Differentiation of MPSoCs Message Classes Using Multiple NoCs

Douglas R. G. Silva, Fernando G. Moraes
FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
douglas.roberto@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—MPSoCs using a distributed memory architecture generates a large volume of messages that may be classified in application messages, as defined by the application developer, and management messages, used to ensure the correct operation of the platform. Both messages classes normally use the same communication infrastructure. Thus, the application traffic can be adversely impacted by the management traffic. Several works observe that different messages classes can be distributed into multiple NoCs, improving the performance and power consumption of the platform. However, these works mainly target shared memory systems. This work suggests the utilization of multiple NoCs in an MPSoC using distributed memory architecture, specializing each network for different message classes. An improvement of up to 40% in the application messages jitter and an average improvement of 5% in the application execution time can be achieved using this strategy.**

Keywords—MPSoC; Network-on-chip (NoC); Multiple networks; Distributed Memory

I. Introduction

Multi-Processor Systems-on-Chip (MPSoCs) are becoming prevalent in embedded systems, supporting a large number of application classes. The memory architecture is an important design consideration in MPSoC platforms. Most platforms features one of these possible memory architectures [1]: *(i)* shared memory architectures using on-chip cache; *(ii)* distributed memory architecture using a scratchpad memory cache [2], where each processor has its private memory and cannot interact directly with another processor memory.

Distributed memory systems require an explicit communication mechanism between the processors. The Message Passing Interface (MPI) is a popular standard with a set of software primitives used to exchange data between processors. Several distributed memory MPSoCs uses a simplified version of this communication library [1][3][4].

The messages in MPSoC platforms may be classified into two categories: application and management. Application messages are related to the application, which includes all the communication defined by the application developer. Management messages are related to the control of the MPSoC resources, including performance monitoring, task mapping, fault tolerance, power management, security, and many other aspects which are not directly related to the application computation, but have an important role in the execution of the platform.

Several proposals explore the use of multiple parallel physical NoCs, where each network is specialized for different traffic classes. Some works [5][6][7] evaluate the performance, area and energy efficiency of different network characteristics and topologies, including in their evaluation the use of multiple networks. These works model the traffic as a cache-coherence communication protocol. The cache coherence traffic can be divided into multiple subclasses, such as Read Requests/Replies, Write Requests/Replies, among others. The cache-coherence communication may take advantage of multiple networks [8][9][10][11] to optimize energy efficiency without degrading the performance, distributing the traffic subclasses into the available networks. Each network is specialized for the characteristics or performance requirements of each traffic subclass.

An additional network can also be used to transfer communication not directly related to the application data. Ciordas et al. [12] proposes a network targeting monitoring traffic, parallel to the network used for the remaining chip activity. Sepulveda et al. [13] propose the use of an additional network to isolate security control traffic from the remaining MPSoC traffic, where this network is accessible only to trustworthy IPs. This methodology allows security zones are established in the MPSoC, avoiding the exposure of sensible data to the remaining components.

Some platforms [14][15][16] that employ several traffic classes count on the usage of multiple networks, specializing each network for a specific class, providing isolation and prioritization of the traffic.

The main differentiation of the proposed work to the state-of-the-art includes: (*i*) memory organization, targeting distributed memory architecture; (*ii*) support for a large amount of management services (not specialized for a given set of services, as cache coherence); (*iii*) evaluation conducted using a RTL circuit description, instead of using platform simulators.

This work suggests the separation of the management messages from the application messages at the network level using two networks, one dedicated for the application traffic and the other dedicated for the management traffic. This work also explores two different management network strategies according to the platform traffic characteristics and evaluates the trade-off between latency/jitter and execution time for the proposed strategies.

II. Reference Platform

The reference platform adopted in this work is an MPSoC architecture composed of several Processing Elements (PEs) interconnected by a Network on Chip (NoC), represented in Fig. 1. This NoC gives support to QoS by employing duplicated physical channels. All packets in the NoC encapsulate messages that are exchanged among the PEs for communication, using an MPI-like API. Each message in this platform has a specific service identification according to its function in the platform. The applications executed in the MPSoC are divided into tasks, which run in parallel in several PEs. This MPSoC adopts a distributed memory architecture. Each task has its private memory, and no memory is shared between the PEs.

This MPSoC uses a hierarchical management architecture [17]. PEs adopt one of the following three roles in the platform: *(i)* the *slaves* (PE_{SL}) are dedicated to the task execution; *(ii)* the *local manager* (PE_{LM}) manages the cluster resources; *(iii)* the *global manager* (PE_{GM}) provides a high-level management of the MPSoC resources and communicates with the application repository. All PEs run an operating system, dedicated to its attributed role.

In summary, the management in this MPSoC consists in performance monitoring, task mapping, resource sharing between clusters, control of communication QoS between the application tasks (using monitoring packets to detect the communication parameters), and task migration between the PE_{SL}.

The MPSoC is divided into *clusters*, defined at design time. Each cluster has multiple PE_{SL} and a single manager. One of these managers is selected to manage all MPSoC resources (PE_{GM}), besides managing its cluster resources.

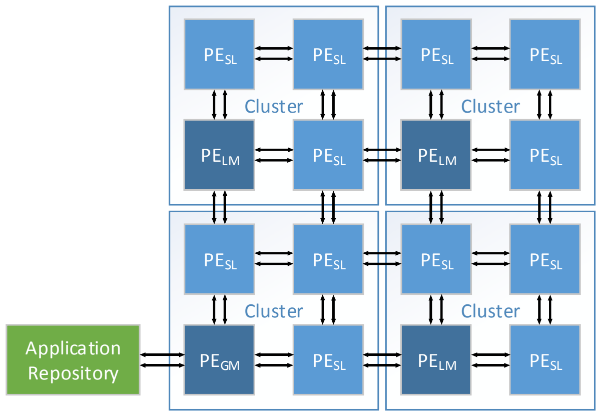

Fig. 1. MPSoC organization.

III. Management Traffic Evaluation

The goal of this section is to present the *motivation* of this work, by evaluating the volume and spatiality of the management traffic, comparing it to the application traffic. The results presented in this section were extracted from selected test cases that illustrate some common workload in the reference platform. The test cases evaluated here use three different applications, all described in C language (real applications, not traces): *(i)* MPEG; *(ii)* DTW; *(iii)* Synthetic (set of tasks communicating at a fixed rate).

All test cases are simulated for 150 ms, each PE_{SL} can execute simultaneously 2 tasks, and all applications are allocated at the beginning of the simulation. The test cases are configured as follows:

- **TC 1:** 8 applications (4 MPEGs and 4 DTWs) executed in a 6x6 MPSoC with four 3x3 clusters;
- **TC 2:** 9 applications (3 MPEGs, 3 DTWs and 3 synthetics) executed in a 6x6 MPSoC with four 3x3 clusters;
- **TC 3:** 16 applications (8 MPEGs, 8 DTWs) executed in an 8x8 MPSoC with four 4x4 clusters;
- **TC 4:** 15 applications (5 MPEGs, 5 DTWs and 5 Synthetics) executed in an 8x8 MPSoC with four 4x4 clusters.

Table I presents a summary of the number of transmitted flits for each TC, divided by traffic categories. The last row indicates the proportion of the management traffic compared to the total traffic. In average, this proportion is between 11% to 15% of the total traffic.

TABLE I. NUMBER OF FLITS TRANSMITTED BY SERVICE

Category	***TC 1***	***TC 2***	***TC 3***	***TC 4***
Application	883,560	741,414	1,797,774	1,532,860
Management	137,508	97,274	304,608	205,798
% Mng	**13.47**	**11.60**	**14.49**	**11.84**

Table II indicates the services with the most significant participation in the management traffic, presenting the total number of flits transmitted for selected services. Monitoring messages are small, but they are very frequent in the platform. Task allocation and the migration messages are less frequent, but they have the largest packet size.

TABLE II. NUMBER OF FLITS TRANSMITTED BY SPECIFIC MANAGEMENT SERVICES

Service	***TC 1***	***TC 2***	***TC 3***	***TC 4***
Monitoring	60,456	27,960	120,456	77,664
Task Mapping	58,920	58,548	147,300	97,580
Task Migration	7,588	0	11,198	10,570
Others	10,544	10,766	25,654	20,004

Table III presents the spatial distribution of the management traffic. The results show that the traffic has a high locality inside the cluster. The traffic between the PE_{GM} and the PE_{SL} from other clusters is also significant, being composed mainly by task mapping messages.

TABLE III. SPATIAL DISTRIBUTION OF THE MANAGEMENT TRAFFIC (IN FLITS)

Traffic	***TC 1***	***TC 2***	***TC 3***	***TC 4***
Intra-cluster traffic: Manager PE $\leftrightarrow$ PE_{SL}	75,042	47,476	166,914	95,146
Inter-cluster traffic: $PE_{GM} \leftrightarrow PE_{SL}$	51,294	46,136	118,362	92,794
Traffic between managers: $PE_{GM} \leftrightarrow PE_{LM}$	3,536	3,614	7,774	7,072
Intra-cluster traffic between PE_{SL}: $PE_{SL} \leftrightarrow PE_{SL}$	7,636	48	11,558	10,786

This evaluation highlighted the behavior of the management traffic: *(i)* important number of injected flits, especially if the application has QoS requirements; *(ii)* monitoring and allocation traffic has an important role in the management traffic; *(iii)* spatial locality between the PE_{SL} and their manager. Such evaluation suggests the usage of an additional network for the management traffic to avoid interferences between the application and management traffic.

IV. Management Network Design

This section describes the implementation of a *management NoC* (*M-NoC*), parallel to the original NoC (data NoC). Two topologies are explored: *(i)* full mesh interconnecting all the PEs; *(ii)* mesh interconnecting all cluster managers.

The *M-NoC* is simpler and considerably smaller than the data NoC employed in the platform mainly due to having fewer and smaller buffers. Table IV lists the central router area results for both networks. The routers were synthesized using a 65 nm technology targeting the operating frequency of 500 MHz. The area of management router is 26% of the data router area.

TABLE IV. ROUTER AREA COMPARISON (LIBRARY CORE65LPLVT, 1.2V, 25OC)

	Area (µm^2)	
Component	***Data Router***	***Management Router***
Buffer (Avg.)	2,100	940
Crossbar	5,203	1,431
Control Logic	2,989	1,130
Router	27,333	7,245

A. *Full Mesh M-NoC*

In this topology, represented in Fig. 2, the *M-NoC* interconnects all the PEs in the network alongside the data network. All management traffic traverses through the management NoC, while the data NoC is dedicated to the application messages.

The communication to the network is controlled by the NI, which supports three physical channels (two for the data network, and one for the *M-NoC*). The NI is divided in receive and send blocks, which allows the NI to send simultaneously and receive packets. The NI is also responsible for serializing/de-serializing the flits since the memory width is 32 bits while the flit width used by both networks is of 16 bits.

At the software layer, in the operating system, all packets are sent by the *send_packet()* procedure. This procedure verifies the packet service, configures the packet to be sent to the correct network, and prepares the packet header to be compatible to the selected network.

B. *Mesh between the managers*

In this topology, represent in the Fig. 3, the management network interconnects only the manager PEs. The wire size between each router is kept small through the inclusion of buffers. This network is used for the management traffic between different clusters. The application packets are constrained to the data network. However, in this topology, the data network also transmits intra-cluster management packets.

The main motivation behind this topology is to avoid the interference between traffics from different clusters, using a low-cost strategy. Also, the manager inter-cluster communication requires fewer hops since it is not necessary to cross multiple routers to reach its destination, wasting fewer clock cycles due to router switching.

To reach its destination, a packet can take a path traversing both the management and data networks. To support this path, a packet can have multiple headers, equal to the number of networks that this packet will traverse. Each header contains the network target address and has a special flag that indicates if the packet must change its network when it reaches its target. Packets can only change the network in the manager PEs. Packets that must change its network are delivered to manager NI and then, redirected to the new network. Every time that this procedure is executed, the NI consumes a header flit.

At the software layer, the *send_packet()* procedure computes the route used by the management, creating the header according to the network changes.

V. RESULTS

This section evaluates the M-NoC network topologies and compares it to the original implementation with a single network.

A. *Experimental Setup*

To compare the different network topologies, three different scenarios are simulated using the clock-cycle accurate SystemC description of the platform. All scenarios are executed in an 8x8 MPSoC with four 4x4 clusters, simulated for 200ms, and each PE$_{SL}$ can execute two tasks simultaneously. The applications executed in each scenario are listed below:

- **Scenario 1:** 25 DTW applications;
- **Scenario 2:** 25 MPEG applications;
- **Scenario 3:** Uses the DTW, MPEG, Dijkstra, MPEG4 and synthetic applications. Each application is instantiated 10 times.

All sent and received packets are logged into text files and are analyzed using a script written in Python, which read the packets logs and matches the sent packets to the received packets (all packets in the platform are unique, since each one contains the source PE and timestamp). This tool supports several analyzes and is used to compute the metrics presented in this section.

The output generated by every application is also logged and is used to determine the application start and end times. The warm-up time is not considered, due to the long simulation time.

B. *Latency and Jitter Evaluation*

Table V lists the application data average latency and jitter for the evaluated scenarios. The management messages values are not considered in this evaluation since only the application data messages are considered critical.

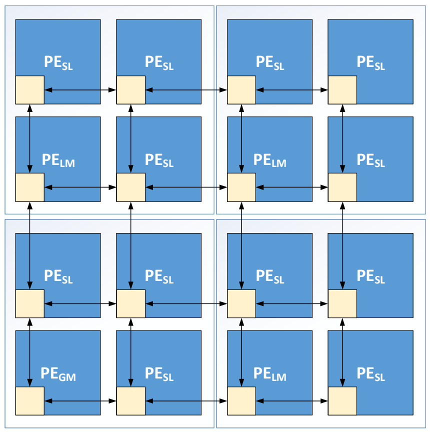

Fig. 2. Full mesh management NoC topology.

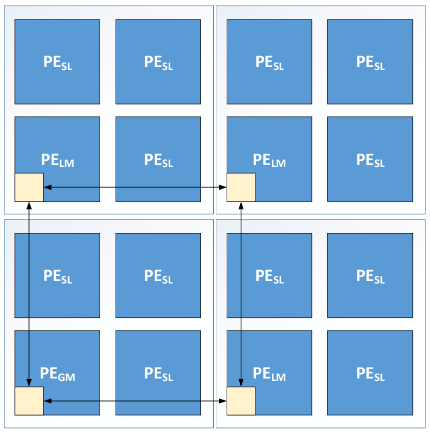

Fig. 3. Mesh between the masters management topology.

The latency used in this evaluation is the number of clock cycles between the injection of header flit into the network until the header flit reaches its target – *network latency*. Since only the first flit of the packet is considered in the evaluation, and the NoC uses the wormhole switching mechanism, the results are independent of the packet size and the time required to copy the packet to the memory when it reaches its destination. The jitter presented here is the standard deviation between all the latencies. Lower jitter values are important for parallel applications since the execution time may increase considerably if some messages experience a much higher latency than the average value [18].

TABLE V. AVERAGE NETWORK LATENCY AND JITTER (CLOCK CYCLES) WITH DIFFERENT M-NOC CONFIGURATION

Scenario	M-NoC	Avg Lat	Jitter
1	None	44,47	316,12
	Full-Mesh	28,73	114,30
	Mesh Mng	46,71	455,39
2	None	29,70	258,44
	Full-Mesh	27,47	204,17
	Mesh Mng	26,06	211,49
3	None	48,24	238,69
	Full-Mesh	37,69	211,62
	Mesh Mng	45,32	274,74

The "Full-Mesh M-NoC" presents better latency and jitter values than the implementation without the M-NoC, reducing the average latency between 10 to 40% and the jitter between 10% to 60%.

However, the "Mesh between the managers M-NoC" presents worse results than the implementation without the M-NoC. This is an unexpected result, but can be explained by Table III since most management traffic occurs inside the cluster or between the PE_{GM} and the slaves, requiring a changing in the network.

C. Execution Time

Table VI lists the average application execution time in milliseconds for different M-NoC configurations. In most of the cases, the applications are executed faster using the "Full-Mesh M-NoC", being in average 5% faster than the implementation without the M-NoC. The "Mesh between the managers M-NoC" presents mixed results and the average applications execution time ends up being similar to the implementation without the M-NoC.

TABLE VI. EXECUTION TIME (MILLISECONDS) WITH DIFFERENT M-NOC CONFIGURATIONS

Scenario	App.	None	Full-Mesh	Mesh Mng
1	DTW	41.63	40.38	42.35
2	MPEG	46.49	43.19	52.31
3	DTW	37.89	29.80	33.47
	MPEG	41.91	41.90	44.34
	Dijkstra	18.38	16.53	18.19
	MPEG4	33.47	34.64	31.94
	Synthetic	45.96	43.21	43.52

The results show that the separation of the management messages from the application messages using different networks for each traffic class can present better performance results when compared to a single network implementation. Besides, the adoption of a second network offers the benefit to increase the management traffic to better control the system.

VI. CONCLUSION

This work analyzed the traffic classes present in distributed memory MPSoC and proposes the utilization of multiple NoCs to improve the application performance, suggesting how to distribute the traffic between the available networks so that the traffic related to the MPSoC management does not interfere with the application data. This work mainly focuses on the aspects of the network used for the management data, proposing two different topologies for this network. Experimental results show a significant improvement in the application messages average latency, jitter and execution time. As future work, more topologies and network parameters for both networks can be explored for design space exploration between the performance and area of the network. In addition, the energy consumption and total area of the networks must be explored, and low power techniques can be applied to the networks considering its traffic patterns.

ACKNOWLEDGMENTS

The Author Fernando Moraes is supported by CNPq - projects 472126/2013-0 and 302625/2012-7, and FAPERGS - project 2242-2551/14-8.

REFERENCES

[1] G. Almeida; G. Sassatelli; P. Benoit; N. Saint-Jean; S. Varyani; L. Torres; M. Robert. *An Adaptive Message Passing MPSoC Framework.* International Journal of Reconfigurable Computing, vol. 2009, 20p, 2009.

[2] R. Banakar; S. Steinke; B. Lee; M. Balakrishnan; P. Marwedel. *Scratchpad Memory: Design Alternative for Cache On-chip Memory in Embedded Systems*. In: CODES, 2002, pp. 73-78.

[3] J. Joven; O. Font-Bach; D. Castells-Rufas; R. Martinez; L. Teres; J; Carrabina. *xENoC - An eXperimental Network-On-Chip Environment for Parallel Distributed Computing on NoC-based MPSoC Architectures*. In: PDP, 2008, pp. 141-148.

[4] E. Carara; R. Oliveira; N. Calazans; F. Moraes. *HeMPS - a framework for NoC-based MPSoC generation*. In: ISCAS, 2009, pp 1345-1348.

[5] J. Balfour; W. Dally. *Design tradeoffs for tiled CMP on-chip networks.* In: ICS, 2006, pp. 187-198.

[6] B. Grot; J. Hestness; S. Keckler; O. Mutlu. *Express Cube Topologies for on-Chip Interconnects*. In: HPCA, 2009, pp. 163-174.

[7] Y. Yoon; N. Concer; M. Petracca; L. Carloni. *Virtual Channels and Multiple Physical Networks: Two Alternatives to Improve NoC Performance*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and System, vol. 32(12), pp. 1906-1919, 2013.

[8] A. Abousamra; R. Melhem; A. Jones. *Déjà Vu Switching for Multiplane NoCs*. In: NoCS, 2012, pp. 11-18.

[9] R. Das; S. Narayanasamy; S. Satpathy; R. Dreslinski. *Catnap: Energy Proportional Multiple Network-on-Chip*. In: ISCA, 2013, pp. 320-331.

[10] S. Volos; C. Seiculescu; B. Grot; N. Pour; B. Falsafi; G. Micheli. *CCNoC: Specializing On-Chip Interconnects for Energy Efficiency in Cache-Coherent Servers*. In. NoCS, 2012, pp. 67-74.

[11] J. Miguel; N. Jerger. *Data Criticality in Network-On-Chip Design*. In. NoCS, 2015, 8p.

[12] C. Ciordas; K. Goossens; T. Basten. *NoC Monitoring: Impact on the Design Flow*. In: ISCAS, 2006, pp. 1981-1984.

[13] M. Sepulveda; D. Florez; S. Das; G. Gogniat, "Reconfigurable Security Architecture for disrupted protection zones in NoC−Based MPSoCs". In: ReCoSoC, 2015, 8p.

[14] M. Taylor; et. al. *The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs*. IEEE Micro, vol. 22(2), pp. 25-35, 2002.

[15] P. Gratz; C. Kim; K. Sankaralingam; H. Hanson; P. Shivakumar; S. Keckler; D. Burger. *On-Chip Interconnection Networks of the TRIPS Chip*. IEEE Micro, vol. 27(5), pp. 41-50, 2007.

[16] D. Wentzlaff; et. al. *On-Chip Interconnection Architecture of the Tile Processor*. IEEE Micro, vol. 27(5), pp 15-31, 2007.

[17] G. Castilhos; M. Mandelli; G. Madalozzo; F. Moraes. *Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes*. In: ISVLSI, 2013, pp. 153-158.

[18] J. Duato; S. Yalamanchili; L. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2004.