Evaluation of Adaptive Management Techniques in NoC-Based MPSoCs

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

***Abstract* — Self-adaptation capability is as a key feature to meet runtime application requirements in dynamic systems such as NoC-Based MPSoCs. This paper presents an evaluation of adaptive management techniques combined with task monitoring, which are able to change at runtime the communication priority, the tasks scheduling priority and execute task migration. The monitoring is designed based on application profiling. The obtained results in terms execution time, latency and jitter, demonstrate the feasibility of such approach and the ability to support real-time applications.**

I. Introduction and Related Work

The use of MPSoCs (Multiprocessor Systems On Chip) with dozens or hundreds of PEs (Processing Elements) interconnected through NoCs (Networks-on-Chip) is a reality in current design of embedded systems [1]. According to ITRS (http://www.itrs.net/), MPSoCs will integrate 1000 PEs in 2025. To keep a scalable growing, systems must contain mechanisms to self-adapt according to the demand of applications and availability of their resources. Techniques such as priority-based task scheduling, communication priorities, task migration, DVFS and circuit switching, has been proposed to cope with the system degradation when multiple general purpose applications are running simultaneously. These techniques are typically managed by a monitoring mechanism, with a global view of the system. To avoid interference with the applications performance, the monitoring scheme should not be intrusive.

State-of-the-art in NoC-based MPSoC monitoring includes works that focus on organizing the monitors in hierarchical levels [2][3]. To keep the system scalability, the monitors are separated in clusters. Each lower level monitor, usually associated with a single PE, sends its monitoring data to an intermediate unit responsible for managing a cluster (cluster manager). This intermediate level unit is often associated with an application. The cluster manager functions include: (*i*) apply the techniques that provide the system adaptability; (*ii*) communicate with other cluster managers; (*iii*) forward its monitored date to a higher-level unit. The higher-level unit creates and deletes clusters according to the flow of applications entering into the system. This solution is only feasible if the data traffic volume on the NoC is low, characterizing the monitoring mechanism as low intrusive. This feature is not easily achieved in large MPSoCs. To cope with this feature, some authors [4][5] adopt physical disjoint networks, isolating the data traffic from the monitoring traffic.

The main goals of task migration in MPSoCs comprise temperature balancing, energy consumption optimization and workload distribution. Acquaviva et al. [6] propose a task migration algorithm that explores temperature uniformity at run-time in MPSoCs. The algorithm, named MiGra, determines the set of tasks that will be migrated from a processor with high temperature to other processor with low temperature. Layouni et al. [7] evaluate a technique of task migration in software with task replication. The technique is executed at run-time and managed by the operating system (OS). Ozturk et al. [8] present a task migration algorithm that decides whether migrates the code or data of the task. This choice is made at run-time, based on statistics collected in the profiling step. Pittau et al. [9] present the task migration performed on a middleware layer. To perform the task migration the Authors evaluate the energy consumption and deadline misses. The experiments show that migration on middleware layer or OS level reduces the energy consumption.

To provide QoS at run-time, Authors [10][11] adopt priorities in the communication flow, separating the generated traffic in different classes, enabling to control the resources allocation according to each priority class. This approach enables to support real-time tasks because it contributes to the isolation of traffic on the network, providing system composability. In this context, some researchers divide the network according to the NoC data traffic [12][13][14]. This division creates sub-networks, by adding new physical channels between routers.

The goal of this work is to implement and evaluate adaptive run-time management techniques in a real NoC-based MPSoC. The first step is to execute the profile of each application, to get its performance parameters. The OS running each PE is responsible to monitor its running tasks, and to detected deadline misses. Based on the monitoring data, some actions may be executed: *(i)* change the task scheduling priority; *(ii)* increase the task time-slice; *(iii)* change the application communication priority; *(iv)* start the task migration.

The originality of the present work is twofold. First, we describe the monitoring scheme along with the run-time management techniques (Sections II and III). Next, such techniques are evaluated (Section IV).

II. Monitoring and Adaptation Modules

The monitoring and adaptation modules manage the available adaptive techniques. Both modules are implemented in software. The monitoring is implemented in the microkernel (OS) of each PE, corresponding to the *local monitors*. The adaptation is implemented in two modules: *(i) request module*, implemented at each local PEs; *(ii) adaptation module*, implemented in a manager PE. This architecture is illustrated in Figure 1.

The Author Fernando Moraes acknowledge the support of CNPq, CAPES, FAPERGS, projects 301599/2009-2, 708/11 and 10/0814-9, respectively. The Authors also acknowledge the National Science and Technology Institute on Embedded Critical Systems (INCT-SEC) for the support to this research.

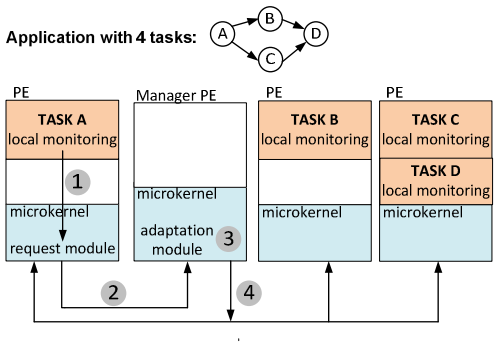

Figure 1 – Monitoring and adaptation architecture.

The monitoring scheme is oriented to a hierarchical structure, where each local monitor sends adaptation requests to a higher-level manager – *manager PE*. The manager PE implements adaptive techniques. The manager PE can be configured to control the entire MPSoC (centralized control) or to control clusters (subset of PEs), exchanging control information with other manager PEs and a higher-level manager.

During the system execution, each *local monitor* (implemented in the microkernel) analyzes the received messages frequency from a given task ('1' in in Figure 1). Each monitored task has a message frequency range defined at design time (profiling step), based on experiments running the applications alone in the system. When a violation is detected, the local monitor invokes the *request module.* This module assembles the monitoring data and sends it to a manager PE, through a *monitoring packet* ('2' in in Figure 1). The manager PE takes the following actions when it receives a monitoring packet: (*i*) identifies the application requesting adaptation and the type of adaptation required; (*ii*) execute the adaption service ('3'); (*iii*) send adaptation packet to all PEs running tasks belonging to the application. The adaptation packet is sent using multicast transmission ('4').

To store monitoring data, each task has a field in its TCB (Task Control Block), which stores the frequency of received messages. Such frequency is measured based on the time interval between subsequent received messages. Comparing the measured frequency to a predefined range (minimum and maximum thresholds), the local monitor can detect deadline misses, and send the monitoring packet to the manager PE.

III. ADAPTIVE MANAGEMENT TECHNIQUES

This section presents three adaptive management techniques. Table I summarizes the effects of the adaptive techniques in the performance of the applications. The task scheduling may improve the computation, while the flow priority may improve the communication. Both approaches can be used to restore the application performance. A more costly approach, task migration, may impact both computation and communication to restore the application performance.

TABLE I. EFFECT OF THE ADAPTIVE TECHNIQUE IN THE COMPUTATION AND COMMUNICATION.

	Effect of the adaptive technique in:	
Technique	Computation	Communication
Scheduling	increase the computation time, by reducing the processor sharing among several tasks	-
Task Migration	move a given task to a free PE (load balancing)	move a task to a non-congested NoC region
Flow priority	-	application traffic is prioritized

A. *Round-Robin Scheduling with Priority-based Preemption*

In the Round-Robin scheduling, the processor is allocated to the tasks following a simple round-robin (circular) policy. Each task may execute for a given period, named *time-slice*, and then the next task is scheduled.

The Round-Robin Scheduling with Priority-based Preemption is described in [15]. The processor is also allocated according to the round-robin policy. However, when a suspended task with higher priority than the current running one becomes ready to run, the later is preempted by the former. In addition, the processor time slice for each task can be set at design time. Such approach adds a certain degree of priority since the time slice for each task can be set according to its processing requirements. A preempted task has its remaining time slice stored in its TCB, which is restored when the task is re-scheduled.

If an application is not meeting the required performance, a request to the manager processor is sent, asking to increase the priority of all applications' tasks. The manager PE sends a multicast packet to all applications' tasks increasing simultaneously the priority and the time-slice (plus 60%) of each task.

B. *Task Migration*

As illustrated in Figure 1, applications are modeled using task graphs. During task mapping, some tasks may be mapped far from each other due to the use of the MPSoC PEs. When resources become available, it is possible to move communication tasks closer to each other. An important benefit of the task migrations, besides performance improvement, is the energy reduction spent in the communication between tasks. In [16] a task migration heuristic is detailed, with the following features: (*i*) complete task migration, including code, data and context; (*ii*) do not require migration checkpoints, i.e., the task may be migrated at any moment; (*iii*) in-order message delivery, i.e, tasks communicating with the migrated tasks will receive the messages in the order they were created. The cost-function of this heuristic is to reduce the communication energy.

As a described in Section II, the local monitor identifies deadline misses. The first action executed when a violation is detected is to request to the manager PE a change in the priority and time-slice value of all tasks belonging to the application. However, if deadlines misses still happens, the task migration is required. The manager PE is responsible to execute the migration heuristic proposed in [16], choosing the PE to receive the task to be migrated.

C. *Communication Priority*

Example of MPSoC that adopts priorities in the communication services is the HeMPS-QoS [17]. Each router has duplicate physical channels at each port. Such approach separates the traffic by creating two disjoint networks, dividing the flow in two classes, high and low priority through channels 0 and 1 of each router port, respectively. The routing algorithm also varies according to the priority of the flow. A low-priority packet uses the minimal deterministic version of the Hamiltonian routing algorithm. A high-priority packet uses the non-minimal partially adaptive version of the Hamiltonian routing algorithm. Therefore, high priority packets may explore non-congested NoC regions, delivering the messages with a smaller latency. In the HeMPS-QoS the packet priority is defined at design time, during the application development.

The present work adopts a different approach. The default flow priority is low, and according to the monitoring result, the microkernel may increase the flow priority. When a given task exceeds the threshold of deadline misses its microkernel requests to the manager PE to change the application flow priority to high. When the monitor identifies the received messages frequency is higher than the maximum threshold, the communication priority of the application returns to low priority.

IV. RESULTS

In this Section, we present results using synthetic and real applications. The experiments use as reference platforms: (*i*) a 6x6 HeMPS [16] instance (with no support to QoS), to evaluate the monitoring with task migration; (*ii*) a 4x4 HeMPS-QoS [17] instance, to evaluate the monitoring with control of communication priorities. All simulations are cycle-accurate, using synthesizable VHDL. The performance of the disturbing applications is not considered, since these applications do not have time constraints (best effort behavior).

A. Monitoring and Task Migration

Relevant features of the MPSoC include: 32-bit PE word and 16-bit flit; memory page size: 16 Kbytes (4,096 works); one task per page; time-slice: 10,000 clock cycles, which can be changed at runtime by the monitoring process.

In the bottom of Figure 2 is presented the graphs of the application with QoS constraints (*main*) and three disturbing ones. All four applications are synthetic. Figure 2 (a) shows the mapping of the main application, which contains six tasks (A, B, C, D, E, F). In a dynamic scenario, where applications are inserted and removed at runtime, the MPSoC resources may become fragmented. At a given moment, new applications (*disturbing*) can be loaded onto the MPSoC (Figure 2(b)), with communication competing with the communication of the main application. In such scenario, the main application performance is penalized and the microkernel of the PE responsible by local monitoring (PE running task F), will request to the manager PE (M) the migration of any task belonging to the main application. Based on the collected data in the profiling step, the manager PE decides which task will be migrated – task C. The manager PE migrate task C to a new PE closer to the other application's tasks (Figure 2(c)), restoring the application performance.

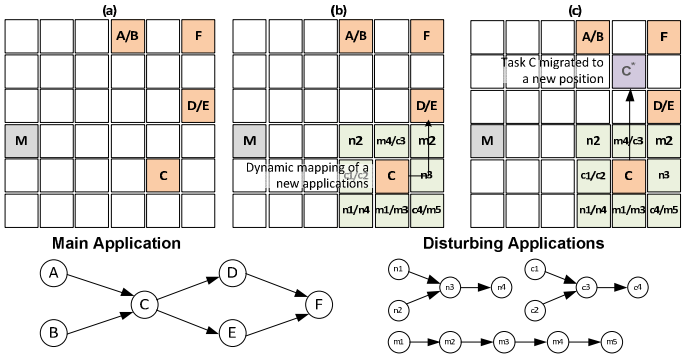

Figure 2– Experimental setup to task migration.

Four scenarios were evaluated: (*i*) main application with no disturbing traffic; (*ii*) main application running with three disturbing applications; (*iii*) same as scenario '*ii*', but with task monitoring changing priorities and time-slice of tasks belonging to the main application; *(iv)* same as scenario '*iii*', but migrating task C.

Figure 3 to Figure 6, present the iteration time of task C. The *main* application is periodic, repeating each task for parameterizable number of iterations. The results obtained in the beginning of the simulation cannot be considered, since it corresponds to the task mapping (warm up), therefore it is as a transient state. Figure 3 (scenario '*i*') shows that after the task mapping, task C iteration time stabilizes in around 4,000 clock cycles (this is due to the difference between the packets arrival time from tasks A and B). We consider this result as the reference one, since there are no disturbing tasks competing with the *main* application. Note also that the execution time in Figure 3 (scenario '*i*') is smaller than the others scenarios, because there is no disturbing traffic.

Figure 4 (scenario '*ii*') shows how the disturbing traffic affects the execution time of task C. After the execution of the disturbing applications, task C returns to a stable execution time. In Figure 5 (scenario '*iii*') the monitoring scheme detects deadline misses, in task F, requiring to the manager PE to modify the scheduling priority and time-slice of all tasks of the *main* application. The execution time of task C then returns to a stable value. Next, new deadline misses occurred and a second adaptation request is executed.

The last evaluation concerns the task migration - Figure 6. The monitoring scheme detects that even modifying the scheduling priority and time-slice is not sufficient to meet the application requirements. In such case, task F ask to the manger processor to migrate one task of the application (task C). During migration, the iteration time of task C increases, restoring the original performance after migration. Note that the oscillation time in the execution of task C reduces compared to Figure 3, since task C is now closer to its communicating tasks.

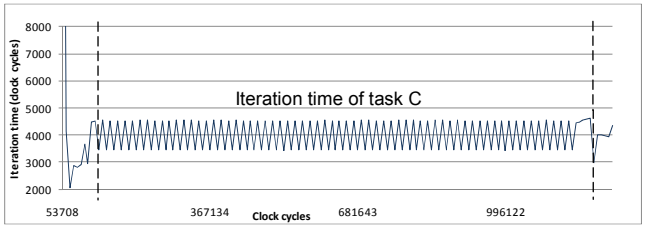

Figure 3 – Task C iteration time, scenario (*i*), without disturbing traffic.

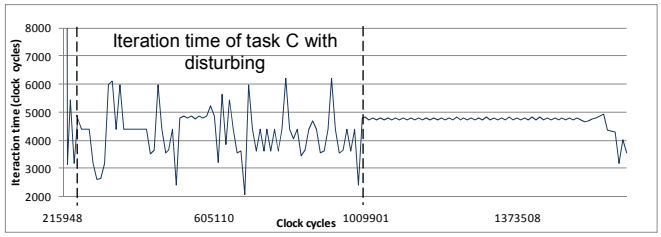

Figure 4 - Task C iteration time, scenario (*ii*), with disturbing traffic.

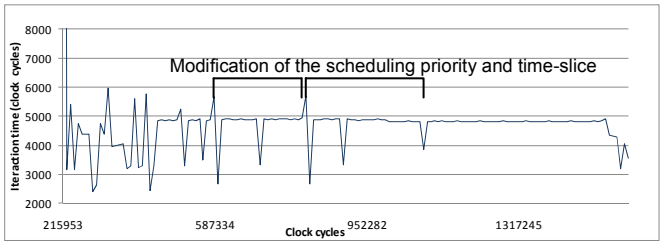

Figure 5 - Task C iteration time, scenario (*iii*), changing the scheduling priority and time-slice.

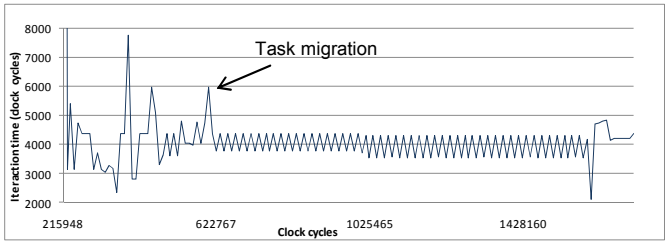

Figure 6– Task C iteration time, scenario (*iv*), with disturbing applications, monitoring, changing the scheduling priority, time-slice, and executing the task migration.

B. Flow Priority

A synthetic and a real application were used to evaluate the adaptability of the priority in the communication. The synthetic

application is a producer-consumer application (2 tasks). Disturbing applications also execute in the system to simulate a scenario where the communication affect the application with performance requirements. Figure 7 shows the latency graph for the producer-consumer application for this first scenario. As expected, the latency presents a small jitter (variation in the latency values) when the disturbing traffic does not execute. During the execution of the disturbing applications, the jitter increases, resulting in unpredictable latency values.

Figure 8 presents the latency graph when monitoring and communication priority are enabled. The latency value used as threshold to modify the flow priority is set in 3,000 clock cycles (highest value in Figure 7). The communication priority is set as low for all tasks in the beginning of the system execution. When the disturbing traffic starts, a parameterizable number of violations in the producer-consumer latency is detected, changing the communication priority to high, restoring the latency values to the original ones. As the latency remains stable for a long period (also parameterizable), the monitoring scheme modifies the communication priority back to low. Again, a set of violations is detected, modifying the priority to high. The goal of Figure 8 was to present the effectiveness to change the communication priority at run-time, together with monitoring. The monitoring window was kept small, to illustrate several priority changes.

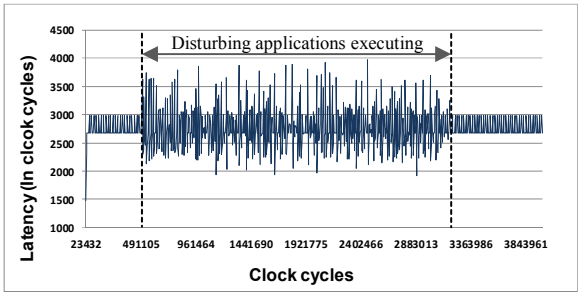

Figure 7 –Latency for the producer-consumer application, with disturbing traffic.

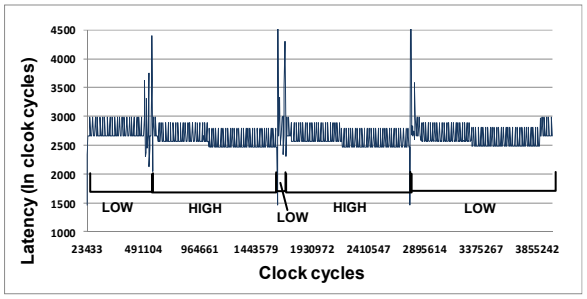

Figure 8 – Latency obtained with disturbing application and with priority adaptation with latency deadline equals to 3000 clock cycles (producer-consumer application).

Without disturbing, most packets are received within an interval of 300 clock cycles (latency in Figure 7 without disturbing between 2,700 and 3,000 clock cycles). With disturbing traffic, the interval between packets varies, resulting in increased jitter, not ensuring QoS. Using the run-time adaption in the flow priority, the jitter is comparable to the scenario without disturbing. Only a small number of packets in the last scenario (27 in a total of 830) have different intervals that correspond to the peak latencies observed in Figure 8.

The real application corresponds to JPEG decoder. The monitoring scheme not modified the total execution time, corresponding to 0.9% of the total traffic of the application. The inclusion of monitoring contributed to the network utilization, because both physical channels may be used by the application. Disabling the monitoring and priority adaptation, the number of messages with a latency that exceeded the deadline was 27. Adding the monitoring and priority adaptation, the number dropped to five messages.

V. Conclusion and Future Works

This paper evaluated three run-time adaptive management techniques for MPSoCs. The proposed techniques act in both communication and computation, using performance monitoring and a manager PE. All techniques prove their efficiency, enabling to restore the default performances, after their application.

The main future work comprises the development of a hierarchical structure for MPSoC management, adding scalability for the proposed techniques for large MPSoCs. In addition, circuit-switching and other scheduling algorithms will be integrated as run-time adaptive mechanisms.

References

[1] 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.

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

[3] Al Faruque, M.A.; Jahn, J.; Ebi, T.; Henkel, J. “Runtime Thermal Management Using Software agents for Multi- and Many-Core Arquiteture”. IEEE Design & Test, v.27(6), 2010, pp. 58-68.

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

[5] Madduri, S.; Vadlamani, R.; Burleson, W.; Tessier, R. “A monitor interconnect and Support Subsystem for Multicore Processor”. In: DATE, 2009, pp. 761-766.

[6] Acquaviva, A.; Carta, S.; Mereu, F.; Micheli, G. “MiGra: a task migration algorithm for reducing temperature gradient in multiprocessor systems on chip”. In: SoC, 2007, 6p.

[7] Layouni, S.; Benkhelifa, M.; Verdier, F.; Chauvet, S. “Multiprocessor task migration implementation in a reconfigurable platform”. In: ReConFig, 2009, pp.362–367.

[8] Ozturk, O; Kandemir, M; Son, S; Karaköy, M. “Selective code/data migration for reducing communication energy in embedded MPSoC architectures”. In: GLSVLSI, 2006, pp. 386-391.

[9] Pittau, M.; Alimonda, A.; Carta, S.; Acquaviva, A. “A Impact of Task Migration on Streaming Multimedia for Embedded Multiprocessors: A Quantitative Evaluation”. In: ESTImedia, 2007, pp. 59-64.

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

[11] Joven, J.; Marongiu, A.; Angiolini, F.; Benini, L.; Micheli, G. “Exploring Programming Model-driven QoS Support for NoC-based Platform”. In: CODES+ISSS, 2010, pp. 65-74.

[12] Lusala, A.K.; Legat, J.-D. “Combining SDM-BASED Circuit Switching with Packet Switching in a NoC for Real-Time Application”. In: ISCAS, 2011, pp. 2505 – 2508.

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

[14] 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.

[15] Li, J; Yao, C. “Real-Time Concepts for Embedded Systems”. CPM Books, 2003, 294p.

[16] Moraes, F; Madalozzo, G; Castilhos, G.; Carara, E. “Proposal and Evaluation of a Task Migration Protocol for NoC-based MPSoCs”. In: ISCAS, 2012, pp. 644-647.

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

[18] Carara, E.; Oliveira, R.; Calazans, N.; Moraes, F. “HeMPS - A Framework for NoC-Based MPSoC Generation”. In: ISCAS, 2009, pp. 1345-1348.