Adaptive QoS Techniques for NoC-Based MPSoCs

Marcelo Ruaro[1], Everton A. Carara[2], Fernando G. Moraes[1]
[1] PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brazil
[2] UFSM – DELC – Av. Roraima 1000 – Santa Maria – 97105-900 – Brazil
marcelo.ruaro@acad.pucrs.br, carara@ufsm.br, fernando.moraes@pucrs.br

Abstract — **With the significant increase in the number of processing elements in NoC-Based MPSoCs, communication becomes, increasingly, a critical resource for performance gains and QoS guarantees. The main gap observed in the NoC-Based MPSoCs literature is the runtime adaptive techniques to meet QoS. In the absence of such techniques, the system user must statically define the resource distribution to each real-time task. The goal of this research is to investigate the runtime adaptation of the NoC resources, according to the QoS requirements of each application running in the MPSoC. The adaptive techniques presented in this work focused in adaptive routing, flow priorities, and switching mode. The monitoring and adaptation management is performed at the operating system level, ensuring QoS to the monitored applications. Monitoring and QoS adaptation were implemented in software. In the experiments, applications with latency and throughput deadlines run concurrently with best-effort applications. Results with real applications reduced in average 60% the number of latency violations, ensuring smaller jitter and higher throughput. The execution time of applications is not penalized applying the proposed QoS adaptation methods.**

Keywords— MPSoC; NoC; QoS; Adaptability

I. INTRODUCTION

With the increased complexity of NoC-based MPSoCs, in terms of processing elements (PEs), more applications can run simultaneously on such systems, requiring management techniques able to meet the applications' constraints. In general purpose MPSoCs, as [1] and [2], applications may start their execution at any moment, characterizing a dynamic workload behavior. In addition, the resource sharing among the running applications may lead to performance degradation. Thus, a major challenge in the design of such systems is to ensure Quality of Service (QoS), without performance degradation after a certain period of execution when several applications were inserted and removed from the system.

The communication infrastructure strongly influences the QoS. The higher the number of PEs, higher is the number of simultaneous transactions among PEs. To meet requirements of real-time applications, in a dynamic workload scenario, applications must include in their specifications the QoS constraints (as throughput or latency). Such constraints are monitored, and the system acts over the communication infrastructure, adapting it to reach the constraints.

The architectural features of the NoC router directly affect the ability to offer QoS. Factors such as arbitration, routing algorithm, buffer depth, flow control, switching mode, virtual channels and the number of physical channels per port, are widely researched and optimized to provide some level of QoS [3][4]. However, most proposals evaluated in the related work Section are restricted to the physical infrastructure. Such restriction leaves a gap to be exploited in the context of software management and NoC adaptation.

This is the motivation of the present work. Starting from a NoC with duplicated physical channels, adaptive routing, support to flow priorities and simultaneous packet switching (*PS*) and circuit switching (*CS*), the objective of this work is to develop a software monitoring scheme to evaluate the NoC performance, and two runtime adaptive techniques: *(i)* flow priority adaptation; *(ii)* establishment/ release of CS. The developed monitoring scheme targets an MPSoC hierarchical architecture [5].

The monitoring is the trigger for the adaptation. The runtime changing of the communication priority as well as the switching mode increases the support to QoS. The dynamic changes of the communication priority together with the routing algorithm, allows the avoidance of congested areas, and better network traffic distribution. In addition, CS guarantees the reservation of a given path, allowing communication to occur with maximum throughput, without the interference of other flows.

The paper is organized as follows. Section II reviews the state-of-the-art. Section III presents an overview of the proposed adaptive techniques. Section IV and V corresponds to the original contribution of this paper, the monitoring technique and the adaptive runtime techniques, respectively. Section VI presents the results, and Section VII concludes the paper, pointing out directions for future works.

II. RELATED WORK

This Section reviews recent works that provide some level of QoS in NoC-based systems. Some of the reviewed works offers runtime adaptive techniques to meet QoS. The QoS management can be controlled locally (at the router or PE level), hierarchically using a clustered approach, or centralized. It is important to observe how routers are interconnected. When single bi-directional links are used, VCs (virtual channels) are commonly adopted to ensure QoS. Some recent works adopts multiple physical channels as an alternative to VCs.

Wang el al. [3] propose a router able to meet QoS constraints through arbitration with dynamic congestion control and adaptive routing. The Authors adopt a NoC with 64-bit bi-directional links, where each router has 2 horizontal links, 4 vertical links, and 4 diagonal links. VCs are not used in the adopted NoC. For the dynamic control of QoS it is used an algorithm that evaluates the congestion at input and output ports. Packets coming from hotspots have high priority to be arbitrated, reducing congestion. However, this congestion control is restricted to the router, where local decisions are made, which may not necessarily eliminate congestion, but just to transfer it to another point.

Salah and Tourki [6] propose a router architecture for real-time applications, using PS, QoS support and a priority-based flit scheduler for BE and GS flows. The scheduler evaluates the deadlines of the incoming flows, selecting VCs according to the flow class, BE or GS. According to the Authors, results show that their router achieves an optimal packets scheduling, increasing channel utilization and throughput, reduction of the network latency, and avoidance of resources conflicts.

Fan et al. [4] propose a router combining VCs with duplicated physical channels. Each physical channel creates a *subnet*, with a dynamic number of VCs. The allocation of VCs is executed at

978-1-4799-1191-2/13/$31.00 ©2013 IEEE

runtime, by an arbiter, reserving VCs and dynamically allocate then at the demand of each port. Each subnet uses a different routing algorithm: XY and YX. The NI chooses which subnet a given flow will use. Once on a subnet, a packet cannot change the subnet. The VCs for XY only accept packets routed according to the XY algorithm, and vice versa.

Winter and Fettweis [7] present and evaluate different implementations of a central hardware unit, name *NoCManager*, which allocates at run-time guaranteed service VCs providing QoS in packet-switched NoCs. The Authors argue that the central *NoCManager* is superior to the distributed technique. Besides this conclusion, the Authors mention scalability issues, and point out a hierarchical method as future work.

Motakis et al. [8] explores the management of the NoC services to adapt the hardware resources through techniques implemented in software, allowing the user to explore the different network services through an abstract API. The implementation is based on the *Spidergon* STNoC platform. Lower-level layers are explored, along with a library of functions named *libstnoc* accessible to the user. The work focuses on dynamic reconfiguration of QoS services through the *libstnoc*. The designer can access information services (energy management, routing, QoS and security), and also enable and change these parameters through memory-mapped registers. The API can also perform a diagnostic service of the traffic of the NoC, changing QoS parameters at run time based in constraints defined by the user.

Cui et al. [9] propose a decentralized heuristic for task mapping. The proposal adopts a cluster-based method, implemented using the Tera Scale platform NoC. The clusters do not have a fixed size. Their size can change at runtime according to the characteristics of each application, and might contain more than one application. The local managers control the cluster resizing in a decentralized fashion.

Liao and Srikanthan [10] explore QoS through a hierarchical structure, dividing the MPSoC in clusters. Each cluster contains one application and a cluster manager. The system also has a global manager responsible for high-level tasks. The goal of the clustering heuristic, implemented in software, is to ensure at runtime the isolation of the traffic between different applications, favoring composability, an important feature for QoS.

Table 1 compares the reviewed works. It is noticeable that in many works the QoS management (2nd column of the Table) is made at the router level. Such approach takes local decisions, which may be inefficient at the system level. On the other, a centralized approach [7] has a global view of the system, but scalability is sacrificed. In [8], a scheme of runtime QoS adaptation is implemented at the PE level, as in the present work, where the user informs the constraints. However, that proposal does not solve the scalability issue, and the work is tightly dependent to the *Spidergon STNoC*. A trade-off is achieved with a hierarchical approach, with several managers distributed in the system [9][10]. Our work may be applied to centralized or hierarchical systems. Results are presented for a centralized management, being possible to extend the architecture to a hierarchical management.

Most NoC designs interconnect routers using single bi-directional links (3rd column). An alternative to such approach, leading to good results, in terms of communication performance and QoS, is to increase the number of physical channels [3][4]. Such approach replaces VCs, with a smaller silicon cost, and may result in disjoint networks, enabling the use of priorities or simultaneous switching modes.

As can be observed in the fourth column of the Table, approaches with QoS management at the router level adopt three main techniques to meet QoS: flow priorities [6], virtual channels (VCs) [4][6][7], and circuit switching [7]. The hierarchical management [9][10] meets QoS favoring composability, i.e., applications are "isolated" in clusters, without the interference of other flows. The work herein proposed adopts flow priorities and simultaneous PS and CS, but controlled by a local or central manager.

The main gap observed in the literature is the adaptive techniques to meet QoS (5th column). In the absence of such techniques, the system user must statically define the priority and/or the switching mode of applications. Systems where QoS management is done locally may be inefficient, since flows may be sent to congested regions, moving the problem for other NoC regions. Hierarchical management is more efficient, since applications may suffer less interference from other applications. The drawback of hierarchical approaches is to find continuous regions to map applications, even if available resources exist. The runtime adaptive techniques proposed uses local monitoring. The local monitoring sends violation events to a manager, which selects the adaptive technique to meet the QoS constraints.

III. Overview of Monitoring and QoS Adaptation Modules

The communication infrastructure adopts a 2D-mesh NoC [11], composed by duplicated 16-bit physical channels, assigning high priority to channel 0 and low priority to channel 1 (high priority packets may use both channels); deterministic Hamiltonian routing [12] in channel 1 and partially adaptive Hamiltonian routing in channel 0; input buffering; credit-based flow control; simultaneous PS and CS. The PE connected to each NoC router contains: (*i*) a 32-bit Plasma processor (MIPS-like architecture); (*ii*) a local memory; (*iii*) a DMA (Direct Memory Access) module; (*iv*) a NI (Network Interface).

The MPSoC contains manager PEs and slave PEs. Manager PEs executes heuristics to control the MPSoC, task mapping, and task migration. Slave PEs run a microkernel, responsible for task communication (local and remote), and multi-task scheduling; and user tasks.

Figure 1 shows an overview of the system, considering a 4x4 MPSoC instance, split into four 2x2 clusters. Monitoring and QoS adaptation are implemented in both slave and manager microkernels.

Table 1 – State-of-the-art comparing works targeting QoS support in NoCs.

Proposal	QoS Management	Physical channel	QoS Technique	Adaptive technique
Wang (2012) [3]	Router level	2 horiz., 4 vertic., and 4 diagonal	Dynamic Arbitration and Adaptive Routing	Congestion Aware Routing Algorithm
Salah (2011) [6]	Router level	One per direction	Flow Priority Virtual Channels	Flit Scheduling
Fan (2011) [4]	Router level	Duplicated channels	Virtual Channels	Virtual channel allocation
Winter (2011) [7]	Centralized	One per direction	Virtual Channels / CS	Central NoC Manager (HW)
Motakis (2011) [8]	PE level	One per direction	Bandwidth allocation	Management API
Cui (2012) [9]	Hierarchical	One per direction	Composability	Reclustering
Liao (2011) [10]	Hierarchical	One per direction	Composability	Cluster Allocation
This Proposal	**Centralized or Hierarchical**	**Duplicated channels**	**Flow Priority / PS+CS**	**Runtime monitoring and flow adaptation according to deadlines**

Two local monitors, implemented in the slave microkernels, evaluate latency and throughput. A global monitor is implemented in the manager microkernel, responsible to evaluate the received monitored data and select the corresponding adaptive technique. The QoS adaptation, in the same way that the monitoring, is also hierarchically implemented, being composed by *QoS adaptation modules*, implemented in the slave microkernel, and *QoS control modules*, implemented in the manager microkernel.

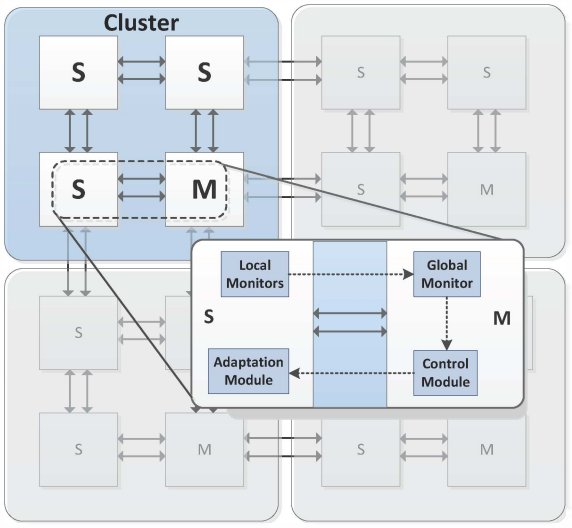


Figure 1 – Overview of the proposed monitoring and QoS adaptation modules. 'S' corresponds to slave PEs, and 'M' to manger PEs. Local/global monitors and adaptation/control modules are software implemented in each PE microkernel.

The QoS adaptation corresponds to the change in the communication flows priority and the switching mode of a given pair of communicating tasks. The change of priorities alters the physical channel used for communication, and explores the adaptability of the Hamiltonian routing algorithm. The establishment of connection changes the switching mode from PS to CS, ensuring maximum throughput to the flow.

All slave PEs ('S' in Figure 1) of a given cluster send monitored data to the cluster manager PE ('M' in Figure 1). The cluster manager PE evaluates the received data, selects the adaptive technique (flow priority or switching mode) and sends the action to be taken to all slave PEs running tasks belonging to the monitored application. Cluster managers are able to exchange the global monitored information. It is important to mention that the cluster size may vary at runtime. For example, if a given cluster has no available resources, it may request resources (PEs) to neighbor clusters, modifying the cluster shape at runtime. For this reason, the communication between cluster managers is necessary.

A. *Profile Configuration*

Monitoring and QoS adaptation require the definition of the application profile. The application developer must execute the target application in the MPSoC, without disturbing traffic, measuring throughput and latency values. These results correspond to the best results the application can achieve in the platform. The throughput and latency values to be respected are defined as deadlines, and must have their values smaller than the best results, to be possible to execute the target application concurrently with other applications.

After the deadline acquisition the user can set these values, in both microkernels, through system calls to enable the monitoring and adaptive QoS techniques. The monitoring is executed at the task level, i.e., each pair of communicating task can be monitored.

IV. MONITORING

This Section details the monitors (local and global). The monitoring process was designed to be generic, adaptable to other MPSoCs and other adaptive techniques, as DVFS, scheduling, task migration. The monitors handle *violation* and *events*:

- ***Violations***: handled by the local monitors, corresponds to a "fine grain" treatment of the monitoring information. Violations are created when a latency or throughput deadline violation occurs. The local monitors store the number of violations, and when a parameterizable number of violations is reached, a message is sent to the global monitor. This message corresponds to an *event*.
- ***Events***: handled by the global monitors, corresponds to a "coarse grain" treatment of the monitoring information. The *event* fires the execution of a heuristic, which selects the appropriate adaptive technique.

The action to accumulate in local monitors a parameterizable number of violations, before sending an event, reduces the traffic induced by the monitoring process, since monitoring packets are not sent at each violation. We adopt three violations as the default threshold number of violations to both local monitors. This value safely estimates an *event*, because it can suppress random peeks of latency or throughput, while keeps a high level of confidence in the local monitors.

A. *Throughput Monitor*

The throughput monitor counts the number of received bits within the monitoring period. The monitored period can be configured in the profile phase. As each processor may execute several simultaneous tasks, each task is individually monitored. When a given packet is received, the monitor identifies the target task and increments the task throughput counter according to the packet size. When the period of the monitoring window expires, the monitor verifies for all tasks with monitoring enabled if the throughput deadline was violated, i.e., a throughput smaller than the specified. After three violations (parameterizable value), the monitor generates a throughput event to the global monitor.

B. *Latency Monitor*

As in the throughput monitor, the latency is computed for all received packets. This latency corresponds to the *task latency*, which considers the task computation and the network latency. The monitor identifies the target task, and computes the time interval between the last two received packets. Each computed latency is verified against its deadline. If the computed latency is higher than the specified latency, a latency violation occurs. Again, after three violations (parameterizable value) a message is sent to global monitor reporting a latency event.

C. *Global Monitor and Control Module*

The global monitor and the control module are software implemented in the manager PEs (MP). The function of the *global monitor* is to receive *events* from the local monitors, select an adaptive technique, and notify it to the *control* module. The *control module* evaluates the feasibility of adaptation, and if it is feasible sends a message to the adaption module of the slave PE holding the task that should execute the adaptation action.

Figure 2 details the global monitoring process. Initially, all flows start with LOW priority (using deterministic routing). When the global monitor receives a latency event for a given flow, it calls the *control module* that changes the priority to HIGH (coupled to adaptive routing). The flow may stay in HIGH priority for a parameterizable amount of time (Flow Counter timer, *FCt*). If any event is received when the flow is in HIGH priority, the switching mode is moved to CS.

Throughput events are more severe than latency events, because real-time applications must have a small jitter during packet reception. Therefore, when a throughput event is received, the global monitor calls the *control module* that changes the flow to circuit switching mode. The flow stays in CS mode during a fixed amount of time (Circuit Switching timer, *CCt*). When CCt expires, the flow returns to HIGH priority and PS, and if any violation event is received, the flow returns to CS.

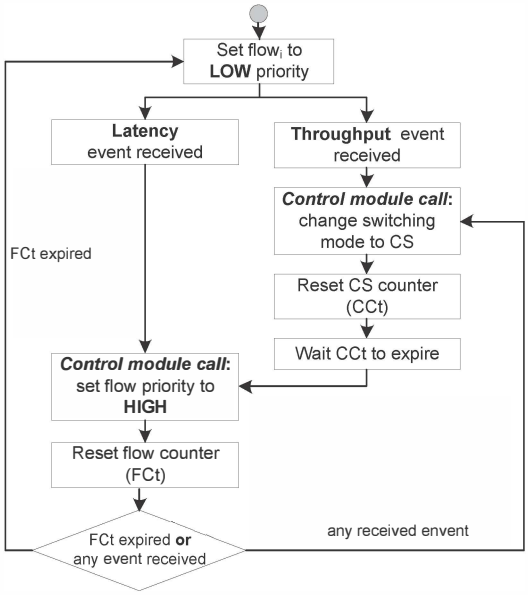


Figure 2 – Global monitoring process, and actions executed by the control module.

Two reasons justify the adoption of a fixed amount of time for a given flow to stay in CS mode: (*i*) CS reserves the entire link bandwidth, reducing the NoC resources for other flows; (*ii*) as CS does not suffer disturbing due to the exclusive link allocation, it is not possible to determine when the disturbing traffic finish. Thus, after CS the flows returns to HIGH, and if any event is received the flow goes to CS.

V. DYNAMIC QOS ADAPTATION

This Section details the dynamic QoS adaptation techniques. The *adaptation module*, located at each slave PE may modify the communication priority or the switching mode according to the message sent by the manager PE *control module*.

A. Communication Priority

As illustrated in Figure 2, all flows start using LOW priority, and deterministic routing. To change the flow priority the following sequence of actions occurs:

1. The monitored task (local monitor) sends a latency event to its global monitor;
2. The global monitor receives the message, applies the heuristic presented in Figure 2, and notifies the control module. The control module sends a message to the task originating the flow to increase its priority;
3. Receiving the adaptation message, the adaptation module execute the following actions: (*i*) identifies the target task; (*ii*) modifies the data structure responsible for control the task to send all new packets to the target task using HIGH priority (implicitly to our implementation, HIGH priority packets are transmitted using partial adaptive routing). The same task may communicate with other tasks using different priorities, since the priority is defined between a communicating task pair.

In the absence of latency or throughput events, at the end of the FCt period, the control module sends to the adaptation module of the source task a message to reduce the flow priority.

B. Circuit Switching

When the global monitor requires a connection establishment, the control module checks the feasibility to switch to CS (explained in next subsection). If the NoC can support CS, the control module sends a request to the adaptation module to the PE holding the source task (PE_{source}) to open a connection with the target task (PE_{target}). Before CS establishment, the adaptation module of the PE_{source} blocks the MPI-like primitive *Send()*, waiting the consumption of all messages generated during the PS mode. After the consumption of all PS messages, the adaptation module sends a connection establishment packet to the PE_{target}, enabling communication through CS.

The process to release a connection occurs when the global monitor identifies the timeout in the CCt timer. The control module sends a management packet to the adaptation module informing that the communication must return to PS with high priority.

1) Evaluation of the CS feasibility

The control module keeps a matrix with the state of all routers ports of the cluster. This *state matrix* is used as input for a procedure that implements the partially adaptive Hamiltonian routing algorithm. This algorithm is executed, having as input the addresses of the communicating tasks that should use CS. If the algorithm finds a path, the CS may be established. The control module updates the state matrix and sends a "change to CS" message to the producer task. If it is not possible, the control module waits new events (throughput or latency) to search again a new path. The complexity of this procedure is $\Theta(n)$, where n is the number of hops+1 between the communicating tasks.

This process ensures that all attempts to establish a connection by a given slave PE will succeed. In addition, the MP has a complete view of all CS connections inside its cluster.

VI. RESULTS

Experiments evaluate the monitoring process, priority, and CS adaptation. Results were obtained using real applications together with best-effort applications. The best-effort applications (*disturbing* applications) generate traffic that interferes with the evaluated applications, inducing deadline misses [13]. All applications are described in C language.

The MPSoC was modeled in VHDL (NoC, NI, DMA) and SystemC (processors and memory), using an RTL cycle accurate description, allowing accurate measurement of latency and throughput values. The MPSoC was simulated with Modelsim (Mentor Graphics).

Two benchmarks are used: MJPEG decoder and Dynamic Time Warping (DTW). The task graphs of such applications are presented in Figure 3. The MJPEG application has five tasks, two responsible to input and output processing (START and PRINT), and the remainder are responsible for image decoding tasks (IVLC, IQUANT and IDCT). In the DTW application, the main flow occurs between the task BANK (bank of patterns), and tasks P1, P2, P3 and P4, which recognize the sample test with the patterns through the DTW algorithm.

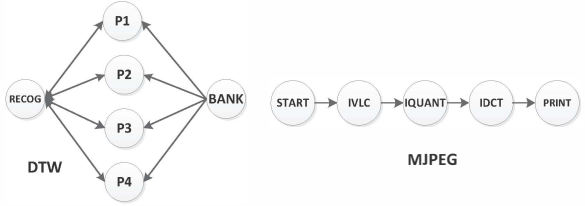


Figure 3 – Task graphs for applications DTW and MJPEG.

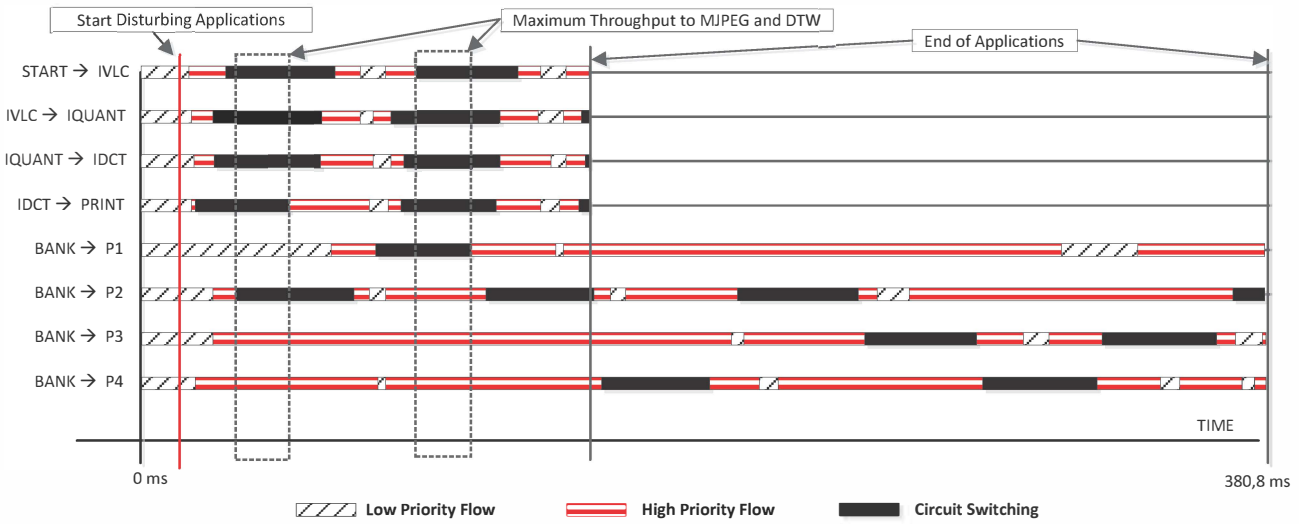


Figure 4 - Flow adaptation of MJPEG and DTW applications during execution.

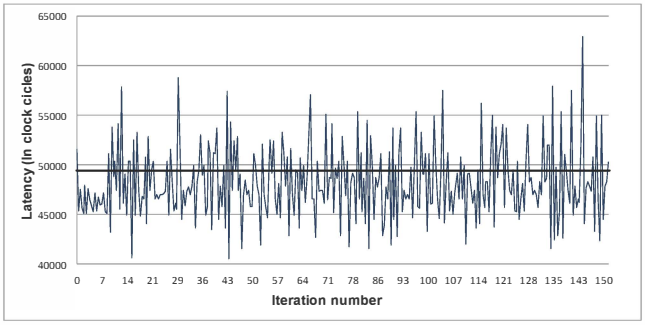
Figure 4 summarizes the communication behavior of the MJPEG and DTW applications running concurrently. Four disturbing applications run concurrently with MJPEG and DTW. Some flows are not presented for sake of simplicity in the Figure. The disturbing applications start their execution at 10 ms, and stay running throughout the simulation. The dotted rectangles highlights when the two applications use the maximum possible number of CS connections. The vertical lines signalize the end of the execution of the target applications. Note that the MJPEG application, due to disturbing traffic, stays most of the time communicating by CS, ensuring lower jitter and latency. It is also possible to observe a rotation of CS in tasks P2/P3/P4 due to this competition for the high priority channel with the PE holding the BANK task.

The task of the MJPEG application used to evaluate the results is task PRINT. The profile phase of the PRINT task set the latency deadline at 48,500 clock cycles. Figure 5 presents the latency values for the PRINT task. The dotted square rectangles in Figure 5(b) correspond to the periods where the communication IDCT→PRINT is through CS. These two CS periods can also be seen in Figure 4 (IDCT→PRINT).

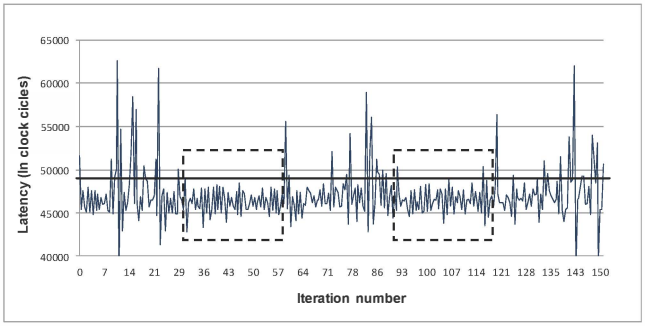
It is important to observe in the graphs of Figure 5 the number of violations and the jitter. ***Violations***: after 10 ms (moment when the disturbing traffic starts), the number of violations without QoS adaptation was 131 (Figure 5(a)). Using the QoS adaptation, such number decreases to 50 (Figure 5(b)), representing 61% of reduction. ***Jitter***: applying QoS adaption only 13 of the 50 violations have a latency superior to 10% of the deadline. This fact can be easily observed comparing both graphs. ***CS***: Observe that the latency in the highlighted dotted square rectangles is inferior to the latency deadline. This demonstrates the effectiveness of the CS to real-time. If an application has hard-real time constraints, it is possible to easily change the protocol, releasing the CS connection only at the end of the real-time application, instead using the CCt timer.

The monitored task of the DTW application is P4. It was chosen to be monitored due to the disturbing flows between tasks BANK and P4. The profile phase of the P4 task set the latency deadline at 76,000 clock cycles (value near to the one obtained during the profile step, to stress the adaptation techniques). Figure 6 presents the latency values for the P4 task. The other tasks present similar results.

Without QoS adaptation, the number of latency violations presented in Figure 6(a) is 300. Using the QoS adaptation such number decreases to 126 (Figure 6(b)) representing 58% of reduction. It is also possible to identity the periods where CS is used (highlighted by a dotted square rectangle), where only 8 latency violations were identified. Such CS periods can also be seen in Figure 4 (BANK→P4). It is important to observe in Figure 6(b) the execution interval 10ms-154ms, where most of the latency violations arise (79). The reason explaining this fact is that the flow BANK→P4 cannot use CS because it concurs with flows IQUANT→IDTC and IDTC→PRINT, which are already using CS. Such result demonstrates the limitation of the NoC to offer QoS for concurrent applications with QoS constraints. Other adaptive methods could be employed, as task migration, moving the task P4 to a position without disturbing traffic.

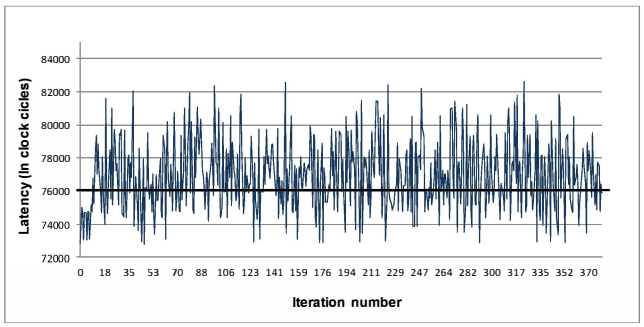


(a) Latency without adaptation

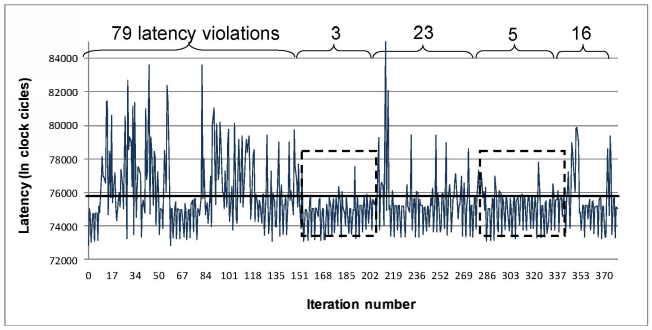


(b) Latency with QoS adaptation

Figure 5 - Latencies of the PRINT task with disturbing traffic. Latency deadline set to 48,500 clock cycles



(a) latency without adaptation



(b) latency with QoS adaptation

Figure 6 - Latencies of the P4 task with disturbing traffic.

Throughput results were similar in all scenarios (without disturbing, with disturbing, with disturbing and QoS adaptation). It was not observed a significant difference, since the tasks are computation intensive, i.e., the communication volume is small. Therefore, for such applications latency and jitter are the critical performance parameters to be monitored.

The adoption of the adaptive techniques did not affect negatively the total execution time. Without QoS adaptation, the disturbing applications increased the total execution time of the MJPEG and DTW applications in 4.0% and 2.7%, respectively. Applying QoS adaptation, the execution time overhead was 1.2% and 1.0% for MJPEG and DTW applications, respectively. Therefore, the use of the adaptive techniques almost restored the baseline execution time (without disturbing traffic).

VII. CONCLUSION

This work presented a runtime monitoring infrastructure and QoS adaptation, using priorities and connection. The generated monitoring traffic was, in average, 6% of the total traffic of the application, which can be characterized as a low intrusive monitoring traffic, compatible with the ones found in the literature [14]. Results demonstrate that the techniques do not affect the total execution time, allowing applications to meet latency and throughput constraints. In addition, such techniques are executed at runtime, adapting the application performance according to the MPSoC state.

As observed in the results, a small number of violations remain after employing QoS adaptation. A simple modification in the CS protocol for hard real time application is to disable the CCt timer. After the first CS establishment, the flow stays in CS until the end of the task execution, ensuring no interferences from flows belonging to other applications, maximum throughput, minimal latency and jitter.

Results were presented for small MPSoCs. Such sizes correspond to a cluster in larger MPSoCs. Future works comprise the extension of the present work to an MPSoC architecture using cluster-based management, ensuring scalability.

ACKNOWLEDGMENTS

Fernando Moraes is supported by CNPq project 302625/2012-7, and CAPES projects CAPES-COFECUB 708/11 and AEX 8418/13-6.

REFERENCES

[1] Tilera Corporation, “Tile-GX Processor Family”. http://www.tilera.com/products/processors/TILE-Gx_Family, 2012.

[2] Howard, J; Dighe, S.; Hoskote, Y. “A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS”. In: ISSCC, 2010, pp. 108-109.

[3] Wang, C.; Bagherzadeh, N. “Design and Evaluation of a High Throughput QoS-Aware and Congestion-Aware Router Architecture for Network-on-Chip”. In: Euromicro, 2012, pp. 457-464.

[4] Fan, W.; Xiang, L.; Hui, S. “The QoS mechanism for NoC router by dynamic virtual channel allocation and dual-net infrastructure”. In: ICCP, 2011, pp. 1-5.

[5] Fattah, M.; Daneshtalab, M.; Liljeberg, P.; Plosila, J. “Exploration of MPSoC Monitoring and Management Systems”. In: ReCoSoC, 2011, pp. 1-3.

[6] Salah, Y.; Tourki, R. “Design and FPGA Implementation of a QoS Router for NoC”. In: International Conference on Next Generation Networks and Services, 2011, pp. 84-89.

[7] Winter, M.; Fettweis, G., P. “Guaranteed Service Virtual Channel Allocation in NoCs for Run-Time Task Scheduling”. In: DATE, 2011, pp. 1-6.

[8] Motakis, A.; Kornaros, G.; Coppola, Marcello. “Dynamic Resource Management in Modern Multicore SoCs by Exposing NoC Services”. In: ReCoSoC, 2011, pp. 1-7.

[9] Cui, Y.; Zhang, W.; Yu, H. “Decentralized Agent Based Re-Clustering for Task Mapping of Tera-Scale Network-on-Chip System”. In: ISCAS, 2012, pp. 2437-2440.

[10] Liao, X.; Srikanthan, T. “A Scalable Strategy for Runtime Resource Management on NoC Based Manycore Systems”. In: ISIC, 2011, pp. 297-300.

[11] Carara, E.; Calazans, N. Moraes, F. “Differentiated Communication Services for NoC-Based MPSoCs”. IEEE Transactions on Computers, early access article, 2012.

[12] Lin, X.; McKinley, P. K.; Ni, L. M. “Deadlock-free Multicast Wormhole Routing in 2-D Mesh Multicomputers”. IEEE Trans. on Parallel and Distributed Systems, v.5(8), 1994, pp. 793-804.

[13] Tedesco, L.; Mello, A.; Giacomet, L.; Calazans, N.; Moraes, F. "Application driven traffic modeling for NoCs". In: SBCCI, 2006, pp. 62-67.

[14] Al Faruque, M.A.; Ebi, T.; Henkel, J. “AdNoC: Runtime Adaptive Network-on-Chip Architecture”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v.20(2), 2012, pp. 257 – 269.