Mapping of Real-Time Applications on a Packet Switching NoC-based MPSoC

Guilherme Madalozzo[1], Leandro S. Indrusiak[2], Fernando G. Moraes[1]
[1] PUCRS University, Department of Computer Science, Porto Alegre, Brazil
[2] University of York, Department of Computer Science, York, United Kingdom
guilherme.madalozzo@acad.pucrs.br, leandro.indrusiak@york.ac.uk, fernando.moraes@pucrs.br

***Abstract*—Many-core systems are commonplace in the consumer electronic market, with Real-Time (RT) applications being part of the workload. Therefore, the design of many-core systems (MPSoCs) requires mechanisms to meet RT constraints. The goal of this paper is to present a mapping heuristic that guarantees the RT constraints in a packet switching NoC-based MPSoC. The RT mapping heuristic adopts the response time using the Response Time Analysis (RTA) test to meet the RT constraints. The RTA ensures that already mapped tasks dot not interfere in the mapping of a new task. Experiments executing in a cycle-accurate MPSoC shows that the use of the proposed mapping guarantees the constraints of embedded RT applications. The proposed mapping is compared with two reference mappings (LEC-DN and HEAT), varying the MPSoC occupation and the number of RT applications executing simultaneously in the system. In all experiments, the RTA mapping meet all deadlines, while the other mapping heuristics present deadline misses. On the other side, the RTA mapping presents a smaller MPSoC occupation since it reduces the CPU sharing.**

Keywords—Mapping; real-time; NOC; embedded applications; MPSoC

I. INTRODUCTION AND RELATED WORKS

The workload of many-core systems includes RT applications. The RT constraints are usually specified as a deadline to execute a given function [1]. Embedded RT applications contain a set of tasks, where the correct execution includes not only the expected results but also when these results are produced [2]. Such embedded applications execute on, for example, multimedia, robotics, and avionics systems.

This paper presents mapping heuristic targeting RT applications for packet switching NoC-based MPSoCs. The goal of a mapping heuristic is to assign tasks to processors. A rich literature on mapping heuristic is available. For example, Singh et. al. [3] survey recent mapping proposals. The optimization function of the proposals includes execution time, reliability, energy consumption, throughput, resource utilization, communication QoS, and temperature, among others.

The *goal* of this work is to propose a mapping heuristic that guarantees real-time constraints on packet switching NoC-based MPSoCs. The proposed mapping adopts the schedulability analysis, which computes the response time of a given task using the Response Time Analysis (RTA) test [2].

Authors in [4] and [5] propose algorithms to map real-time streaming applications to reduce the energy consumption and guarantee the latency constraints. To reduce the energy consumption, Holzenspies et al. [5] optimize the resources usage in a spatial mapping. Liu et al. [4] present an analysis of the energy consumption in two task mapping heuristics: First-Fit-Decreasing (FFD) and Worst-Fit-Decreasing (WFD).

In [3], the proposal is to map communicating tasks into the same processing element to reduce the communication overhead. Marwedel et al. [6] present a mapping that considers both performance and thermal characteristics of real-time systems. The work analyzes the timing property and provides real-time guarantees on investigated systems, integrating the modular performance analysis (MPA).

Baloukas et al. [7] define an RT mapping that reduces the number of memory accesses and storage. The heuristic presented by Authors is called memory-aware mapping, focuses on finding a mapping that minimizes the energy consumption for the system while considering the memory requirements of tasks. Zadrija et al. [8] implement a mapping with the goal to optimize the workload distribution. The work presents a mapping engine, with two task mapping algorithms, i.e. load balance and longest processing time.

Most of the above works target communication QoS or optimization of some performance parameter (e.g. scheduling and memory accesses). RT analysis is not the first optimization goal of such works. The *main contributions* of this work include:

- Proposal of a mapping heuristic that guarantees real-time constraints using RTA;
- Validation of the proposed mapping heuristic in a clock cycle-accurate NoC-based MPSoC model;
- Comparison of the proposed mapping with two mapping heuristic: LEC-DN [9], which minimizes the communication energy; and HEAT [10], which distributes the workload into the system to minimize the energy consumption. The goal is to compare the present proposal with mappings having cost functions found in the literature (communication energy and workload distribution).

II. APPLICATION MODEL

Applications are modeled as task graphs [11], A_{pp}=<T,C>. The set of the application tasks, T={t_1, t_2, t_n}, corresponds to the graph vertices. C={(t_i, t_j, w_{ij})} | (t_i, t_j) Є T and w_{ij} Є N*} represents the communication between tasks, corresponding to the graph edges. The communication between tasks occurs through message passing, using a non-blocking *Send()* and blocking *Receive()* (MPI-based primitives).

A 4-tuple defines the constraints of an RT task t_i: {C_i, T_i, D_i, P_i} (Figure 1), where: C_i: computation time of t_i; T_i: t_i period; D_i: deadline to execute t_i ($D_i \leq T_i$); P_i: t_i priority. The task priority defines the execution priority in a multitask OS.

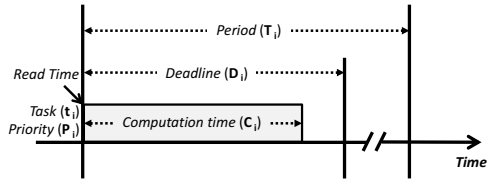


Figure 1 – Real-time constraint model for a task t_i.

The application defines the constraints of the RT tasks. A system call (*syscall*) named *RealTimeConstraints* is in charge of informing the operating system (OS) of the real-time (RT) constraints of each task. Figure 2 presents an example of a task code that configures the RT constraints of task A. At line 3, task A calls the *RealTimeConstraints syscall*, notifying to the OS the constraints. At lines 4-8, the task executes some computation, repeated periodically. Note that it is possible to re-execute at runtime the *RealTimeConstraint* syscall modifying the RT constraints.

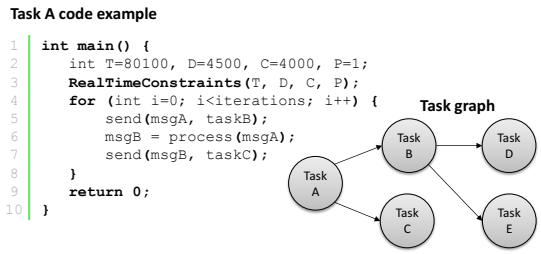


Figure 2 – Application model, task graph and task code example.

An application, like the one of Figure 2, contains n tasks with different periods. The *hyper-period* is the interval of time where all tasks execute one iteration. Therefore, the hyper-period includes the execution time of all tasks of the application, as well as the communication time due to the exchange of messages between tasks.

This work adopts a *priority-preemptive scheduling* [12] to execute the tasks mapped to each processor of the system. The tasks may assume 3 states: (*i*) *running*, when the task is executing; (*ii*) *waiting*, when OS blocks the task, or the task is waiting for a NoC interruption (e.g. blocked by a *receive()*); and (*iii*) *blocked*, when the task is preempted by the scheduler to execute a task with higher priority. Figure 3 presents a scenario with 2 tasks mapped in the same processor. Initially, t_1 starts its computation. When t_2 is ready in the processor (at time 2.5), t_1 is *blocked*, and t_2 starts its computation (t_2 priority is higher than t_1 priority). At time 4.5 t_2 change its status to *waiting*, enabling the execution of t_1. Note that task t_1 never blocks t_2, because t_2 priority is higher than t_1 priority.

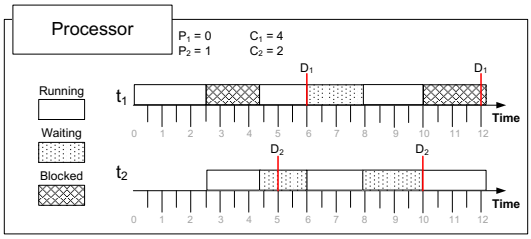


Figure 3 – Priority-preemptive scheduler with two tasks sharing the same processor.

A feature of the *priority-preemptive scheduling* is that it always prioritizes the task with higher priority. Thus, the task with lower priority can miss its deadlines, as we can note in Figure 3 with task t_1 missing a deadline at time 12. To avoid deadline misses, the next Section presents a schedulability analysis that evaluates the interferences between RT tasks sharing a processor.

III. SCHEDULABILITY ANALYSIS

The schedulability analysis evaluates whether a many-core system can fulfill all its timing constraints [1]. This work implements the Response Time Analysis (RTA) [2] as a technique to evaluate how much the interference from higher priority tasks, in the same processor, can delay the computation time of a lower priority RT task.

Equation (1) [1][2] computes the response time (R_i) of t_i, that result in the longest possible time interval between the ready time (start time) of t_i and its iteration termination.

$$R_i = C_i + \sum_{\forall\, t_j \in hp(t_i)} \left\lceil \frac{R_i}{T_j} \right\rceil . C_j \qquad (1)$$

The computation occurs by adding the computation time required by t_i (C_i) on each release and computation time of all tasks that can preempt t_i in the same processor (C_j). The $hp(t_i)$ function denotes the set of all tasks, in the same processor and with a higher priority, that can preempt t_i. To guarantee deadlines, R_i needs lower or equal to D_i.

The computation time (C_i) includes the communication time of t_i with other tasks. Table I evaluates the relationship computation versus communication for four benchmarks, with one task mapped per PE (processing element). These experiments show that the communication time is inferior to 6% of the computation time. Associated to the small NoC usage by the applications, tasks of the same application are mapped in the same region, minimizing the interference with other flows. Thus, the communication overhead between communicating tasks can be embedded in the C_i.

TABLE I – COMMUNICATION AND COMPUTATION TIME, FOR ONE ITERATION OF EACH BENCHMARK.

Applications	*Execution Time (ck cycles)*	*Communication time (ck cycles)*	*Computation/ communication*
MPEG	70,590	863	1.2%
Dijkstra	78,000	2,517	3.2%
DTW	125,750	1,666	1.3%
VOPD	24,037	1,396	5.8%

IV. REAL-TIME MAPPING HEURISTIC

The present work adopts a NoC-based MPSoC with distributed memory architecture. Each processor contains a scratchpad memory as local storage memory and executes a multitask OS.

The proposed RT mapping heuristic uses the following definitions and functions:

Def. 1: *N_Apps*, set of applications to be mapped into the system.

Def. 2: $app_i.size$, the number of tasks of app_i, where $app_i \in N\_Apps$.

Def. 3: *pe_set*(PE_{add}, *n_hops*), a function that returns a set of PEs, in a region defined by the position of PE_{add} and the set of PEs in a distance up to *n_hops* to PE_{add}.

Def. 4: *rt_available*(pe_i), a function that returns *true* when PE_i has no RT task mapped on it.

Def. 5: *is_rt_app*(app_i), a function that returns *true* if app_i is RT.

Def. 6: *response_time*($task_i$, pe_i), a function that returns *true* if the response time (R_i) of $task_i$ on pe_i, based on Equation (1), is lower than the deadline of $task_i$.

The proposed algorithm tries to map all tasks of a given application, considering the already mapped applications. If a task cannot be mapped, this implies that the current MPSoC configuration cannot guarantee the RT constraints, and the application is not mapped into the system.

The system supports RT and Best-Effort (BE) applications (no constraints to meet). Algorithm 1 presents the pseudo-code of the RT mapping heuristic. The inputs of the algorithm are the application to map, app_i, and the PE address used as the initial position to map the tasks of app_i – *seed*. The output is a vector with the position of all tasks of app_i. The first *seed* to map a given application is defined at the address (1,1), and the tasks of app_i are mapped, if possible, at this address and in the surrounding PEs. The *seed* for the next application is selected two hops far from the rightmost mapped task of app_i, or two hops above of the leftmost mapped application. This procedure to elect a *seed* to map an application ensures locality and minimizes the interference of flows belonging to different applications.

The loop starting at line 1 tries to map all tasks of app_i. Line 2 assigns an invalid position for t_j, and line 3 defines the search space around t_j as 1 hop. Line 4 starts an internal loop, exiting when all PEs

were evaluated or t_j was mapped. Lines 7 to 17 map the RT tasks. Line 7 gets a set of candidate PEs to receive t_j. The loop from line 8 to 14 evaluate all PEs in the candidate set, evaluating the availability of the PE and the RTA test (lines 9-10). The first available PE in the candidate set with the $R_{task} \leq D_{task}$ is selected to receive t_j. Lines 15-17 increase the search space in such a way to increase the candidate set on line 7. If the application is BE, the tasks are mapped according to the LEC-DN heuristic [9] (line 19). If a given task cannot be mapped a message is displayed (line 22).

```
Input: application app_i, seed
Output: app_mapped[app_i.size]
1.   FOR EACH task t_j IN app_i
2.     app_position[t_j] ← -1
3.     n_hops ← 1
4.     WHILE all PEs in the system were not evaluated
5.           AND app_position[t_j]=-1 DO
6.       IF is_rt_app(app_i) THEN
7.          neighbors_list ← pe_set(seed, n_hops)
8.          FOR EACH PE_k IN neighbors_list
9.             IF rt_available(PE_k) AND
10.                      respose_time(t_j, PE_k) THEN
11.               app_position[t_j] ← PE_k
12.               break;
13.            END IF
14.         END FOR
15.         IF app_position[t_j] = -1 THEN
16.            increase(n_hop, 1)
17.         END IF
18.      ELSIF
19.         Map the task executing the LEC-DN mapping [9]
20.    END WHILE
21.    IF app_position[t_j] = -1 THEN
22.       message(Application is not schedulable!)
23.    END IF
24.  END FOR
25.  return app_mapped[]
```

Algorithm 1 – Mapping of RT tasks.

Two situations prevent the mapping of a given application. The first one is the absence of available PEs. The second one is applicable for RT applications, when the *response_time()* (line 10) function returns *false* for all PEs.

The proposed mapping guarantees that RT applications do not miss deadlines when executed in the system. The designer configures the platform with the set of the applications to execute and the MPSoC configuration. The RT mapping (algorithm 1) tries to map the application set. If all tasks can be mapped, the system can execute the application set. Otherwise, the designer must modify the MPSoC characteristics or the constraints of the RT applications.

These design steps assume a static mapping, i.e., the mapping decisions are made at design time. There is no restriction to execute the proposed mapping at *runtime*. A manager PE with the knowledge of the system utilization may map the incoming RT applications according to the algorithm 1.

V. EXPERIMENTAL SETUP AND RESULTS

The MPSoC consists of a set of homogeneous PEs interconnected by a packet switching NoC, without virtual channels. Only local memories are used. The adoption of this memory organization reduces the NoC traffic, using message passing as communication model. A clock-cycle accurate SystemC description models the MPSoC. Applications and OS are described in C language, compiled from C code and executed over cycle-accurate models of the processing cores.

This work adopts two reference task mappings: (*i*) Low Energy Consumption – Dependences Neighborhood (LEC-DN) [9], and (*ii*) HEAT (Hierarchical Energy-Aware Task Mapping) [10]. The LEC-DN mapping uses the total communication energy of the communicating tasks as the cost function, considering the distance between tasks and total communication volume (in *flits*). The HEAT mapping distributes the workload and the energy consumption. The goal of the HEAT mapping is to minimize the occurrence of hotspots, increasing the system lifetime.

A. Description of the Evaluated Scenarios

Four applications are used as benchmarks: (1) *MPEG*: with 5 tasks, 100 deadlines, decode video frames; (2) *Dijkstra*: with 6 tasks, 161 deadlines, finds the shortest paths between nodes in a graph; (3) *DTW*: with 6 tasks, 120 deadlines, measures the similarity between two images; (4) *VOPD*: with 12 task, 12 deadlines.

Definition: *MPSoC occupation*, the number of tasks that a given MPSoC configuration can execute simultaneously, according to the number of PEs and the number of tasks each PE may execute. For example, a 6x6 MPSoC, with 2 tasks per PE, can execute 72 tasks.

The experiments adopt two MPSoC sizes, 6x6 and 8x8, and each PE is configured to execute 1 to 3 tasks. The MPSoC occupation may vary from 10% to 100%. A script configures the scenarios selecting the application set to execute according to the MPSoC occupation.

Table II presents the number of deadlines to meet, assuming that all applications executing in the system have RT constraints (100% RT scenarios). When the RTA test fails in a given configuration, the number of deadlines is not presented. Note that as the number of tasks per PE increases, the occupation per scenarios reduces. The reason is that the RT mapping fails with higher CPU sharing.

TABLE II – NUMBER OF DEADLINES IN THE 100% RT SCENARIOS (38 DIFFERENT SCENARIOS).

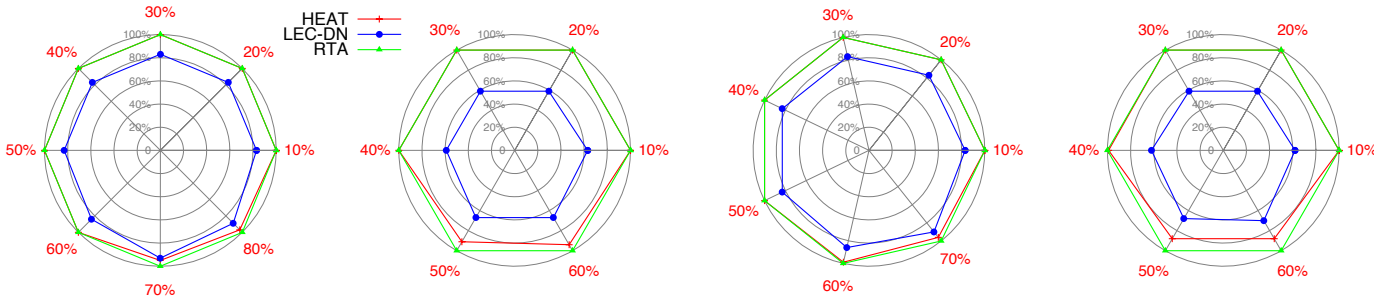
MPSoC Occupation (%)	*6x6 MPSoC size* Tasks per PE			*8x8 MPSoC size* Tasks per PE		
	1	2	3	1	2	3
10	100	120	161	100	161	112
20	120	281	173	161	273	513
30	161	173	393	112	513	686
40	281	293	554	273	566	-
50	281	393	-	393	786	-
60	173	554	-	513	-	-
70	173	-	-	405	-	-
80	293	-	-	566	-	-
90	393	-	-	686	-	-
100	393	-	-	786	-	-

Table III presents a realistic scenario, with a mix of RT and BE applications (RT+BE). Only the MPEG application has RT constraints. The goal of the 100% RT scenarios is to stress the RTA test. Now, in the RT+BE scenarios, the occupation of the MPSoC is higher, reaching 60% in the 8x8 MPSoC, with three tasks per PE.

TABLE III – NUMBER OF DEADLINES IN THE RT+BE SCENARIOS (47 DIFFERENT SCENARIOS).

MPSoC Occupation (%)	*6x6 MPSoC size* Tasks per PE			*8x8 MPSoC size* Tasks per PE		
	1	2	3	1	2	3
10	100	100	100	100	100	100
20	100	100	100	100	100	200
30	100	100	100	100	200	300
40	100	100	200	100	300	400
50	100	200	300	100	300	500
60	100	200	300	200	400	600
70	100	200	-	200	500	-
80	100	300	-	300	-	-
90	200	-	-	300	-	-
100	200	-	-	300	-	-

It is important to mention that the MPSoC occupation due to the RTA test observed in Tables II and III is not absolute. A different application set may vary the MPSoC occupation. *The relevant result is the observed reduction on the MPSoC occupation when the percentage of RT applications or the CPU sharing increases.*



(a) 6x6 MPSoC - 2 tasks per PE (b) 6x6 MPSoC - 3 tasks per PE (c) 8x8 MPSoC - 2 tasks per PE (d) 8x8 MPSoC - 3 tasks per PE

Figure 4 – Percentage of deadlines met by the mapping heuristics (external numbers: MPSoC occupation, vertical numbers: deadlines met).

B. Results

The first evaluation concerns the 100% RT scenarios. In all evaluated scenarios, the HEAT and RTA mappings met all RT constraints. In the 6x6 and 8x8 MPSoC sizes, with more than 1 task per PE, the LEC-DN mapping missed 60% to 80% of the deadlines in all scenarios. The small MPSoC occupation for 2 (60% and 50%) and 3 tasks (40% and 30%) favored the HEAT mapping since its optimization function is the workload distribution, minimizing the CPU sharing. On the other side, the LEC-DN mapping minimizes the communication energy, favoring the CPU sharing, and consequently inducing deadline misses.

The second evaluation concerns the RT+BE scenarios. The three mappings heuristics met all deadlines for scenarios with 1 task per PE. All mappings heuristics favors the locality of the communicating tasks, explaining this result. For scenarios with 2 and 3 tasks per PE, only the RTA mapping met *all* deadlines in *all* scenarios. Figure 4 presents the percentage of deadlines met by each mapping. The external numbers (in red) represent the MPSoC occupation, the percentage numbers (vertical labels) correspond to the percentage of deadlines that were met, and the lines represent the mapping heuristics.

Using a mix of RT+BE applications changes the behavior of the results compared to the scenarios 100% RT:

- The LEC-DN mapping always misses deadlines. Such results show that traditional mapping heuristics, having the goal to reduce the NoC traffic, should not be applied to applications with RT constraints.
- The HEAT mapping, with two tasks per PE, met up to 96% of the deadlines (Figure 4(a) and (c)), and with three tasks per PE met up to 94% of the deadlines (Figure 4(b) and (d)). As the goal of this heuristic is to optimize the workload avoiding hotspots, the interference between RT tasks is not accounted, resulting in deadline misses.

The RT mapping guarantees 100% of deadlines in all scenarios due to the RTA test. These results showed that an explicit optimization function must be adopted in mapping heuristics to ensure the behavior of RT applications. Secondary optimization functions may be added to the RT mapping, evaluating, for example, the occurrence of hotspots.

VI. CONCLUSION AND FUTURE WORKS

This paper proposed a mapping heuristic that guarantees the deadlines of RT applications, executing in a packet switching NoC-based MPSoC system. The characterization of the amount of traffic generated by different benchmarks showed that it is possible to include the communication cost in the computation cost because the benchmarks are computation-centric, with a small injection rate (<6%). This is a relevant result since the NoC architecture may be kept simple, without using virtual channels. The evaluation of other mapping heuristics demonstrated that it is not possible to adopt them with a workload having applications with RT constraints. To execute a workload with RT applications an explicit RT optimization function is required. In the current work, the RTA test enabled the implementation of the RT mapping.

Future works include: (*i*) evaluated other RT schedulers (e.g. least slack time, round-robin based on priority, rate-monotonic, earliest deadline first); (*ii*) extend the RT mapping to map tasks at runtime; (*iii*) evaluate the latency and jitter of the RT communication flows; (*iv*) extend the method to communication-centric applications.

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] Indrusiak, L. "*End-to-end Schedulability Tests for Multiprocessor Embedded Systems based on Network-on-Chip with Priority-preemptive Arbitration*". Journal System Architecture, v.60(7), 2014, pp.553-561.

[2] Audsley, N.; Burns, A.; Richardson, M. "*Applying New Scheduling Theory To Static Priority Preemptive Scheduling*". Software Engineering Journal, v.8(5), 1993, pp.284-292.

[3] Singh, A.; et al. "*Mapping on Multi/Many-core Systems: Survey of Current and Emerging Trends*". In: DAC 13, 10p.

[4] Liu, D.; Spasic, J.; Chen, G. "*Energy-Efficient Mapping of Real-Time Streaming Applications on Cluster Heterogeneous MPSoCs*". In: ESTIMedia, 2013, pp.1-10.

[5] Holzenspies, P.; Smit, G.; Kuper, J. "*Mapping streaming applications on a reconfigurable MPSoC platform at run-time*". In: SOC, 2007, pp. 74-77 .

[6] Marwedel, P.; Teich, J.; Kouveli, G. "*Mapping of Application to MPSoCs"*. In: CODES+ISSS, 2011, 10p.

[7] Baloukas, C.; Papadopoulos, L.; Soudris, D. "*Mapping embedded applications on MPSoCs: The MNEMEE approach*". Lecture Notes in Electrical Engeneering, v.105, 2011, pp.165-179.

[8] Zadrija, V. Sruk, V. "*Mapping algorithms for MPSoC synthesis*". In: MIPRO, 2010, pp.624-629.

[9] Mandelli, M.; Castilhos, G.; Sassatelli, G.; Ost, L.; Moraes, F. "A Distributed Energy-aware Task Mapping to Achieve Thermal Balancing and Improve Reliability of Many-core Systems". In: SBCCI, 2015, 6p.

[10] Castilhos, G.; et al. "*Hierarchical Energy Monitoring for Task Mapping in Many-core Systems*". Journal Science Architecture, v.63, pp.80-92, 2016.

[11] Roig, C.; Ripoll, A; Guirado, F. "*A New Task Graph Model for Mapping Message Passing Applications*". IEEE Transactions on Parallel and Distributed Systems, v.18(12), 2007, pp.1740-1753.

[12] Goossens, J.; Funk, S.; Baruah, S. "*Priority-driven Scheduling of Periodic Task Systems on Multiprocessors*". Real-time Systems, v.25(3), 2003, pp.187-205.