Software-Based Mechanism for Network-on-Chip Performance Increase

Marcelo Linck, Gabriel Paz, Augusto Santos, César Marcon

Pontifical Catholic University of Rio Grande do Sul – Porto Alegre, Brazil
{marcelo.linck, gabriel.rodrigo, augusto.pasqualotto}@acad.pucrs.br; cesar.marcon@pucrs.br

***Abstract*— This work proposes the Quality-of-Service Monitoring and Controlling Driver (QoSMCD), a software-based mechanism that improves the Network-on-Chip (NoC) performance by controlling the packet injection rate, thus reducing link congestion without modifying the hardware. The proposed mechanism works together with the task scheduling policy for controlling the injection of packets into the NoC. This work serves as a basis for a high-level control system for traffic injection implemented in cooperation with the scheduling of a real-time operating system.**

Keywords—Network-on-Chip; Flow control; Latency; NoC; Software-based; Communication efficiency.

I. INTRODUCTION

Multiprocessor System-on-Chip (MPSoC) based on Network-on-Chip (NoC) allows implementing applications with high degree of parallelism in communication and computation [1]. In the past few years, some works (e.g., [2]-[6]) have been addressed the monitoring and control of the packet Injection Rate (IR) to avoid reaching link saturation, achieving a desirable Quality-of-Service (QoS) on the NoC communication. Ogras et al. [2] proposed an architecture that predicts and avoids congestion, controlling the number of packets injected into the NoC and, thus, delaying the traffic. Nousias et al. [3] projected an online adaptive flow control mechanism for wormhole switching, which uses virtual channels and control signals to adjust the input rate aiming to match output rates. Wang et al. [4] proposed a global feedback-based flow control that allocates channel bandwidth and buffer space based on the feedback signals of the congested network resources. The packets destined to the congested nodes are placed in a temporary buffer to avoid increasing the congestion. Tang et al. [5] proposed a link status monitor and control indicating the amount of data flow sharing the same channel. This information allows readapting the injection level avoiding NoC saturation. Yuan et al. [6] designed a congestion-aware routing mechanism in which a congestion agent is set near the congestion areas to maintain and control it, throttling its links and detouring packets paths to help overcome the congestion.

Despite all works presenting satisfactory results, none of them cares about avoiding hardware changes. The works in [2]-[4] need to modify the router architecture to implement their technique. On [5], a new external network must be implemented to share their proposed pattern system. Finally, the implementation proposed in [6] requires modifying both the router and the network itself.

We propose the *Quality-of-Service Monitoring and Controlling Driver* (QoSMCD), which is a traffic control mechanism that works in coordination with the operating system scheduler. QoSMCD manages its actions on the NoC status, requesting the rescheduling of tasks to avoid link saturation. It is a software-only traffic flow implementation for controlling the packet IR. Thus, unlike previous works, no modification is made on the hardware side.

This paper is organized as follows. Section II presents the traffic distributions used in the experimental results and describes the saturation problem. Section III details our proposed solution. Section IV discusses the experimental results and the effectivity of our solution. Finally, Section V presents the conclusion and further investigations.

II. PROBLEM FUNDAMENTALS

Increasing the packet IR to a given level saturates the links, producing an excessive increase in latency and jitter, and limiting the rate that the packets are injected based on the NoC throughput. This work explores a set of three types of traffic that gave a significant variation to the experimental results: *One-to-One Complementary* (Complementary), *Hotspot to a single node* (Hotspot-0), and *Cluster Hotspot* (C-Hotspot). We may refer to this set of distributions as **set A**. Figure 1 shows the source and target of the routers for each type of traffic on set A.

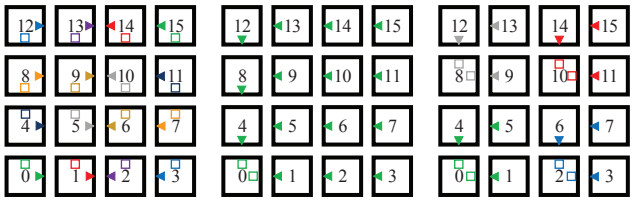

Figure 1 - Traffic distributions on set A: (a) Complementary, (b) Hotspot-0 and (c) C-Hotspot. The triangles indicate the output port of the packet burst on the source node; the squares indicate the receiving port on the destination node. Nodes with the same color are communicating.

In the Complementary traffic, the source nodes send packets to the destination nodes connected to the routers that complement the address of each source router. Consequently, the first node sends packets to the last node, the second node sends packets to the second-to-last node, and so on. Meanwhile, in Hotspot targeting node 0, all nodes are always sending data to node 0, purposely creating a congestion on the network. Finally, C-Hotspot organizes the NoC in four clusters containing four routers each. All nodes of each cluster send data to only one node, creating a hotspot in each cluster.

All experiment of this work uses Hermes NoC [7] with the following characteristics: 4×4 size; Round-Robin arbitration;

XY routing algorithm; 8-flit input buffer; 16-bit flit size; and without virtual channels. Furthermore, the all of traffic distributions use fixed 14-flit sized packets.

The NoC saturation point is a crucial issue that affects the traffic behavior because the delay/jitter becomes unpredictable, preventing to fulfill the communication requirements of some applications (e.g., real-time). Figure 2 depicts the behavior of the latency for the traffic distributions.

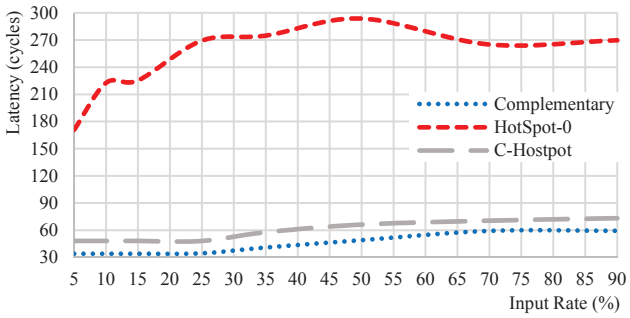

Figure 2 – Packet latency for the three chosen distributions on set A. Data represent the average value of latency for each input rate and each traffic distribution.

Complementary presents the lowest values of latency since each communication has a unique target node (traffic ratio of 1:1). Following comes C-Hotspot and Hotspot-0, with traffic ratios of 1:3 and 1:15, respectively. It is notable that the latency increases when the number of nodes sending packets to the same destination rises, due to the concurrency of packets on the same input port; it happens because the arbiter must select the input ports according to the arbitration algorithm, granting fair access. When a packet needs a high number of hops to reach the target node and other packets are competing for the same input port, the arbiter in each hop plays a meaningful role in latency increase. Figure 3 presents the behavior of the output rate for the traffic distributions.

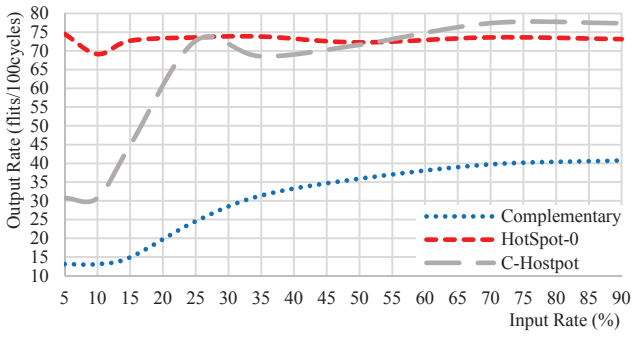

Figure 3 - Output traffic for the chosen traffic distributions on set A. The output rate is denoted by the number of flits the nodes are receiving in an interval of 100 cycles.

Hotspot-0 keeps its value almost constant along the plot in around 80 flits/100 cycles since node 0 is continuously receiving an enormous amount of data for all IRs. Meanwhile, C-Hotspot and Complementary start with low output traffic rate, increasing at around 25% of IR, the same moment the latency increases. This point is known as the saturation point, the moment when the output rate increases to its highest value and stabilizes. Also, all distributions present their lowest latency values for low IRs. This is an important characteristic of the saturation phenomenon; i.e., increasing the IR may not rise the output traffic rate after a certain point.

The value of the output rate of Hotspot-0 and C-Hotspot are high because they present higher communication ratios and a larger number of ports receiving data, which increases the number of ports with incoming data; therefore, reducing the idle time of the local port (due to the Round-Robin arbitration, the local port may become idle while the arbiter is selecting ports without incoming data). Since Complementary has a 1:1 traffic ratio, the number of ports receiving data is only one. Hence, the arbiter loses some cycles arbitrating the other ports, reducing the output rate.

III. The Proposed Methodology

NoC saturation depends on many factors, such as the: (i) width of the links, which is known at design time; (ii) operating frequency, which is defined at design time, but can be changed at runtime; (iii) router arbitration algorithm that in some cases, can reduce the rate of consuming packets by the router; and (iv) application traffic characteristics (e.g., the number of packets, packet size, and packet routing) that can vary widely during system execution. Only applications with deterministic computation and communication may have predictable traffic characteristics. These variabilities do not allow the designer to estimate at design time the maximum throughput of the network operation, requiring dynamic mechanisms to identify the network status. Besides, the traffic on the NoC does not saturate homogeneously; i.e., some channels may be saturated while other channels are operating with low traffic rate. The NoC traffic saturation may reduce the average throughput while increasing unordered and severely the variability of packet latency. Additionally, packets with the same size, but different communication paths can take dozens of times more cycles to perform the same distance on the NoC. This variability in the packet latency is perceived much more severely by distributed applications because it affects the application execution time due to the tasks synchronization and the data dependence.

QoSMCD is a driver implemented in coordination with the scheduler of the operating system that captures the latency of packets. According to the packet latency, QoSMCD reschedules communicating tasks to reduce NoC congestion. This approach reduces hardware costs and allows exploration of various network types for the same MPSoC with low implementation effort. The driver requires no modification on the communication network.

Inside the first packet of each communication, the QoSMCD of the source node inserts (i) a timing information, which is a timestamp containing the moment that the packet is injected into the NoC; (ii) a *maxLatency* value defining the maximum value acceptable for packet latency; and (iii) a *threshold* value, defining the number of packets with more than *maxLatency* that are accepted before taking any high-level control decision. Inside the remaining communication packets, QoSMCD only inserts the respective packet timestamp. The solution presents a relative low impact on packet size overhead, the timestamp information in all packets increase the packet in 32 bits (2 flits), and the first packet of the burst requires additional 16 bits (1 flit) to insert its information. Figure 4 depicts a Message Sequence Chart (MSC) exemplifying the main steps of the communication and rescheduling process.

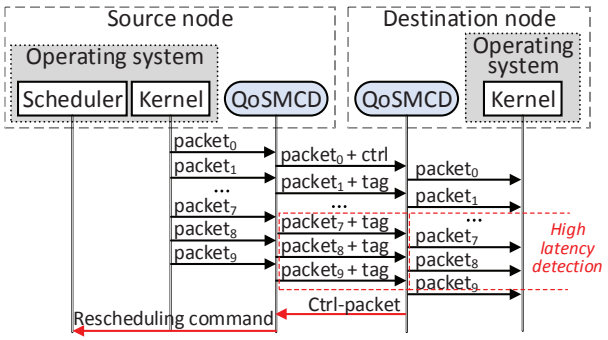

Figure 4 – MSC exemplifying the main steps of communication. $Packet_0$ inserts control information (ctrl), whereas the remaining packets insert only timestamps (tag). Packets 7, 8 and 9 have high latency requiring the target QoSMCD to transmit a control packet to the source QoSMCD.

If the QoSMCD in the destination node receives more than *threshold* times packets with latencies higher than the *maxLatency* in a row from the same source, this communication path is seen as congested. Thus, the QoSMCD in the destination node sends back to the QoSMCD in the source node control packet containing the task ID (the task Identification (ID) is part of the payload of all packets) causing the congestion. Upon receiving the 3-flit control packet, the QoSMCD fires a rescheduling request to the operating system scheduler that answers when possible.

IV. Experimental Results

This section shows experiments results using the same traffic distributions presented in Section II, competing with a newly introduced set. Here, we implemented the MPSoC functionality creating a packet injection module and a scheduler connected to each processor node that supports multiple synthetic tasks. When no external requests are made the scheduler uses the *First-Come, First-Served* algorithm.

Figure 5 shows the new set of distributions (**set B**) composed of tasks that compete during scheduling time with the three scenarios presented previously on set A. All nodes of the Complementary and the Hotspot-0 scenarios have tasks from both sets, and only four nodes of the C-Hotspot have more than one set.

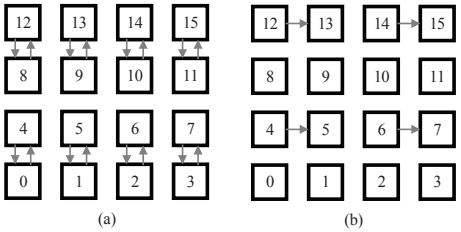

Figure 5 - Set B of scenarios used in coordination with set A, in order to validate the proposed solution. When scheduled, the new tasks follow the behavior on (a) for Complementary and Hotspot-0, and on (b) for C-Hotspot.

Figure 5 depicts the behavior of these scenarios when the new tasks are scheduled. Figure 5(a) shows the *Up and Down* distribution, which divides the NoC into two segments, the upper part and the bottom part, each one with equally 8 nodes. In each part, the nodes communicate vertically with their neighbors, having their destination not surpassing one hop. This distribution is used for the additional tasks of the Complementary and the Hotspot-0. In example, in one experimental setup, node 12 will have two tasks competing on the scheduler, one that sends packets to node 3, and another that sends packets to node 8. Following, Figure 5(b) presents the *To the Right* distribution, where only the upper-left node of each cluster is assigned with a new task. This task simply sends packets to its right neighbor. This have tasks that will compete with the C-Hotspot ones.

Figure 6, Figure 7 and Figure 8 contain six plots of latency and throughput (output rate) for each traffic distribution.

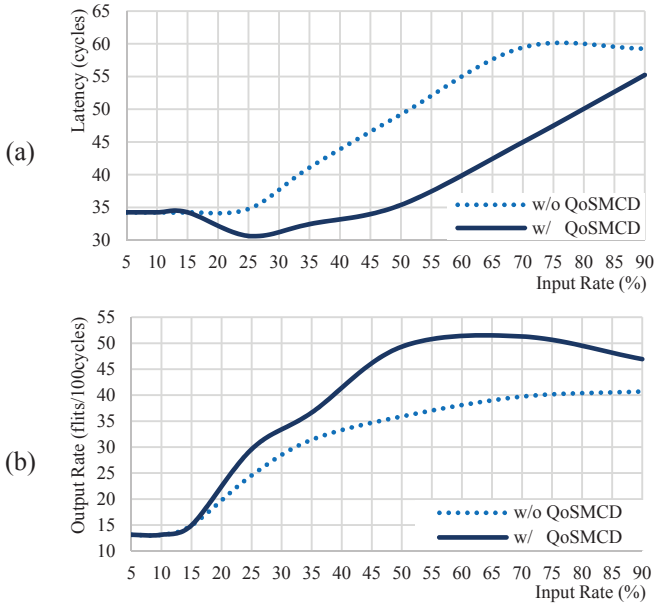

Figure 6 – (a) Latency and (b) output traffic rate for **Complementary** distribution when applying QoSMCD on the experiments depicted in Section II.

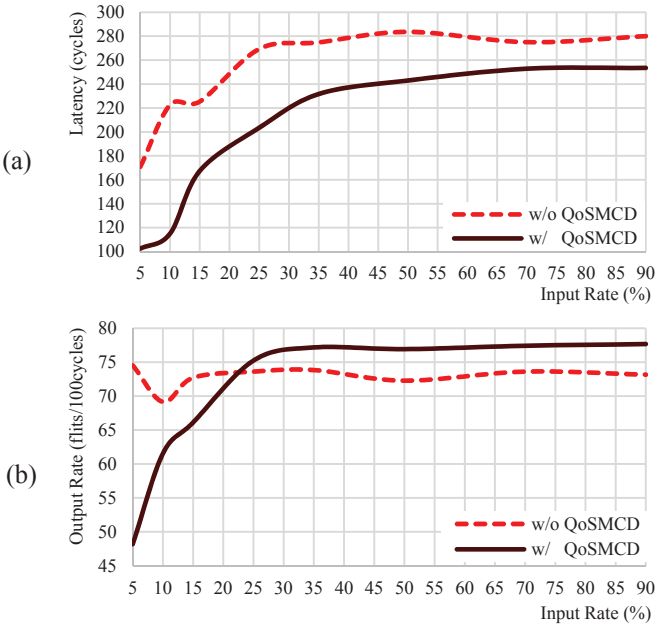

Figure 7 - (a) Latency and (b) Output traffic rate for Hotspot-0 distribution when using the proposed QoSMCD on the experiments of Section II.

Figure 6(a) and (b), show that QoSMCD played a significant role in reducing the latency and increasing the throughput for Complementary traffic, respectively. Initially, since the input rate is relatively small, the latencies did not hit a high value; therefore, both plots remain the same. Nonetheless, around the saturation point, it is possible to notice

the improvement of QoSMCD. The packets started having difficulties to reach their destination with a reasonable latency, and some schedulers were requested to reschedule their tasks to reduce the network congestion. The same happened to the output traffic rate since new tasks were scheduled, the average output rate in the whole NoC increased. In these cases, we noticed an average decrease up to 14.9% in latency and an average increase of up to 11.8% in output traffic rate.

Figure 7(a) and (b) show the behavior of the latency and output traffic rate for the Hotspot-0 distribution, respectively. Due to the Hotspot-0 traffic distribution presenting high latencies since the beginning of the simulation, the mechanism using QoSMCD acts in all the input rates. This setup presented an average reduction in latency of about 35.3%. Meanwhile, the output traffic rate initially decreases and later increases for lower and higher IRs, respectively. This is given by the effect of scheduling tasks that do not have zero as their destination, therefore, unburdening the node. In this case, the solution presented an overall output gain of 12.5%.

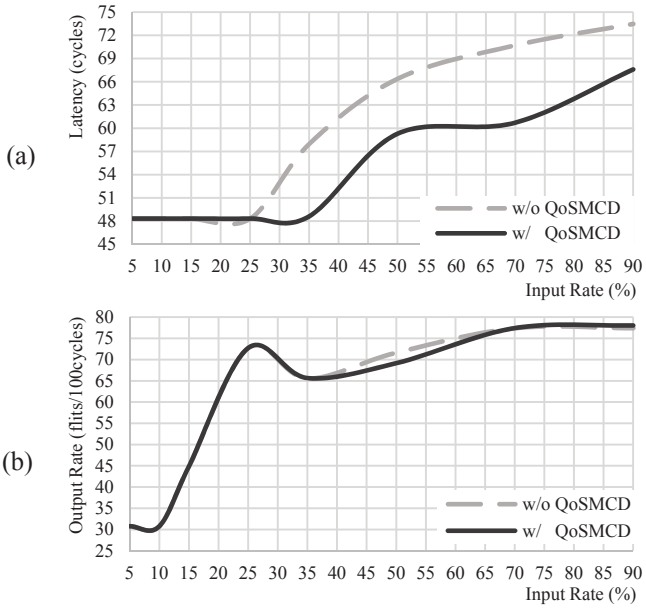

Figure 8 - (a) Latency and (b) output traffic rate for C-Hotspot distribution when applying the proposed QoSMCD on the experiments of Section II.

Finally, Figure 8(a) and (b) clarify the results of the mechanism using QoSMCD with a scenario under C-Hotspot distribution. In this traffic distribution, we analyze the effect of changing only one node sending data to a hotspot destination. In this case, by applying the set B tasks to the top-left nodes, QoSMCD provides an overall average decrease of 7% in latency. The output traffic rate maintains almost the same, although, at about the input traffic rate of 45%, there is a small increase that does not last long.

One final analysis of the proposed solution focuses on the execution time of tasks. Figure 9 presents the relation of last-flit *time variation* (i.e., packet jitter) between input traffic rates. In this case, *time variation* is calculated by the time the last flit reached its destination on a simulation with tasks minus the same moment for a simulation without tasks.

This timing analysis shows that using the scheduling mechanism with QoSMCD reduces the execution time of the tasks for the Complementary and Hotspot-0 distributions after the saturation point significantly, while it maintained the same for the C-Hotspot traffic distribution. This proved that our proposed solution could reduce the latency of packets and, therefore, increase the overall system efficiency. It is important to point out that these results were measured under extreme conditions; in other words, the traffic distributions were chosen to present sound effects that the proposed mechanism can provide.

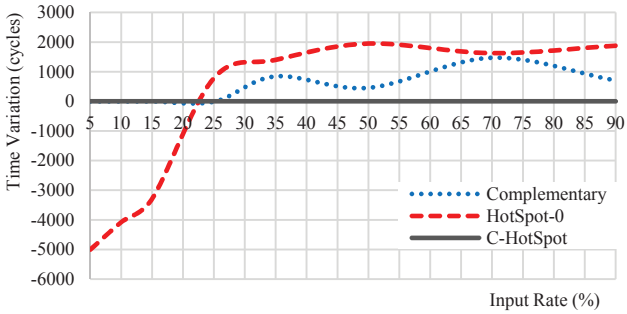

Figure 9 – Packet latency variation with and without the effects of applying QoSMCD for the three traffic distributions.

V. CONCLUSION

This work proposed QoSMDC, a software and non-hardware-intrusive solution for avoiding the saturation traffic issue. The main idea is to control the rate of injecting packets into the network on a software-level basis based on the network information collected by transmitting timestamps in packets and calculating network latencies. QoSMDC is a flexible mechanism, which could be used by different NoC structures, mainly because it does not need any modification of the hardware to be implemented. The solution proposed here was never presented by any other work. When applying a scheduling mechanism with QoSMCD to application tasks in three synthetic traffic scenarios, the experimental results presented a decrease in latency of up to 90% for an extreme traffic distribution such as a hotspot for a node and 16% for a less aggressive traffic, such as a hotspot for a cluster.

REFERENCES

[1] L. Benini, G. De Micheli. **Networks on Chips: A New SoC Paradigm.Computer**, v.35, n.1, pp. 70-78, 2002.

[2] U. Ogras, R. Marculescu. **Prediction-based Flow Control for Network-on-Chip Traffic.** ACM/IEEE Design Automation Conference, v.43, pp.839-844, 2006.

[3] I. Nousias, T. Arslan**. Wormhole Routing with Virtual Channels using Adaptive Rate Control for Network-on-Chip (NoC).** NASA/ESA Conference on Adaptive Hardware and Systems, pp.420-423, 2006.

[4] X. Wang, G. Gan, D. Fan, S. Guo. **GFFC: the Global Feedback based Flow Control in the NoC Design for Many-core Processor**. IFIP International Conference on Network and Parallel Computing, v.6, pp.227-232, 2009.

[5] M. Tang, X. Lin. **Injection Level Flow Control for Network-on-Chip (NoC)**. Journal of Information Science and Engineering, v.27, n.2, pp.527-544, 2011.

[6] M. Yuan, W. Fu, T. Chen, W. Hu, M Wu. **CABSR:Congestion Agent Based Source Routing for Network-on-Chip**. IEEE Intl Conf High Performance Computing and Communications, 2014 IEEE Intl Symp on Cyberspace Safety and Security, 2014 IEEE Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), v. 1, 6, 11, pp. 669-676, 2014.

[7] F. Moraes, N. Calazans, A. Mello, L. Möller, L. Ost. **HERMES: an infrastructure for low area overhead packet-switching networks on chip**. Integration, The VLSI Journal, v.38, issue 1, pp. 69–93, 2004.