Non-intrusive Monitoring Framework for NoC-based Many-Cores

Angelo Elias Dalzotto, Caroline da Silva Borges, Marcelo Ruaro, Fernando Gehm Moraes
PUCRS – School of Technology, Porto Alegre, Brazil
{angelo.dalzotto,marcelo.ruaro, caroline.s}@edu.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—Many-core Systems on Chip (MCSoCs) require resource management to achieve scalability at the computation and communication levels. The monitoring infrastructure feeds management tasks with raw data, enabling these tasks to detect behaviors corresponding to constraint violations or a trend that signalizes a future violation. Several works available in the literature use monitoring to apply their management techniques but do not specify how to implement the monitoring framework. We propose a monitoring framework for MCSoCs, with the following features: (i) generic: the infrastructure can carry data related to different monitored features; (*ii*) monitored data does not disturb NoC flows; and (*iii*) reduced overhead compared to other monitoring methods. The monitoring framework is loosely coupled to the MCSoC by using a dedicated NoC to carry monitoring and management messages, decoupling data traffic from management traffic. Results adopt the Observe-Decide-Act management method, comparing the proposed monitoring framework to a standard monitoring approach. Results show a reduction in the data NoC traffic (12%), faster management responsiveness to act on deadline violations (up to 77%), and reduced applications execution time (on average 8%).**

***Index Terms*—monitoring, management, self-adaptability, multiple physical networks (MPN), NoC-based many cores**

I. Introduction

The increased number of cores in Many-Core Systems on Chip (MCSoC) results in complex resource allocation problems. To keep scalability at the computation and communication levels, it is necessary to *manage* the many-core resources. Besides application admission and task mapping, system management includes actions to keep the system operating at safe conditions (e.g., safe temperature and power) and deliver to applications resources enabling them to meet their constraints (e.g., real-time deadlines). Such systemic and application constraints can conflict, making management complex.

The MCSoC management might be centralized [1] or distributed [2, 3]. *Distributed management* is the approach used to attain scalability in larger MCSoCs. An example of flexible and distributed management is the Observe-Decide-Act (ODA) control loop [4]. ODA organizes the management in specialized tasks to enable the system to achieve self-awareness and self-adaptability. These properties allow the MCSoC to configure itself based on its perception of the application constraints and system status [5].

Observation is the key feature in the ODA approach or any other strategy, enabling management techniques to guide decisions and actuations (such as DVFS, scheduling, task migration, and power gating). A *monitoring infrastructure* feeds observation tasks with raw data, enabling these tasks to detect behaviors corresponding to constraint violations or a trend that signalizes a future violation [6]. Although several works in the literature use monitoring in MCSoCs, the monitoring infrastructure is usually abstracted since these works do not describe it. This lack of details is the gap in the literature to be fulfilled.

The *goal* of this work is to propose a monitoring framework for MCSoCs, with the following features:

- generic: the infrastructure can carry data related to different monitored features, such as QoS or temperature;
- non-intrusiveness: monitored data does not disturb NoC flows;
- reduced performance penalty: minimization of hardware interrupts in processors receiving monitoring data.

The *original* contribution of this work is the proposal of a lightweight monitoring framework with hardware and software modules, enabling faster management actuation. The monitoring framework design is generic, and other NoC-based SoCs may benefit from this proposal.

This paper is organized as follows. Section II presents related work on many-core monitoring and the trend of using multiple NoCs in a many-core. Section III presents the baseline MCSoC architecture, its management organization, and the motivations that led us to this work. Section IV describes the monitoring framework, the main contribution of this work. Section V presents experimental results comparing the proposed framework with a clustered distributed monitoring technique. Finally, Section VI concludes this paper and points out directions for future works.

II. Related Work

Monitoring is the foundation of self-adaptation in many-cores, requiring distinct metrics for managing each system resource [7]. An example of early work is the Kornaros and Pnevmatikatos proposal [8]. The Authors adopt centralized management, with monitoring performed by hardware agents on the network interface. To Author's goal is to monitor real-time applications, using a hierarchical, cluster-based monitoring infrastructure.

Rahmani et al. [9] present a Dynamic Power Management (DPM) technique for avoiding dark cores. Their work use sensors and counters at the PE level, distinct constraints calculators, and a DPM controller. The Authors do not specify

how monitored data flow through the system and *where* the controller is located. Caimi et al. [10] list security mechanisms for NoC-based many-cores that rely on monitored data, such as traffic characteristics, bandwidth, and resource availability, without presenting details of how to treat and transmit the monitored data through the system. Gheibi et al. [11] apply Machine Learning (ML) in self-adaptive systems, relying on monitoring feedback loops to ensure the system goals. Note that the works [7, 9, 10, 11], despite exploiting monitoring for resource management, the monitoring itself is not the focus of the work, with the monitoring framework abstracted.

Da Silva et al. [12] propose a thermal management for MCSoCs. In their work, each Processing Element (PE) periodically monitors the type and number of executed instructions, the number of memory accesses, and the number of flits traversing the router. The PE sends this monitored data to a manager PE through the NoC. This manager computes the power dissipated by each PE, transmitting this information to a hardware temperature estimator using the NoC. The main drawback of the adopted monitoring infrastructure is using the same NoC for both data and monitoring.

Tsoutsouras et al. [13] present a Per-Application Management (PAM) organization. The Authors use hierarchical management, dividing the many-core into clusters, where each cluster has a manager responsible for monitoring its area. The management is applied by per-application managers that negotiate resources between each other.

Sudusinghe et al. [14] describe a ML-based Denial of Service (DoS) attack detection. In their work, NoC traffic data is gathered at each router and sent using a dedicated NoC to a core responsible for security management. The Authors justify the adoption of a dedicated NoC for transmitting monitoring packets due to the advantages of Multiple Physical NoCs (MPN) over Virtual Channels (VC) [15].

Recent work also adopts MPN for management. Choi et al. [16] propose a hybrid approach for heterogeneous many-core architectures targeting deep learning kernels. Their approach uses a standard NoC to communicate CPUs to GPUs, and a wireless NoC for CPU to memory controller communication. DRACON [17] is a dedicated hardware infrastructure for management, that uses: (*i*) a global management NoC for communication between managers in a cluster-based organization; (*ii*) a local management NoC that connects the cluster managers to its locally managed PEs; (*iii*) a data NoC for PE to PE data exchange. Sant'Ana et al. [18] adopt a dedicated NoC that acts as a lightweight firewall, which only the cluster-based managers can insert packets into this dedicated NoC.

Besides system management, MPN is used in deadlock-free cache coherence protocols [19, 20, 21], circuit-switching configuration in Software-Defined Networks (SDN) [22], and for path discovering in secure zones mechanisms [23].

Our proposal does not restrict the monitoring to a centralized manager as in [14], and does not enforce a hierarchical path for the monitored data as in [12, 13]. We follow the current trend of using MPN, using a broadcast NoC to carry the monitored data.

Three reasons justify the adoption of a broadcast NoC:

- isolate data traffic from monitoring traffic, increasing system security;
- faster propagation of these packets to several PEs simultaneously;
- fault-tolerance, since broadcast transmission may bypass faulty links.

The management strategy is orthogonal to our monitoring framework. According to the system constraints, it might be used by centralized managers, Cluster-Based Management (CBM), or hardware-accelerated peripherals.

III. Architecture and Motivation

We assume a many-core using a 2D-mesh NoC as the communication infrastructure in this work. PEs contains an NoC router, a processor, a network interface (NI), and a local scratchpad memory. PEs may execute management and user tasks. Peripherals, such as shared memories and hardware accelerators, may be connected to the borders of the NoC. At least one peripheral is required, responsible for deploying applications.

The management strategy is the ODA, where Observation, Decision, and Actuation tasks execute in user space at any PE. This method does not require dedicated management processors, as in [1, 2, 3], improving system utilization. All PEs execute the same operating system (OS), which schedules ODA and user tasks. Figure 1 shows a 6x6 MCSoC instance with four Observation tasks, three Decision tasks, and two Actuation tasks. These tasks are not restricted to a given location and may be allocated according to the application constraints. For example, if an application with QoS constraints enters the system, a new Observation task may be mapped near to this application to receive QoS monitoring data.

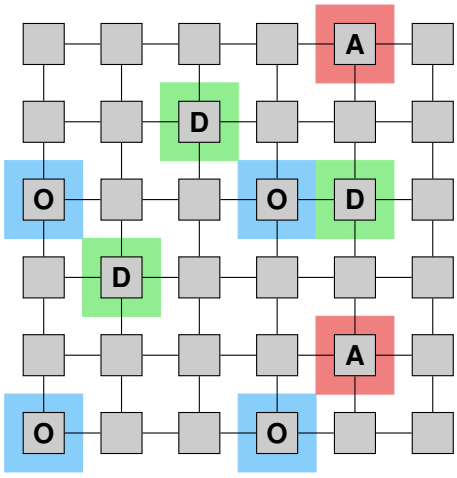

Fig. 1. ODA model. **O** – Observation, **D** – Decision, **A** – Actuation tasks.

The ODA method enhances management parallelism and flexibility but increases the number of messages flowing through the NoC. Table I presents the number of messages and flits comparing ODA to a CBM approach in a 6x6

many-core, modeled at the RTL level. The CBM management has four 3x3 clusters with four manager PEs. The ODA management contains 4 Observers, 4 Deciders, and 1 Actuator. The workload comprises four applications: Encryption, Audio and Video decoding, Dijkstra's algorithm for a graph shortest path, and Pattern Recognition.

TABLE I
NUMBER OF MESSAGES AND FLITS IN A CBM AND ODA APPROCHES.

	CBM	ODA	ODA overhead
Number of messages	2,137	3,623	70%
Number of flits	163,007	233,089	43%

In Table I, despite increasing the number of messages by 70%, the total number of flits propagating through the NoC increases by 43%. Despite the traffic increase, ODA detected more violations than CBM and consequently acted in applications to meet deadlines due to the parallelization of the management tasks.

Although the number of messages increases, the volume of flits does not increase in the same proportion. Thus, monitoring and management messages are small, and adopting an additional NoC could be arguable. However, we justify the adoption of an additional NoC for monitoring for the following reasons:

A. Mixing applications and monitoring traffic potentially affect application performance and monitoring security. Monitored data can carry sensitive system information, such as DVFS levels, and a malicious application could damage the system by generating false DVFS monitoring packets [24].
B. General-purpose NoCs usually adopt point-to-point communication (unicast). Management may require multicast transmission. For example, to send a message for all tasks of a given application to modify voltage-frequency levels (DVFS).
C. Monitoring and management messages require transmission with low latency. NoC congestion can delay the actuation reaction time, leading, for example, to missed deadlines in applications having real-time constraints.
D. Data packets interrupt processors upon the reception of a new message. As monitoring messages are frequent and small, processors that receive them have their performance degraded due to the constant hardware interrupts.

IV. MONITORING FRAMEWORK PROPOSAL

Three subsections compose this Section. First, we briefly describe the dedicated NoC, then the PE architecture, and finally the monitoring framework.

A. Dedicated NoC for Monitoring and Management Messages

The dedicated control NoC uses as reference a state-of-the-art broadcast NoC [25], named $brNoC$. $brNoC$ transmits small management and monitoring messages in a single flit, with its size equal to 80 bits, where 40 bits corresponds to the payload (it is possible to parameterize the flit size at design time). Data transmission using this NoC presents low latency and fault tolerance due to the broadcast transmission, equivalent to flooding behavior, suited for management purposes.

Figure 2 details the $brNoC$ architecture used as base for this implementation. Its topology follows the same 2D-mesh used by the data NoC, with North, South, East, West, and Local ports. The $brNoC$ modules are: (*i*) an Input Arbiter and Input Finite-State Machine (I-FSM); (*ii*) a central Content-Addressable Memory (CAM); and *(iii)* an Output Arbiter and Output Finite-State Machine (O-FSM). The $brNoC$ Input and Output logic are independent. A round-robin Output Arbiter selects a CAM line to propagate to the outputs (broadcast). The O-FSM searches the *pending* field for messages that need to be sent. sa except to the one where it came from. To write a message to the CAM, the I-FSM must assert that the data is not in the memory and it has available space, marked by the *used* field.

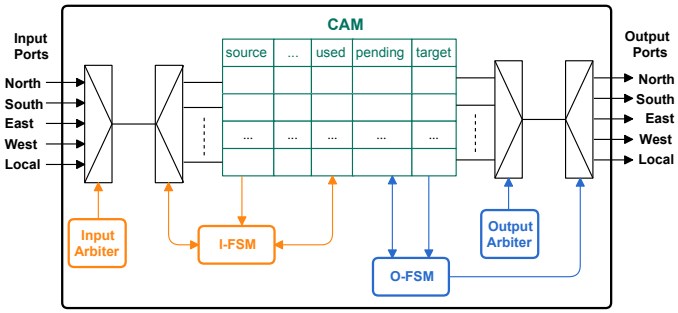

Fig. 2. BrNoC architecture. Source: [25].

The most relevant $brNoC$ feature is that all messages fit in one flit. The advantages of 1-flit messages are: (*i*) no input buffers on local ports; (*ii*) simplified switching mode, which enables the broadcast; and (*iii*) smaller router silicon area, corresponding to 50% of the data router.

The absence of input buffers does not mean that flits go directly through the $brNoC$ router, since this would be unfeasible in terms of delay. The flits that arrive at the router are stored on a CAM line and later transmitted to the output ports. The average propagation latency of the one flit packet is 7 clock cycles per hop. The data NoC takes at least 6 clock cycles per hop for arbitration and routing, plus the time needed to transmit the payload. Besides faster propagation to several targets simultaneously, another advantage of broadcast propagation is that it reaches the target routers even if there is congestion in the $brNoC$, through a non-minimum path.

The $brNoC$ can carry out the transmission of three broadcast modes:

1) `ALL`: broadcasts a message to **all** processors;
2) `TARGET`: the message arrives at all PEs, but only a given PE, the **target**, effectively receives the message;
3) `CLEAR`: broadcast started by the $brNoC$ after a `ALL` or `TARGET` broadcast message with the goal to release the CAM line associated to this message.

Software *services* use these broadcast modes. For example, to identify a given monitored data (e.g., power, performance,

latency, deadline violations), to notify a PE to make an actuation (e.g., task migration, change on DVFS level), or a management information (e.g., start/end of applications).

Due to the $brNoC$ payload size limitation, management messages with larger payloads use the data NoC. Examples of such messages include task code, data, task context, and stack content in allocation or migration services.

The $brNoC$ meets the first characteristic that the monitoring framework must provide: be generic. The $brNoC$ can carry data related to different monitored features.

B. Processing Element with brNoC

Figure 3 presents the PE architecture. The Direct Memory Network Interface (DMNI [26]) is a Network Interface (NI) with Direct Memory Access (DMA) capabilities. The processor and the DMNI are connected to a dual-port local memory. The DMNI simplifies the OS drivers and, at the same time, improves performance when sending/receiving packets to/from the NoCs. Observe that both routers are independent and connected to a DMNI port. The goal is to propose a generic method, decoupling monitoring and data, easing the integration of the $brNoC$ in other designs.

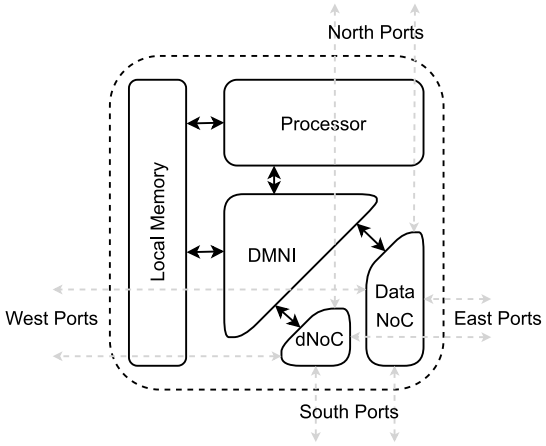

Fig. 3. Proposed PE architecture. dNoC corresponds to the $brNoC$ router.

The feature of 1-flit messages simplifies the integration of the $brNoC$ router to the DMNI module. The absence of logic to control the packet flow and size minimizes the hardware required for this integration.

This PE architecture meets the second characteristic that the monitoring framework must provide: non-intrusiveness. The monitoring messages are isolated in a second NoC.

C. Hardware and Software Infrastructure

Figure 4 details the modified DMNI architecture to receive monitoring and management messages from the $brNoC$. The service embodied in a message arriving in the $brNoC$ local port defines how the DMNI sends it to the local memory:

1) Monitoring messages: the DMNI computes the address to write the message into the Observer task memory space without interrupting the processor.
2) Control messages: treatment similar to the reception of data packets, i.e., the DMNI raises an interrupt to the processor upon data reception. A buffer stores the control messages ($brNoC$ buffer in Figure 4) to avoid $brNoC$ stalls due to the time spent interrupting the processor and executing message reception.

The DMNI arbiter was modified to prioritize $brNoC$, because broadcasts only advance when all propagated ports acknowledge the reception of a given message. Therefore, they should occur within the minimum delay to avoid performance loss. This priority does not disturb the reception of messages coming from Data NoC because broadcast messages contain a single flit, and Data NoC reception is usually buffered and tuned for throughput, not low latency.

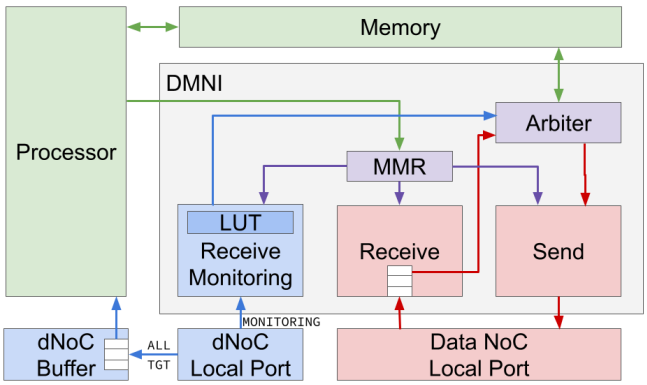

Fig. 4. DMNI block, and integration with $brNoC$.

The Observer task has a *monitoring table* in its memory space. Each monitoring table line has two cells: (*i*) task ID that generated the monitored data; (*ii*) monitoring data. The Observer task periodically evaluates the received data (e.g., number of violated deadlines), transmitting this information to a Decision task.

The processor sets up monitoring message reception through $brNoC$ by using Memory Mapped Registers (MMR). A monitoring Application Programming Interface (API) is responsible for providing system calls to Observer tasks that are capable of configuring MMRs with the base address of the monitoring table (MT_{BA}). Each monitored metric should have its own MT_{BA} configured. This configuration occurs during the Observer task startup.

When receiving a monitoring packet from a given task running on a PE, we must verify if this task has already written monitoring data in the monitoring table. To avoid unnecessary memory accesses, the DMNI received a LUT (look-up table) indexed by the tuple {task ID, PE address}. The LUT address is used as an offset to be added to the MT_{BA} when there is a match between the received {task ID, PE address} with the one stored in the LUT. If the tuple is not available in the LUT, it is written in the first position indexed by the PE. When a task finishes its execution, the OS notifies this event by broadcast, causing all DMNIs with the {task ID, PE address} entry to invalidate the position in the LUT.

After computing the monitoring line address, the DMNI writes it into the monitoring table. Note that monitoring messages can be overwritten without compromising the observation. Therefore there is no need to verify if the Observer task consumed the previous data before writing it into the table. For example, consider that the Observer task is monitoring the temperature. The temperatures sent by PEs have slight differences between successive monitoring messages, being the behavior of the thermal trend observed after the reception of several monitoring messages.

The previous discussion presented the reception of a monitoring or control message through $brNoC$. The processor does not need to use DMA to send a control message through $brNoC$. Instead, it directly injects the single flit message into the $brNoC$ using MMRs.

As the MCSoC may have several Observer tasks, each of these tasks broadcasts its capabilities. The monitoring software stack receives this broadcast and configures, in each PE, the nearest Observer for each announced capability.

At the task level, the monitoring messages are generated by Low-Level Monitors (LLM), responsible for fetching the metrics being monitored without complex computation. The OS invokes the LLMs periodically, collecting data from, e.g., instruction counters for power and temperature estimation or QoS monitors. By joining the LLMs with the proposed monitoring framework, it is possible to either send a monitoring message to the nearest Observer task or send it to all Observers.

The hardware and software infrastructure meets the third characteristic that the monitoring framework must provide: reduced performance penalty. Writing monitored data directly into the observer task memory space avoids the time spent handling interrupts, executing context saving, and running the packet reception driver.

V. Results

The proposed monitoring framework is evaluated using the following metrics:

1) *Communication volume*: the number of transmitted flits through both Data NoC and $brNoC$;
2) *Management responsiveness*: latency between management actuations;
3) *Applications performance*: the execution time of applications running in the many-core.

A. Experimental setup

- MCSoC size: 7x7.
- Applications: Encryption (9 tasks), A/V Decoding (7 tasks), Dijkstra's graph shortest path algorithm (7 tasks), and Pattern Recognition (9 tasks). Each application has a distinct parallel model, such as pipeline or master-slave.
- ODA management with two distinct monitoring approaches: (*i*) a ***clustered*** monitoring with four 3x3 virtual clusters, without the proposed monitoring framework; and (*ii*) the ***proposed*** monitoring framework as described in Section IV.

The ODA management has the following tasks:

- **4 Observers**: capable of monitoring RT statistics of executing tasks. Each PE sends these metrics of all its executing tasks to its nearest Observer;
- **4 Deciders**: each Observer sends its data to its nearest Decider, which knows the RT constraints of the executing application, decides whether or not to perform a task migration to avoid or stop violating these constraints.
- **1 Actuator**: capable of negotiating task migrations with the OS executing in each PE upon requests coming from the Deciders.

The RT tasks monitoring window is set to 500 μs. This means that all RT tasks generate a monitoring message at every 500 μs. This period is smaller than the default scheduling timeslice for RT OS, such as FreeRTOS, providing enough actuation triggers even with short simulation times.

Each application has most of its tasks statically mapped to the same PE to simulate high resource usage and induce deadline violations to trigger task migration, which is used as the actuation mechanism.

B. Communication Volume

Figure 5 presents the communication volume with the two monitoring approaches, executing the complete set of applications. The transmitted flits through Data NoC, in blue, are reduced by 12% in the proposed approach. The number of messages transmitted by $brNoC$, in yellow, is only 2% of the transmitted flits through Data NoC in the proposed approach, evidencing the efficiency of $brNoC$ for monitoring and management purposes.

Note that Data NoC is still used for management messages with payloads larger than the $brNoC$ flit size. In Figure 5, the reduction in flits transmitted through Data NoC could translate to better energy efficiency due to the lower complexity of the $brNoC$ compared to the Data NoC.

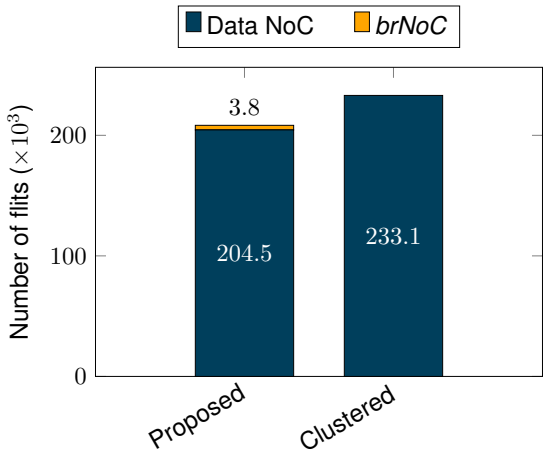

Fig. 5. Communication volume in the proposed approach versus clustered monitoring. A $brNoC$ flit (80 bits) is normalized to the same size as the Data NoC flit (32 bits), i.e., the number of brNoC flits is multiplied by 2.5.

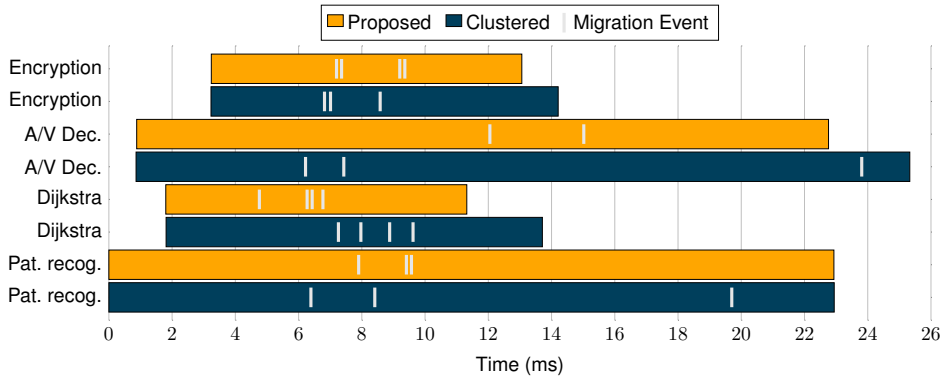

Fig. 6. Management responsiveness for the proposed approach and clustered monitoring scenarios.

C. Management Responsiveness

Figure 6 evaluates the management responsiveness for each application. The x-axis represents the simulation time, and the bars on the y-axis represent the applications execution lifetime, evaluated for two scenarios: (*i*) the proposed approach in yellow; and (*ii*) the clustered monitoring approach in blue. Additionally, white columns inside the bars represent a task migration event triggered by the Decision task. The average task migration time is 140 μs, with the hop distance having a negligible impact on the migration time.

Applications Encryption, A/V Decoding, and Pattern Recognition start migrating earlier with the clustered monitoring. This may seem that the proposed approach reacts later. Actually, we observed that deadline violations are delayed as a consequence of two factors: (*i*) reduction of monitoring and management traffic volume in the Data NoC, resulting in greater communication availability; (*ii*) faster communication protocol due to the use of $brNoC$ for the communication API of applications.

The migration actuation of tasks violating deadlines occurs in bursts for Encryption, Dijkstra, and Pattern Recognition applications. The parallel execution of the ODA tasks enables faster actuation, reducing the average minimum latency between actuations by 77%.

Summarizing Figure 6, the proposed monitoring framework reduces the interference of monitoring with data traffic and provides faster management responsiveness. This can be observed by migration actuations occurring later than in the clustered monitoring and in moments closer to each other.

D. Applications performance

Another data revealed in Figure 6 is the applications execution time reduction. On average, the application execution time reduces by 8%. When a deadline miss occurs, the Decision task migrates tasks that share a processor, increasing execution parallelism, thus reducing its execution time.

The faster management responsiveness, which induces execution time reduction, is promoted by:

- faster monitoring supported by $brNoC$ (low latency and DMA);
- reduced hardware interrupts. The four PEs holding Observer tasks reduced on average 45% the number of hardware interrupts using the monitoring framework compared to the clustered monitoring.

These two factors enhanced the overall performance of the Observer tasks.

VI. Conclusion and Future Work

This paper presented a monitoring framework using a dedicated NoC for management purposes. We compared the monitoring framework integrated with an ODA management to a clustered monitoring. Results showed faster management reactivity (77%), with burst actuations enabled by parallel monitoring and lower performance overhead. The features of the monitoring framework that enabled this improvement are: (*i*) smaller communication volume in the Data NoC, reducing NoC congestion; (*ii*) use of the dedicated NoC for management messages, such as monitoring, enhancing its handling and delivery latency; and (*iii*) DMA usage to transfer monitoring data directly to the observer tasks, reducing the overhead due to the packet handling.

Other identified advantages of using the dedicated broadcast NoC are:

- A separate network for management messages reduces the interference in user application data traffic, enhancing performance;
- The management messages flow in a separate network to which user tasks do not have access, enhancing system security;
- Broadcast messages follow a flood behavior that is fault-tolerant, thus desired for management purposes;
- The treatment performance of management messages is enhanced due to their small size;
- Provides a separate flow to monitoring messages through DMNI, reducing processor interrupts.

Future work includes using the proposed monitoring framework in multi-objective management, and evaluating power-performance-area of the PE in the proposed approach, since only the $brNoC$ area was evaluated in this work.

ACKNOWLEDGMENT

This work was financed in part by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), grant 309605/2020-2; and CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), Finance Code 001.

REFERENCES

[1] X. Huang, X. Wang, Y. Jiang, A. K. Singh, and M. Yang, "Dynamic Allocation/Reallocation of Dark Cores in Many-Core Systems for Improved System Performance," *IEEE Access*, vol. 8, pp. 165 693–165 707, 2020.

[2] H. Wang, J. Ma, S. X. Tan, C. Zhang, H. Tang, K. Huang, and Z. Zhang, "Hierarchical dynamic thermal management method for high-performance many-core microprocessors," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, pp. 1–21, 2016.

[3] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel, "DistRM: Distributed resource management for on-chip many-core systems," in *CODES+ISSS*, 2011, pp. 119–128.

[4] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "A generalized software framework for accurate and efficient management of performance goals," in *EMSOFT*, 2013, pp. 1–10.

[5] B. H. C. Cheng, H. Giese, P. Inverardi, J. Magee, R. de Lemos, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, "Software Engineering for Self-Adaptive Systems: A Research Roadmap," in *Proceedings of the Dagstuhl Seminar*, 2008.

[6] N. Dutt, A. Jantsch, S. Sarma, and others, "Self-Aware Cyber-Physical Systems-on-Chip," in *ICCAD*, 2015, pp. 46–50.

[7] K. Moazzemi, A. Kanduri, D. Juhász, A. Miele, A. M. Rahmani, P. Liljeberg, A. Jantsch, and N. Dutt, "Trends in On-chip Dynamic Resource Management," in *DSD*, 2018, pp. 62–69.

[8] G. Kornaros and D. N. Pnevmatikatos, "Real-Time Monitoring of Multicore SoCs Through Specialized Hardware Agents on NoC Network Interfaces," in *IPDPSW*, 2012, pp. 248–255.

[9] A.-M. Rahmani, M.-H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen, "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *ISLPED*, 2015, pp. 219–224.

[10] L. L. Caimi, R. Faccenda, and F. G. Moraes, "A Survey on Security Mechanisms for NoC-based Many-Core SoCs," *Journal of Integrated Circuits and Systems*, vol. 16, no. 2, pp. 1–15, 2021.

[11] O. Gheibi, D. Weyns, and F. Quin, "Applying Machine Learning in Self-Adaptive Systems: A Systematic Literature Review," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 15, no. 3, pp. 1–37, 2021.

[12] A. L. da Silva, A. L. d. M. Martins, and F. G. Moraes, "Fine-Grain Temperature Monitoring for Many-Core Systems," in *SBCCI*, 2019, pp. 1–6.

[13] V. Tsoutsouras, S. Xydis, and D. Soudris, "Application-Arrival Rate Aware Distributed Run-Time Resource Management for Many-Core Computing Platforms," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 285–298, 2018.

[14] C. Sudusinghe, S. Charles, and P. Mishra, "Denial-of-Service Attack Detection Using Machine Learning in Network-on-Chip Architectures," in *NOCS*, 2021, pp. 35–40.

[15] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual Channels and Multiple Physical Networks: Two Alternatives to Improve NoC Performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1906–1919, 2013.

[16] W. Choi, K. Duraisamy, R. G. Kim, J. R. Doppa, P. P. Pande, R. Marculescu, and D. Marculescu, "Hybrid Network-on-Chip Architectures for Accelerating Deep Learning Kernels on Heterogeneous Manycore Platforms," in *CASES*, 2016, pp. 1–10.

[17] D. Gregorek, J. Rust, and A. Garcia-Ortiz, "DRACON: A Dedicated Hardware Infrastructure for Scalable Run-Time Management on Many-Core Systems," *IEEE Access*, vol. 7, pp. 121 931–121 948, 2019.

[18] A. C. Sant'Ana, H. M. Medina, K. B. Fiorentin, and F. G. Moraes, "Lightweight Security Mechanisms for MPSoCs," in *SBCCI*, 2019, pp. 1–6.

[19] J. Balkind, K. Lim, F. Gao, J. Tu, D. Wentzlaff, M. Schaffner, F. Zaruba, and L. Benini, "OpenPiton+ Ariane: The First Open-Source, SMP Linux-booting RISC-V System Scaling From One to Many Cores," in *CARRV*, 2019, pp. 1–6.

[20] D. Giri, P. Mantovani, and L. P. Carloni, "NoC-Based Support of Heterogeneous Cache-Coherence Models for Accelerators," in *NOCS*, 2018, pp. 1–8.

[21] J. L. Abellán, E. Padierna, A. Ros, and M. E. Acacio, "Photonic-based express coherence notifications for many-core CMPs," *Journal of Parallel and Distributed Computing*, vol. 113, pp. 179–194, 2018.

[22] K. Berestizshevsky, G. Even, Y. Fais, and J. Ostrometzky, "SDNoC: Software defined network on a chip," *Microprocessors and Microsystems*, vol. 50, pp. 138–153, 2017.

[23] M. M. Real, P. Wehner, V. Lapotre, D. Göhringer, and G. Gogniat, "Application Deployment Strategies for Spatial Isolation on Many-Core Accelerators," *ACM Transactions on Embedded Computing Systems*, vol. 17, no. 2, pp. 55:1–55:31, 2018.

[24] E. M. Benhani and L. Bossuet, "DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC," in *ICECS*, 2019, pp. 489–492.

[25] E. Wachter, L. L. Caimi, V. Fochi, D. Munhoz, and F. G. Moraes, "BrNoC: A broadcast NoC for control messages in many-core systems," *Microelectronics Journal*, vol. 68, pp. 69–77, 2017.

[26] M. Ruaro, F. B. Lazzarotto, C. A. Marcon, and F. G. Moraes, "DMNI: A Specialized Network Interface for NoC-based MPSoCs," in *ISCAS*, 2016, pp. 1202–1205.