could be formalized into the form of a distribution traffic matrix.

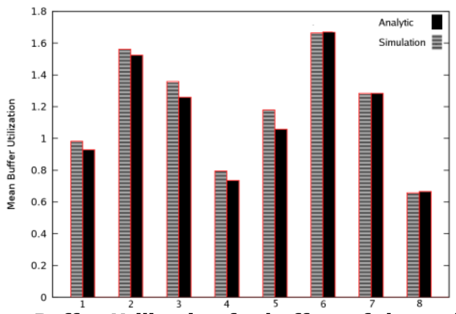

Fig.5 Mean-Buffer-Utilization for buffers of the path $N_{5,1} \rightarrow N_{1,5}$ at offered-load 30%

Table.2 Run-Time comparison

NoC	Path	Simulation Run-Time (second)	Analytical Run-Time (second)
4x4	$N_{4,1} \rightarrow N_{1,4}$	9332 (~ 3 hours)	1
5x5	$N_{5,1} \rightarrow N_{1,5}$	16121 (~ 5hours)	2
6x6	$N_{6,1} \rightarrow N_{1,6}$	34415 (~10 hours)	6
10x10	$N_{10,1} \rightarrow N_{1,10}$	More than 48 hours	46

Although currently we do not cover the case of several VCs, we believe that if the distribution of traffic on each virtual *network* is specified (to have one traffic-matrix and one routing function *per virtual network*) then it is possible to extend our method to several stages of arbitration within each router. A comprehensive study of this possibility is left for future work.

References

[1] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Des. Test,* vol. 22, pp. 414-421, 2005.

[2] T. Bjerregaard, "The MANGO Clockless Network-on-Chip: Concepts and Implementation." vol. PhD thesis: Technical University of Denmark, 2005.

[3] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework," in *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*: IEEE Computer Society, 2005.

[4] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi, *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*, 2008.

[5] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Legacy SystemC co-simulation of multi-processor systems-on-chip," 2002, pp. 494–499.

[6] S. G. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O. P. Gangwal, "Cost-Performance Trade-Offs in Networks on Chip: A Simulation-Based Approach," in *Proceedings of the conference on Design, automation and test in Europe - Volume 2*: IEEE Computer Society, 2004.

[7] A. Sheibanyrad, I. Miro Panades, and A. Greiner, "Systematic comparison between the asynchronous and the multi-synchronous implementations of a network on chip architecture," in *Design Automation and Test in Europe (DATE)* Nice, France: EDA Consortium (IEEE, ACM), 2007, pp. 1090-1095.

[8] P. Hu and L. Kleinrock, "An analytical model for wormhole routing with finite size input buffers," 1997, pp. 549-560.

[9] A. E. Kiasari, D. Rahmati, H. Sarbazi-Azad, and S. Hessabi, "A Markovian Performance Model for Networks-on-Chip," 2008, pp. 157-164.

[10] W. J. Guan, W. K. Tsai, and D. Blough, "An analytical model for wormhole routing in multicomputerinterconnection networks," 1993, pp. 650-654.

[11] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis," 2007, pp. 1-6.

[12] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers,* vol. 39, pp. 775-785, 1990.

[13] A. Khonsari, M. Ould-Khaoua, and J. Ferguson, "A General Analytical Model of Adaptive Wormhole Routing in k-Ary n-Cube Interconnection Networks," *SIMULATION SERIES,* vol. 35, pp. 547-554, 2003.

[14] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Network Delays and Link Capacities in Application-Specific Wormhole NoCs," *Special Issue of the Journal of VLSI Design,* 2007.

[15] E. Salminen, A. Kulmala, and T. D. Hamalainen, "On network-on-chip comparison," in *DSD 2007. 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, 2007.

[16] W. J. Dally, *Principles and practices of interconnection networks*: Morgan Kaufmann, 2004.

[17] E. Salminen, A. Kulmala, and T. D. Hamalainen, "On network-on-chip comparison," 2007, pp. 503-510.