Effects of the NoC Architecture in the Performance of NoC-based MPSoCs

Douglas R. G. Silva, Bruno S. Oliveira, Fernando G. Moraes
PUCRS University, Computer Science Department, Porto Alegre, Brazil
douglas.roberto@acad.pucrs.br, bruno.scherer@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—The goal of this work is to evaluate the impact of multiple Network-on-Chip (NoC) architectural parameters over the performance of applications running on Multiprocessors Systems-on-Chip (MPSoCs) using message passing as communication protocol. Nowadays, MPSoCs have so many constraints of performance that bus-based communications are not able to achieve the full potential of MPSoCs, therefore, the adoption of NoCs is a trend for the communication infrastructure in MPSoCs due to their performance compared to bus-based architectures and scalability compared to crossbar-based architectures. However, there is an important gap in the literature with works evaluating the impact of NoC parameters in the performance of applications running in MPSoCs. This work proposes the evaluation of how different NoC parameters affect applications running in a real MPSoC, trying to answer the following question: how does a given NoC parameter affect the performance of the MPSoC?**

Keywords—NoC-based MPSoC; NoCs; routing; topology; performance evaluation.

I. INTRODUCTION

MPSoCs may be designed as general-purpose platforms, being able to run a set of different applications with different performance constraints, or designed to execute a given application, with its architecture optimized at design time [1]. The interconnection architecture used in MPSoCs varies. Some Authors have foreseen that bus-based and point-to-point connections would not fulfill the communication requirements of today's integrated circuits. NoCs are an alternative, due to their efficient power usage and reliability [2], scalability of bandwidth (compared to buses) [3] and reusability.

State-of-the-art of NoC research presents many works evaluating application specific NoC designs [4]. However, a gap in the literature was identified: how *real* applications running in an MPSoC based on message passing are affected by different NoC architectural parameters? The *goal* of this work is to generate these missing metrics, and thus to evaluate how different NoC parameters influence the performance of applications executing in NoC-based MPSoCs.

An MPSoC consists of a set of processing elements (PEs) interconnected by a given network topology. This work adopts common features found in MPSoCs [5]:

(i) each PE contains at least one processor with a private memory;
(ii) applications are modeled as task graphs;
(iii) communication model is message passing;
(iv) there is no shared memory between the PEs;
(v) a multi-tasking operating (OS) system runs at each PE;
(vi) a mapping function maps tasks onto PEs, being possible to have more than one task per PE.

The communication API adopts two MPI-like primitives: a non-blocking *Send()* and blocking *Receive()*. To implement a non-blocking *Send()*, a dedicated memory space in the OS, named *pipe*, stores each message written by tasks. Using such communication method, a packet is injected into the NoC only when it is requested, in burst mode. This feature reduces the NoC congestion, because a given PE is waiting the packet for consumption, and the burst mode avoid idle time between flits.

Two different PEs instances are used: manager PE (PE_M), and slave PEs (PE_{SL}). The PE_M is responsible for the management of system resources. For small MPSoCs (e.g. 5x5 PEs), one PE_M may be sufficient, characterizing a centralized approach. For large MPSoCs, distributed management is adopted, partitioning the MPSoC in clusters, with one per PE_M cluster [6]. Slave PEs execute applications' tasks. Regardless the PE instance, all PEs contains the following components, as illustrated in Fig. 1:

(i) a Plasma processor [7], a 32-bit RISC processor with a subset of the instructions from the MIPS2000 architecture;
(ii) a dual-port private memory (RAM), which stores the OS, named *microkernel*, and applications' tasks (the memory is organized in equally sized pages);
(iii) a direct Memory Access (DMA) module enabling to send/receive data to/from the NoC, without interrupting the processor;
(iv) a network interface (NI), which connects the NoC to the processor and to the DMA, responsible for implementing the communication protocol with the NoC.
(v) Message passing is the communication mode. There is no shared memory in the system.

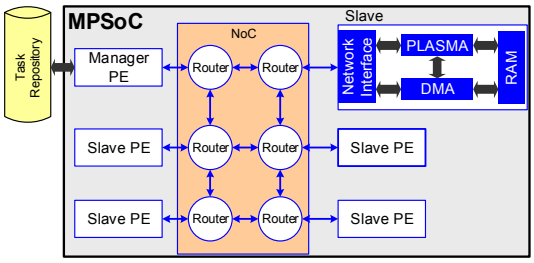


Fig. 1. An instance of the MPSOC with 6 PEs.

An external memory, named task repository, contains the object code for all tasks that possibly run on the system. The PE_M is the only PE accessing such memory, being responsible for transmitting the tasks' object code to the slave PEs, according to some dynamic mapping heuristic [8].

II. NOC ARCHITECTURAL PARAMETERS

A. Buffer Depth

Buffers are an integral part of the network router. In most NoC architectures, the buffers are responsible for the largest portion of router silicon area and dissipated power. Therefore, it is an advantage to reduce the buffer depth.

B. Routing Algorithms

The present work evaluates the performance of five routing algorithms: XY, West-First, Negative-First, Odd-Even and the Torus Routing Algorithm for Network on Chip (TRANC).

The XY routing algorithm is deterministic, meaning that packets always follow the same path, even when there are congestions. The

XY algorithm compares the target packet address to the current router address. When they are the same, the algorithm redirects the packet to the local port. If not, the packet is first sent in the horizontal direction until it reaches the target router column. Then, the packet is sent in the vertical direction, until it reaches its destination.

The West-First (WF) algorithm [9] is partially adaptive. When the target router address is in the same column of the packet or in the West of the current router, packets are routed deterministically, similarly to the XY algorithm. When the target router is located to the East of the source router, adaptive routing is used. The Negative-First (NF) [9] algorithm is similar to the WF algorithm, being also partially adaptive. It considers North and West directions as "negative" directions and South and East as "positive" ones. The algorithm forbids that packets take curves from positive directions to a negative one.

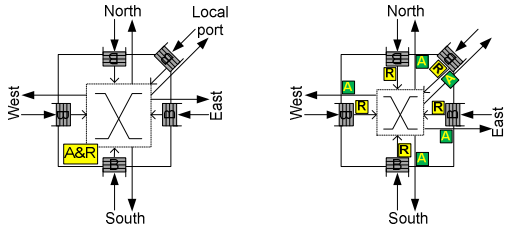
The Odd-Even (OE) is an adaptive routing algorithm [10]. The OE does not impose a limitation to packet directions. It allows packets to route adaptively in every direction. To avoid deadlocks, two particular rules limit the turns in specific columns. It interchangeably calls a column as *odd* or *even*, according to the coordinates of the specific columns. The turning model follows two rules: (*i*) a packet cannot execute a turn from East to North when in an *even* column and it cannot perform a turn from North to West when in an *odd* column; (*ii*) a packet cannot execute a turn from East to South when in an *even* column and it cannot perform a turn from South to West in an *odd* column. Applying these two rules, packets can take any other turns to avoid congestions.

The TRANC algorithm is a deterministic routing algorithm designed for torus networks [11]. It is evaluated to another available routing algorithm for torus, an adapted WF algorithm.

C. Arbitration

In particular cases, multiple packets arriving in different input ports may require the same output port. Therefore, it is necessary to schedule the output port aiming to provide a fair usage of the output channel by the input channels. This scheduling must be done ensuring that no packet waits indefinitely to be directed to its target. The act of scheduling the usage of the output port is the responsibility of router arbiters. This work evaluates two implementations of the arbitration: centralized and distributed round robin. Note that both implementations are at the *router level*, not implying a centralized knowledge of the entire NoC status.

The centralized arbitration consists in a single module for each router, managing the use of all output ports in this specific router (Fig. 2(a)). This implementation may induce a small performance penalty because the arbiter cannot schedule more than one packet simultaneously.



(a) Centralized control (b) Distributed control

Fig. 2. Router architectures (B: input buffer, A: arbitration, R: routing logic).

In the distributed arbitration, each router port controls independently the routing and the arbitration (Fig. 2(b)). Each input port has a routing module associated to it, and each output port has an arbiter. The routing module requests a specific output port and the arbiter selects one specific input port if it has received multiple requisitions. This implementation is faster, because multiples packets can be allocated to their output port simultaneously, however, it requires more area, since it is necessary to replicate the routing and arbitration logic for each port.

D. Topology

The present work evaluates the 2D-mesh and the 2D-torus topologies. Both are known for their advantages, as scalability for the mesh and the smaller diameter for the Torus. When the topologies are compared, both use the same routing algorithm, with the purpose to quantify the impact of the topology in the performance.

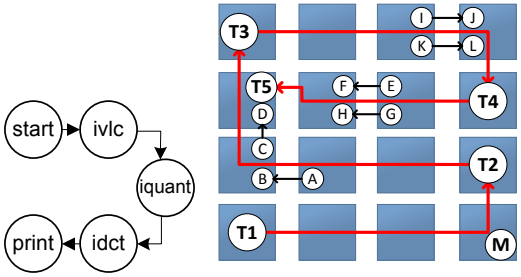
In the mesh architecture, except the routers at the edges, all other routers have four neighboring routers, and a PE connected to a local port. Its major advantage is scalability, being possible to connect a large amount of PEs in a regular-shape structure.

The torus topology is also based on an *m x n* mesh, but it is characterized by connecting all edges to their equivalent on the opposite side. The folded torus topology is a topology where the physical placement of the routers are made in such a way that the wires always have the same length (2 hops), thus, minimizing the problem of long links connecting edge routers.

III. EXPERIMENTAL SETUP

The MPSoC is modeled in VHDL (NoC, NI, DMA) and SystemC (processors and memory), using an RTL description. Such model is clock-cycle accurate. This model allows accurate measurement of performance values. Results are obtained from the evaluation of two scenarios, corresponding to a real use of the MPSoC (the applications are managed by a small multi-task operating system). In both simulated applications, data packets have in average 128 flits. Small flits (6 to 8 flits) are typically control packets (e.g. read request, mapping request, end of a task, etc.).

The first scenario uses an *mpeg* decoder as the application under evaluation (described in C language), with a set of *producer/consumer* applications generating disturbing traffic. The *mpeg* application contains 5 tasks: *start, ivlc, idct, iquant, print.* The *start* task sends frames to the *ivlc* task. Tasks *ivlc, idct, iquant* are responsible for executing the decoding algorithm. The last task, *print*, receives the decoded frames. A 4x4 MPSoC instance is used. Fig. 3 presents the application task graph and its mapping, as well as the disturbing flows (A→B, C→D, etc.).

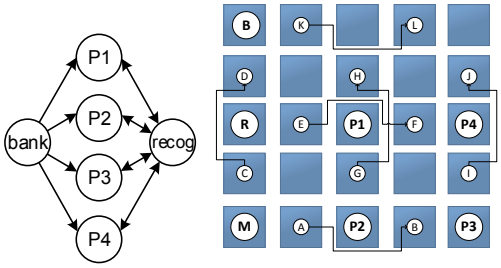


(a) Application Graph (b) Application Mapping

Fig. 3. First evaluated scenario (T1: start.c; T2: ivlc.c; T3: idct.c; T4: iquant.c, T5: print.c). M is the manager processor.

The PE_M (M) is mapped in the bottom-left PE. Each *producer/consumer* (PC) application contains 2 tasks, generating traffic at the maximum processor rate (on average 15% of the available link bandwidth). The producer generates dummy packets, with a payload containing 200 flits, addressed to the consumer task. The goal of this application is to disturb the application under evaluation.

The second scenario uses the *DTW* (dynamic time warping) as the application under evaluation, also with a set of *producer/consumer* applications generating disturbing traffic. The *DTW* application contains seven tasks. Fig. 4 presents the application task graph and its mapping. Differently from the MPEG application, which has a pipeline behavior, in this application several tasks communicate with the same task (e.g. *P1* to *P2* read from task *bank*). Therefore, much more interferences between flows are expected.



(a) Application Graph (b) Application Mapping

Fig. 4. Second evaluated scenario. M is the manager processor. Only the disturbing flows are illustrated.

Both task mapping scenarios benefits the *torus* topology and adaptive routing, since a deterministic algorithm would be affected by the disturbing traffic. Thus, these mapping scenarios can highlight the advantages of these techniques.

Three performance metrics were evaluated: (1) **Network latency:** the amount of clock cycles between the injection of the header flit into the network and the reception of the last flit; (2) **Jitter**: standard deviation of the network latency; (3) **Execution time**: time to execute a given application. The obtained results are derived from the entire simulation time, including the beginning of the simulation where the object code of the tasks is transmitted to the slave processors. The warm-up time is not considered, due to the long simulation time.

The applications chosen by the Authors, as an MPEG decoder and the DTW, are representative applications for MPSoCs. Even with a heavy disturbing traffic, the execution time was not affected varying the NoC architectural parameters. The reason is that the bandwidth offered at each NoC link (1.6 Gbps – 16 bits@100 MHz) is enough to support the traffic of these applications, together with disturbing flows.

IV. RESULTS

A. Buffer Depth

Four different buffers depths were evaluated: 4, 8, 16 and 32 flits. Results are extracted from a network with a *mesh* topology, *XY* routing algorithm, and centralized arbitration (Fig. 2(a)). When no disturbing traffic is added, the average latency is practically the same for different buffers configurations, and consequently no jitter is observed.

Adding the disturbing traffic in the MPEG scenario, the average latency is almost the same for different buffers configuration, as shown in Table I. In this specific example, for flow T2→T3 the smaller latency is observed with a buffer depth equal to 4, while in flow T4→T5 with a buffer depth equal to 32. The observed differences, for each flow, are not greater than 5%.

TABLE I. AVERAGE LATENCY AND JITTER (CLOCK CYCLES) FOR DIFFERENT BUFFER DEPTHS, WITH DISTURBING TRAFFIC, FOR THE MPEG APPLICATION.

	Flow T2 → T3				Flow T4 → T5			
	4	8	16	32	4	8	16	32
Min	416	410	410	410	404	400	400	400
Avg	447	450	455	468	549	549	543	538
Max	883	1323	1455	1227	1169	1235	1118	1236
Jitter	98	120	152	138	175	174	174	176

Table II present latency and jitter values for two flows of the DTW application. The same behavior is observed: there is not a buffer size ensuring smallest latency. In this scenario, DTW application, larger variations are observed (up to 20%), due to the competition among the flows of the DTW application.

Similarly to the latency evaluation, the buffer depth is not correlated to the jitter, as shown in the last lines of Table I and Table II. In some specific cases, there is an important jitter reduction, as in flows T2→T3 and bank→P1, due to the mapping of the tasks.

TABLE II. LATENCY AND JITTER (IN CLOCK CYCLES) FOR DIFFERENT BUFFER DEPTHS, WITH DISTURBING TRAFFIC, FOR THE DTW APPLICATION.

	Flow Bank →P4 (DTW)				Flow Bank →P1 (DTW)			
	4	8	16	32	4	8	16	32
Min	536	529	529	529	524	519	519	519
Avg	655	645	749	706	672	666	733	589
Max	1808	2090	2186	1842	1721	1722	1792	1630
Jitter	276	284	367	331	305	295	351	170

The impact on the overall execution time when changing the buffer depth was minimal. For the MPEG application the execution time of the tasks was in average 57.5 ±0.2 ms, and for the DTW the execution time of task varied between 77.8 and 78.1 ms.

Table III presents the router area and the estimated power consumption for a 5-port router, XY routing, centralized arbitration, in a 65 nm technology. Each added buffer slot increases the area by 130 μm^2 in average for each buffer. In the same way, the average router power consumption is proportional to the buffer depth.

TABLE III. ROUTER AREA AND ESTIMATED POWER (5-PORT ROUTER) FOR A 65 NM TECHNOLOGY (LIBRARY CORE65GPSVT).

Buffer depth		4	8	16	32
Cell area in μm^2 tech 65 nm	Router	7,628	10,241	15,468	24,671
	Buffer	1,153	1,675	2,727	4,568
	Crossbar	1009	1009	1009	1009
	Control	862	862	862	862
Total Power (mW)		0.839	0.912	1.130	1.480

The buffer depth is the first NoC parameter evaluated. The NoC buffer is used to compensate the amount of time required by arbitration and routing. In an MPSoC environment, injection rates are small (bellow to 10%), not requiring in this way a buffer to amortize the performance penalty induced by the router neither to store packets due to congestion. *Therefore, this work proposes the use of small buffers, 4 or 8 flits depth, to reduce the NoC area and power.*

B. Mesh Routing Algorithms

Table IV present latency and jitter values for 4 different routing algorithms. The XY-D uses distributed control (Fig. 2(b)). With no disturbing traffic, latency values are similar, presenting a small variation of 2.2%. A different scenario is observed when disturbing traffic is added. Using XY as reference, the adoption of WF or NF reduced 5% the latency in flow T3→T4, and the adoption of OE 14% reduction in the flow bank→P4.

TABLE IV. LATENCY AND JITTER (IN CLOCK CYCLES) FOR DIFFERENT ROUTING ALGORITHMS, WITH DISTURBING TRAFFIC.

	MPEG, flow T3 → T4					DTW, flow Bank → P3				
	NF	OE	WF	XY	XY-D	NF	OE	WF	XY	XY-D
Min	409	405	410	405	400	547	539	548	539	530
Avg	410	435	410	432	437	675	581	676	706	673
Max	417	838	411	865	856	2438	980	1936	1936	1955
Jitter	0.7	92	0.5	85	102	310	110	305	305	291

In addition, the routing adaption enabled to reduce the maximum latency. As can be observed in Table IV, for flow T3→T4 NF and WF presented the smaller latency variation, while in flow bank→P3 the OE routing algorithm.

The adoption of distributed routing and distributed arbitration (XY-D) does not bring gains in the evaluated scenarios. The reason explaining such behavior is the small contention observed in the MPSoC. *Therefore, simple NoCs, with routers having centralized arbitration and routing are recommended.*

The impact on the overall execution time when changing the routing algorithm was also minimal.

The routing algorithm in mesh networks is the second NoC parameter evaluated. The net effect of the routing algorithm seems to be small. However, it may strongly reduce the jitter, an important parameter for applications with QoS. *The Authors recommend adopting MPSoCs with adaptive routing algorithms, since they potentially may benefit applications.*

C. Topology Comparison

The major weakness of the torus topology is the wraparound wires, which may be mitigated using a folded torus topology, increasing the wire length twice compared to the mesh topology. The torus architecture should present a smaller latency and jitter when packets travel from one edge to the other of the mesh. To highlight the advantage of the torus topology it was chosen as observed flows the communication T4→T5 (MPEG) and bank→P4 (DTW). To evaluate just the effect of the topology we used the same routing algorithm for both simulations, the WF routing algorithm.

Flow T4→T5 presents 1 hop in the torus topology and 3 hops in the mesh topology. In the scenario with no disturbing traffic, the latency is constant, being 404 and 394 clock cycles for the mesh and torus topologies respectively. The mesh topology, in this scenario, has disturbing flows E-F and H-G in the path T4→T5 (MPEG), presenting higher latency variation - Table V. The torus topology presents an average latency 16.8% smaller, with a smaller jitter. The observed variation comes from the PE sharing, between tasks *print* (T5) and D. The same behavior is observed with flow bank→P4, which used the wraparound link to reduce the latency. An overall observation is that in all scenarios the torus topology reduced the latency values (remember that the mapping of the simulated scenarios, Fig. 3 and Fig. 4, favors the torus topology).

The jitter with disturbing traffic, last row of Table V, is 27.3% and 28.0% smaller with the torus topology for flows T4→T5 and bank→P4, respectively.

TABLE V. LATENCIES AND JITTER VALUES OF FOR MESH AND TORUS TOPOLOGIES, WITH DISTURBING TRAFFIC, IN CLOCK CYCLES.

	Flow T4 → T5 (MPEG)		Flow bank→P4 (DTW)	
	Mesh	Torus	Mesh	Torus
Min	404	394	536	522
Avg	566	489	664	594
Max	1188	1026	1803	1623
Jitter	191	139	303	218

The impact on the overall execution time when changing the routing algorithm was minimal.

The topology is the third NoC parameter evaluated. In a *homogeneous* MPSoC, applications hardly could benefit from a torus topology, since mapping heuristics have as cost function to place communicating tasks as close as possible (obviously it is possible to create dedicated mapping heuristics for torus topologies). *However, the torus is a choice if IPs have fixed mapping at the edges of the mesh, as shared memory controllers.* In such a case, the access time for such memories would benefit from a torus topology.

D. Torus Routing Algorithms

This section evaluates how the routing algorithm affects the network latency and jitter in torus networks. Two routing algorithms are evaluated: TRANC and WF. Results presented in Table VI are extracted with 16-flit buffers depth.

TABLE VI. LATENCY AND JITTER VALUES FOR DIFFERENT MPEG FLOWS, FOR DIFFERENT TORUS ROUTING ALGORITHM IN CLOCK CYCLES.

	MPEG, flow T2 → T3		MPEG, flow T4 → T5		DTW, flow bank → P4	
	TRANC	WF	TRANC	WF	TRANC	WF
Min	408	408	394	394	522	522
Avg	408	435	510	489	522	594
Max	408	855	1026	1026	525	1623
Jitter	0	85	166	138	0.5	218

With disturbing traffic, the TRANC algorithm shows an advantage of 6.6% when compared to the WF routing algorithm for flow T2→T3 (avg row in Table VI). The WF takes the horizontal path first, thus being affected by the disturbing CD. The TRANC takes the vertical path first, being not affected by the disturbing traffic. Using another flow, T4→T5, the WF algorithm had an advantage of 4.4% compared to the TRANC algorithm. In this situation, the WF used the wraparound link, while the TRANC not. The flow bank→PW (DTW) is another example where TRANC performance is better, with an almost zero jitter.

The last line of Table VI compares the jitter of the flows, with disturbing traffic, demonstrating that the jitter is a function of the task mapping coupled to the adaptation strategy of the routing algorithm. Therefore, there is not a general rule to select the best routing algorithm.

V. CONCLUSION

The NoC literature contains many works evaluating the NoC performance according to various NoC parameters. *However, in the Authors' knowledge, this is the first work exploring how the NoC parameters affect the performance of NoC-based MPSoCs*. Some important guidelines may be advanced: (1) buffers: keep them as small as possible; (2) router architecture: adopt centralized control, avoiding the replication of the arbiter and the routing logic; (3) routing algorithm: adopt adaptive routing, as WF/OE, since they may reduce latency and jitter; (4) topology: adopt 2D-mesh topology for homogeneous MPSoCs, and folded tours for heterogeneous MPSoCs.

Results point out that in MPSoCs using message passing, real applications (as MPEG and DTW) does not generate enough traffic to stress the NoC. A naïve explanation could be: the NoC is idle most of the time, because processors are executing theirs tasks. In fact, the reduction of the injection rate was obtained by buffering messages by the OS, and them transmitting them in a burst. Without burst transmission, flits would be injected in the NoC as they would be generated by the software, resulting in idle intervals between flits, increasing the overall usage of the NoC links. This does not apply to MPSoCs using shared memory, since there is an overhead in the NoC due do the cache coherence protocol.

Therefore*, the design of the NoC router may be the simplest one*, reducing the area and power overhead due to the communication infrastructure. It is important to remember NoCs offer parallels transactions and scalability, features not supported by bus-based infrastructures.

ACKNOWLEDGEMENTS

The author Fernando Moraes acknowledges the support granted by CNPQ, processes 472126/2013-0, 550118/2013-6 and 302625/2012-7; and CAPES process 708/11.

REFERENCES

[1] Singh, A.K.; Kumar, A.; Srikanthan, T. *A Design Space Exploration Methodology for Application Specific MPSoC Design*. In: ISVLSI, 2011, pp. 339 - 340.

[2] Yong Z.; Pasricha, S. Reliability-*aware and energy-efficient synthesis of NoC based MPSoCs*. In: ISQED, 2013, pp. 643-650.

[3] Kumar, R.; Deshpande, H.; Choi, G.; Sprintson, A.; Gratz, P. *Bidirectional interconnect design for low latency high bandwidth NoC.* In: ICICDT, 2013, pp. 215-218.

[4] Modarressi, M.; Tavakkol, A.; Sarbazi-Azad, H. *Application-Aware Topology Reconfiguration for On-Chip Networks.* IEEE Transactions on Very Large Scale Integration Systems, v.19(11), 2011, pp. 2010-2022.

[5] Garibotti, R., et al. *Simultaneous Multithreading Support in Embedded Distributed Memory MPSoCs*. In: DAC, 2013, 6p.

[6] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F. *Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes*. In: ISVLSI, 2013, pp 153-158.

[7] *Plasma CPU*. Available at: http://plasmacpu.no-ip.org.

[8] Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. *Mapping on multi/many-core systems: Survey of current and emerging trends*. In: DAC, 2013, 10p.

[9] Glass, C.J.; Ni, L.M. *The Turn Model for Adaptive Routing*. In: ISCA, 1992, pp. 278-287.

[10] Ge-Ming Chiu. *The odd-even turn model for adaptive routing.* IEEE Transactions on Parallel and Distributed Systems, v.11(7), 2000, pp. 729-738.

[11] Rahmatia, D.; Sarbazi-Azada, H.; Hessabia, S.; Kiasarib, A. *Power-efficient deterministic and adaptive routing in torus networks-on-chip.* Microprocessors and Microsystems, v.36(7), 2012, pp. 571–585.