Heuristics for Mapping Real-time Applications to NoC-based Architectures using Genetic Algorithms

Iaê Santos Bonilha, Osmar Marchi dos Santos
Department of Electronic and Computing
Universidade Federal de Santa Maria
Santa Maria, Brazil
{iaesb, osmar}@inf.ufsm.br

Leandro Indrusiak
Computer Science Department
University of York
York, UK
lsi@cs.york.ac.uk

Abstract— **One crucial step in the development of an application for a NoC-based MPSoC, consists on the allocation of the tasks that compose the system. When we consider such architecture, besides the computation time of the tasks, we also have to take into account the communication time between tasks in order to allocate and guarantee that tasks do not miss their deadlines. Communication between tasks is a factor that impacts directly in the real time system temporal behaviour. However, its combination with task execution is not fully taken in consideration in NoC-based architectures. In this paper we focus on the discussion and development of different heuristics that are used for task mapping with the use of genetic algorithms. We use different response-time equations in order to provide a feasibility analysis that works for both task execution and communication. Our evaluation of the algorithms show that certain heuristics can provide quite an improvement on the processor utilization.**

Real-time systems; Genetic algorithms; Multi-core; Network-On-Chip

I. Introduction

Real-time systems are intrinsically related to embedded system – systems designed with a specific application in mind. Due to physical limitations regarding processor speed imposed over single-processor systems [9], multi-processor systems are commonly used nowadays, especially for the development of (real-time) embedded systems. Often, such systems are characterized as having soft and hard real-time constraints, where deadline miss for a task with hard real-time constraint may result in catastrophic problems [12].

In order to use multi-processor architectures for the development of real-time systems, especially those with hard real-time constraints, the designer has to deal with the problem of task mapping if the system uses a static allocation of tasks (know as partitioned scheduling). Therefore, each task competes for processor occupation only with the tasks at the same processor and, therefore, different task mappings may produce different temporal behaviours for the task set being mapped. Task mapping on a real-time system may be characterized as an NP-complete problem known as bin packing [11].

Another inherit problem of real-time multi-processor architectures is the time that memory sharing may affect the system [1]. Since the time taken to share memory may directly impact over the timing behaviour of the task set. From a theoretical point of view, the memory sharing time is often ignored.

In this work, we consider Network-on-Chip (NoC) architectures for multi-processor systems. NoC uses the concept of computer network to exchange data between cores, enabling the calculation of sharing memory time between cores of the system by use prioritized communication flows between tasks. Considering such architecture and combining different response-time equations for both task and communication flows [6], it is possible to execute a feasibility analysis over the system in order to achieve temporal guarantees [10].

The contribution of this paper is the discussion and development of different heuristics for improving the task mapping in the context of genetic algorithms. This problem has been considered in previous work [2][3][4]. However, in this work we focus on using inherited aspects from real-time systems and NoC-based architecture and we provide three different main heuristics for task mapping: (i) utilization balance between cores; (ii) quantity of deadline misses; (iii) network stress. Our evaluation results show that the timing constraints imposed by the communication flows have a direct impact over the timing constraints for the tasks in the system. Therefore, it becomes very important to consider the time for memory sharing in the context of real-time multi-processor systems.

This paper is organised as follows. Section II describes the NoC architecture we will be using in this work. Section III presents the computational model, and the combination of response-time equations that enables the feasibility analysis of the system. In Section IV, different heuristics derived from inherent properties of the NoC architecture and system model are proposed. In order to address the problem of task mapping, Section V describes the basic principles behind genetic algorithms, which is used to implement and evaluate the proposed heuristics. Experimental work presenting the results for the developed task mapping heuristics using genetic algorithms is shown in Section VI. Conclusions and future work are presented in Section VII.

II. NoC Architecture

The design space of NoC architectures is very large, as many of its components can be parameterized to better meet design goals: routers, arbiters, buffers, flow controllers, among others. However, the experience acquired through the development of many commercial and research-oriented NoCs allowed the identification of a few mechanisms that are adequate for a wide variety of NoC configurations, and so they were adopted widely. For example, sophisticated routing algorithms generate more overhead to the overall and do not significantly reduce communication latency when compared with simple deterministic routers such as XY. Wormhole switching [5] is another example of a mechanism that was widely adopted in NoCs because it does not require large capacity buffers (which in turn means lower power and area overhead of the routers, a top priority among NoC designers).

In this paper, we also adopt such widely used architectural patterns and consider NoCs with mesh topology, wormhole switching and XY deterministic routing. In this paper, we focus on one particular architectural construct that provides a flow controller based on priority preemptive virtual channels. By assigning priorities to packets, and by allowing high priority packets to preempt the transmission of low priority ones, we are able to use response-time equations in order to analyse the schedulability analysis of the traffic inside the NoC [6]. Figure 1 shows the internal structure of a NoC router using such architecture. In each input port, a different FIFO buffer stores flits of packets arriving through different virtual channels (one for each priority level). The router assigns an output port for each incoming packet according to their destination. A credit-based approach guarantees that data is only forwarded from a router to the next when there is enough buffer space to hold it.

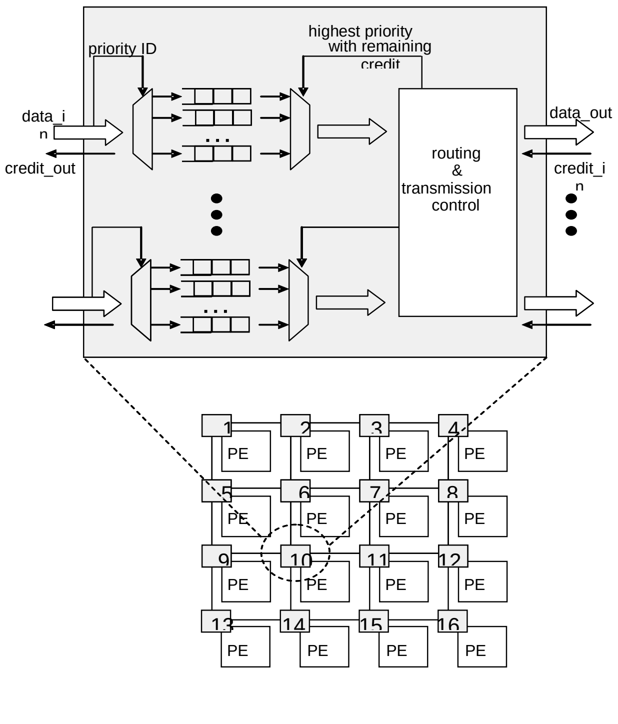

Figure 1. NoC architecture with detail of the router structure

At any time, a flit of a given packet will be sent through its respective output port if (i) it has the highest priority among the packets being sent out through that port and (ii) the next router in its route has enough space in its buffer. If the highest priority packet cannot send data because it is blocked elsewhere in the network, the next highest priority packet can access the output link. The identified architecture is therefore able to provide guaranteed throughput (GT) to traffic of higher priority, and also provides means to calculate upper latency bounds to best-effort (BE) traffic, as the priority ordering clearly shows when a packet will be blocked. Its approach to guarantee throughput is more efficient than time-division multiplexing (TDM), as used in many NoC. Priority preemptive arbitration does not unnecessarily reserves resources, so low priority traffic can always use the NoC if there are no requests from GT flows.

III. COMPUTATIONAL MODEL

In this paper we will be using fixed priority scheduling for a multi-core system. The scheduling strategy used will be partitioned scheduling, where no migration of tasks is allowed [9]. We consider that each task in the system has a periodic behaviour, and is composed of two parts, described by: (i) computation time, (ii) followed by communication time. This way, we define that each task in the system executes its defined computation time in the allocated processor. After its processing time, the task is able to send data through the NoC. The second part of the task’s behaviour is optional, since not every task in the system is required to communicate with other tasks. Using this behaviour for the tasks of the system, we are able to represent both task computation and communication using response-time equations, which enables the analysis of the system in order to verify if deadlines are missed. This representation and analysis is presented in [10] and is discussed as follows.

Tasks in the system are represented by the tuple t_i=(C_i, T_i, D_i, c_i). The task period (T_i) is the minimum time interval between two consecutive releases of the referenced task. The task's computation time (C_i) represents the processor time needed for a task to complete its execution. A task deadline (D_i) is the time window where the task's results will be valid, considered to be equal to the tasks period in this paper. After a task complete its execution, it will start the communication activities described by c_i.

As described earlier, the communication scheme used in this paper will be wormhole switching [5] with the minor exception that instead of multiple communication flows (communication produced by a single process) sharing the same virtual channel, every communication flow will have its own virtual channel with the same priority of the task that generates the flow. Therefore, in order to define the priorities for the tasks in the system, and consequently the priority of the communication flows, we adopt the rate monotonic algorithm [7] – an optimal algorithm, among fixed priority algorithms, based on the period of the tasks. The restriction related to the definition of priorities is due to the characteristics of the system. That is, the actual period for a communication flow is based on the period of the task. In terms of priority definitions, both should have the same relation in the overall system architecture.

The communication flow ci related to a task is composed by a set of properties represented by the tuple c_i=($L_i, T_i, D_i, J_i^R, J_j^I$) where L_i stands for the communication latency, T_i represents the minimum period between two consecutive packets releases on the communication flow (equal to the sender tasks period), D_i is the deadline for the communication flow c_i (equal to the sender tasks deadline). In this model J_i^R represents the release jitter for the communication flow and J_j^I represents the interference jitter that denotes the maximum deviation from the period between the successive transmissions of packets.

In order to verify that the timing characteristics of the application model under the NoC-based architecture, we have to calculate the worst-case latencies for both tasks and communication flows. First we calculate the worst-case latency for the task execution, which is used as input to calculate the worst-case latency of the communication flows of the system. Moreover, in both cases we have to check if no deadline is violated upon the completion of the execution of the task or the transmission of the communication flow. Both evaluations are done analytically, using the equations presented in this section.

In the first step we obtain the worst-case response-time for the execution of the tasks executing in each core using classical response-time analysis [6] (Equations (1) and (2)). The use of response-time analysis enables the exact representation of the worst-case response-time for a task and, at the same time, checks if the task meets its deadline.

$$W(i) = I_i + C_i \tag{1}$$

$$r_i = W(i) \tag{2}$$

In Equation (1), I_i is the interference caused by higher priority tasks, which is calculated by Equation (3).

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{W(i)}{T_j} \right\rceil C_j \tag{3}$$

Where *hp(i)* stands for the set of all flows with a priority higher than the flow being analysed.

In the second step a priority window analysis [6] for the communication flow is performed. This communication flow follows the execution of the respective task that generates the flow. The result from the previous analysis is used as input (parameter J_j^R in Equation 4) for this equation. In terms of the equation, we assume that the response-time for the task (r_i in Equation 2) corresponds to the release jitter of the communication flow. The priority window result will include the worst-case response time and therefore will result in the total worst-case response-time for both execution and communication of the task. To calculate the priority window we will use the following equations.

$$W_c(i) = L_i + \sum_{j \in hp(i)} \left\lceil \frac{W_c(i)+J_j^R+J_j^I}{T_j} \right\rceil L_j \tag{4}$$

$$R_i = W_c(i) + J_i^R \tag{5}$$

IV. Heuristics For Task Mapping

Heuristics are relevant pieces of information to solve a specific problem, usually enabling an evaluation of the current state of a possible solution. In this work we are concerned with different factors that affect the temporal behaviour of the system model described in the previous section. Each of these factors is refined into a numeric value in order to provide grounds for the evaluation of the solution. In the following sub-sections we discuss different heuristics that are able to guide the task mapping process.

A. Utilization balance between cores

In the seminal paper by Lyu and Layland [7], it was proven that the rate monotonic model ensures task scheduling without deadline misses up to a given bound based on processor utilization. Thus any utilization balance between cores that achieve this bound for each core will automatically ensure that no deadline miss happens. Even if the utilization balance does not achieve the utilization bound in each core, it will reduce the deadline miss possibility by not letting cores get overloaded.

In order obtain a numeric value that measures how well balanced the systems processors are, the statistic measure of *standard deviation* is utilized. A lower *standard deviation* value represent that the core utilization will be closer to a homogeneous distribution. The utilization *standard deviation* is given by the following equation.

$$\sigma_u = \frac{\sum_{i=1}^{n}(U_i + A_u)}{n} \tag{6}$$

Where U_i stands for the utilization of the core *i*, A_u stands for average core utilization and *n* stands for the number of cores in the system.

B. Quantity of deadline misses

Using an exact feasibility analysis, such as response-time analysis, it is possible to obtain the number of deadlines a given task distribution has missed. This is a straightforward measure since it uses the response-time analysis described previously, and used to check if the whole system is schedulable.

However, changes between task distribution sets may cause subtle improvements that cannot be detected by this heuristic during mapping. This heuristic fails to detect the possibility of future improvements on its resulting mapping, since an improved mapping can be achieved with the same amount of deadlines missed. For example, a task set may take more cycles of the heuristic mapping algorithm in order to reduce the number of deadline misses on a task set.

C. Network stress

The stress imposed on the network by a given task disposition will affect the temporal behaviour of the system directly, where overloaded routers will possibly result in communication flows missing deadlines.

The router utilization behaviour differs from the CPU utilization behaviour because, while the total CPU utilization will always remain the same for the same task set, the utilization sum of all routers will vary according to the task distribution between the cores in the same task set. Due to this property, the use of the *standard deviation* of the router utilization (σ_n) alone would not be able to provide a satisfying evaluation – the router utilization could be balanced but all routers in the NoC would be overloaded. Therefore, the maximum router utilization among all routers in the system (MAX_r) was included in the analysis. The follow equation was created to calculate the network stress measure.

$$S = MAX_r + MAX_r * \sigma_n \quad (7)$$

Utilizing the max router utilization along with the average deviation of the router utilization will imply in the reduction of the max router utilization along with the improvement of the utilization balance. The way this equation was specified, it will give a higher importance to the max utilization reduction because overloaded routers will quite possibly result in deadline misses, utilizing the homogeneity of the utilization distribution among all routers in the system as a untie criteria.

V. Task Mapping using Genetic Algorithms

According to [8], the idea behind the genetic algorithm is to evolve a population of solutions to a certain problem using operators inspired on genetic variety and natural selection. Genetic algorithm uses an analogy to evolutionary biology in order to move inside the solution space. This approach assumes that every solution on the solution space may contain traces of the optimal solution and that the quality of a given solution is directly related to how close a solution is to the optimal solution. Considering each solution as an individual, with the given solution mapped in its genetic code (chromosome) and a measure of how close this solution is to the optimal solution (factor known as fitness).

The algorithm starts with the creation of a random population, a set of *n* individuals representing *n* distinct solutions to the given problem. Each individual belonging to this population will be evaluated in order to identify its fitness. After the identification of the population fitness, the reproduction process takes place, where the fitter individuals will have a higher chance to reproduce.

The reproduction process consists in dividing each individual chromosome in *k* points, separating the chromosome in *k*+1 parts. A new individual will be created inheriting chromosome parts from both parents in an alternating manner. This process will continue between individuals in the current population until there are enough new individuals to fill a new population that will then substitute the current population. After the new population is created, an evaluation of its individuals fitness will begin and the cycle will continue until the stop condition defined by the developer is achieved. In order to enhance the efficiency of the algorithm, some additional notions are used:

- **Elitism**: has the objective of preserving the fittest individuals in a given population, copying these individuals to the new population. This practice aims to preserve good sets of genes in the population, although, it may cause unwanted effects. Copying high fitness individuals to the new population will, in time, reduce the genetic variety of the population, increasing the similarity of the population to the elite individuals in each cycle. In order to restrict this effect, usually, only one individual (the fittest) will be taken from the old population;
- **Mutation:** this practice consists in causing random variations in some individuals of the new population. The mutation occurrence will be dictated by a mutation chance value (measured through a percentage). The mutation aims to inject genetic variety to a population in order to avoid that the population becomes stagnated (all individuals sharing most of the genetic traits). A high mutation chance may compromise the algorithm convergence, nullifying the result of the reproductions. The mutation chance must be calibrated to avoid population stagnation without compromising the algorithm convergence.

In order to address the problem of task mapping using a genetic algorithm, the chromosome has to be modelled in a way that it represents a solution to the problem. For this work, the chosen chromosome is composed of an array of (task, processor) pairs with one fix position for each task in the task set. The array is ordered by task so any position on a given chromosome will represent the same task on any other chromosome. After the chromosome is modelled, certain aspects of the genetic algorithm have to be adjusted for task mapping:

- **Reproduction:** the chromosome previously discussed was modelled in a way that it is guaranteed that, at any given position in one gene represents the exactly same task at the same position in another gene. During reproduction, three random points are chosen, both participant genes are divided in the chosen points and a new individual is created with alternated parts from each one of the participants.
- **Elitism:** elitism was used in the developed algorithm, and it preserves only the fittest individual of the population.
- **Mutation:** the mutation procedure was defined as a reallocation of a random task to a new random processor. The newly resulted mapping is evaluated in both utilization balance and network stress criteria. If the new mapping gets worst in both criteria, the mapping is restored and a new task reallocation is attempted. Since the mutation is focused on improving the solution, a high mutation probability was chosen (25%).

- **Fitness:** one or more of the previously discussed heuristics evaluates the fitness measure of the population individuals.
- **Stop condition:** the stop condition utilized was an arbitrary number of generations produced by the algorithm.
- **Population size:** after tests of convergence and stagnation considering the population for the task set used in this paper (described in the next section), the size chosen for the population was of 100 individuals.

VI. Evaluation

In order to evaluate the genetic algorithm using the heuristics described earlier, we used a synthetic task set composed of 78 tasks. From the 78 tasks that compose the task set, 39 have relevant computational time (producing occupation on the cores) and 39 tasks have no relevant computational time, but are used to communicate with the tasks with relevant computational time, therefore generating communication flows. This particular task set was designed to impose certain difficulties on the task mapping process, including:

- Chunky tasks: the task set has tasks generating 75% of core occupation. This forces the algorithm to isolate them in a given core, since the presence of other tasks on the particular core have a very high probability of generating deadline misses.
- High CPU demand: provided a set of 12 cores, the average core utilization reaches 74.58%.

With this task set, we analysed each heuristic (described in Section IV) for the task mapping over a 4x3 (12 cores) NoC-based architecture. In the evaluation for each heuristic, we executed 10 runs of the genetic algorithm generating 350 cycles of reproductions (generating new populations, where each population is composed of 100 individuals).

The evaluation results are displayed in bar graphics (Figures 2, 3, 4 and 5), dividing the deadline misses into task deadline misses and flow deadlines misses. The bars for task and flow deadline misses were pilled up with the objective of providing a total deadline misses scale for comparison. Each bar (total of 10) represents a run for the genetic algorithm.

A. Results for the utilization balance between cores heuristic

Figure 2 presents the results obtained for the heuristic of utilization balance between cores. Looking at the results, we can see that the mapping results produced a higher number of flow deadline misses in comparison to the task deadline misses. Although this heuristic produced a higher number of flow deadline misses in its test, the flow deadline misses number was in average only 15% higher than the task deadline misses. Considering that the heuristic does not take flows in consideration during the mapping process, this cannot be considered a bad result. The computation balance imposed by this heuristic, in itself reduces the core overload that makes the core tasks have more spare processing time. This causes them to finish the computation earlier, therefore releasing their flows earlier, giving the communication flows more time to arrive to their destination. On this heuristic, the best result achieved a schedulability of 92% of all.

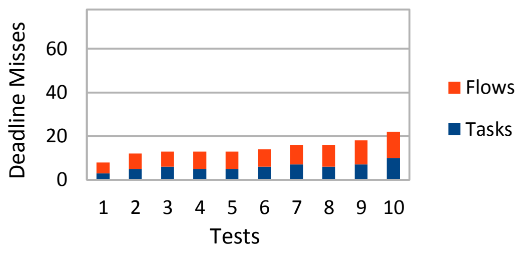

Figure 2. Test results for the utilization balance between cores heuristic

B. Results for deadline misses quantity

Figure 3 presents the results obtained for the heuristic based on deadline misses. This heuristic attacks directly the schedulability problem and, as predicted, it achieves the lower number of deadline misses, achieving 94,9% of total schedulability. However, this heuristic is more computationally demanding when compared to the others. In order to obtain more beneficial changes (that may not generate a lower number of deadline misses in each iteration) this heuristic should be combined with other heuristics capable of tackling communication flows.

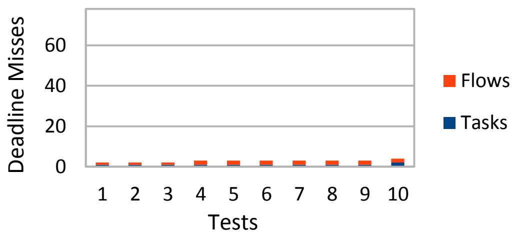

Figure 3. Test results for the deadline misses quantity heuristic

C. Results for network stress

Figure 4 presents the results obtained for network stress. This heuristic showed a high result of deadlines missed, but a balanced number of deadlines missed for both tasks and flows. It would be expected that the heuristic showed a higher number of task deadlines missed, since it ignores altogether the execution of the task. However, by ignoring task execution will cause overloaded cores and these cores will result in tasks missing their deadline by so much that their generated flows will be release late enough to miss their deadlines upon release. This heuristic also showed a wide interval variation on its results. The explanation comes from the balance condition presented on cycles (not letting the core get overloaded on the first swap), where the slightest interaction with another parameter (which could be considerate a interaction with another heuristic) can improve the results of an isolated heuristic. On its bets results, this heuristic obtained a 78,2% of both task and flow schedulability.

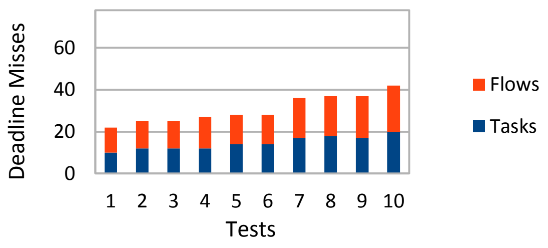

Figure 4. Test results for the network stress heuristic

D. Results for the combination of different heuristics

Figure 5 presents results obtained by combining different heuristics discussed before. In order to achieve better results during the mapping process, we tested several combinations for the heuristics described previously. The best results were achieved with a combination (at several stages) from all the aforementioned heuristics.

Firstly a filter was added on the cycle function using the network stress and occupation balance heuristics. This filter prevents the cycle function to provide new states that do not improve the occupation balance or lower the network stress. Therefore, the cycle function will keep generating new states without returning until an improvement is made in one of these factors.

After a new state is reached, the evaluation function will focus its evaluation on the deadline misses, giving more importance to the deadline misses quantity. This combination of heuristics resulted in a 100% task schedulability and 94,9% flows schedulability. Showing an improvement on comparison to the best results from the heuristics discussed before.

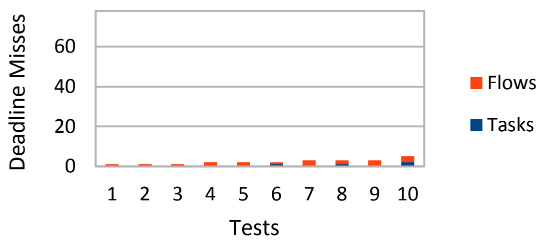

Figure 5. Test results for the combination of heuristics

VII. Conclusions and Final Remarks

The communication between tasks is a factor that impacts directly over the real time temporal behaviour of a multi-processor system. By using a NoC-based architecture, as the one described in this paper, it is possible to use schedulability analysis in order to check if both tasks and flows are schedulable in the system. This is used as the basis to check different task mapping heuristics based on genetic algorithm.

The results provided by the experiments showed that the approach used in this paper is a viable approach to deal with the mapping problem. Although it did not provide a full schedulable mapping, it came really close to full schedulability, which is a good result considering the high CPU occupation provided by the task set (fixed-priority in a partitioned scheduling system) and the presence of chunky tasks.

The implementation process highlighted a specific area where this approach could be improved. The execution time had an average of 30 minutes per run. The schedulability analysis used to implement the quantity of deadlines missed heuristic was the culprit. This is not a problem, since the analysis is off-line, but should be taken into consideration in case of a possible online task mapping.

Moreover, the system model uses a notion of partial dependence, where the release of a communication flow depends on the task finishing its execution. However, the execution start time for the task that will receive this communication does not depend on the communication flow arrival. To simulate this dependence, a deadline was given to the communication flows. This model could be improved to provide a full dependence notion that would increase its analysis. Although, it should be mentioned that this alteration would demand a more complex schedulability analysis increasing even more its computational demand.

References

[1] B. Nikolic, P. M. Yomsi, and S. M. Petters, “Worst-case memory traffic analysis for many-cores using a limited migrative model.” in RTCSA, 2013.

[2] P. Mesidis and L. Indrusiak, “Genetic mapping of hard real-time applications onto NoC-based MPSoCs—A first approach”, ReCoSoC'11, pp. 1–6, 2011.

[3] A. Racu and L. Indrusiak, "Using genetic algorithms to map hard real-time on NoC-based systems", ReCoSoC'12, 1-8, 2012.

[4] M. Norazizi, S. Sayuti, L. Indrusiak and A. Ortiz, "An optimisation algorithm for minimising energy dissipation in NoC-based hard real-time embedded systems", RTNS'13, 3-12, 2013.

[5] L. M. Ni e P. K. McKinley. “A survey of wormhole routing techniques in direct networks”, IEE Computer, vol. 26, issue 2, 62-76, 1993.

[6] Z. Shi e A. Burns. “Schedulability Analysis and task mapping for real-time on-chip communication”, Real-Time System, vol. 46, issue 3, 360-385, 2010.

[7] C.L. Liu, and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”. Journal of the ACM, 20 (1), 1973, pp 46-61.

[8] M. Mitchell. “An introduction to genetic algorithms”. Massachusetts Institute of Tecnology, MIT press, 1998.

[9] R. I. Davis e A. Burns. “A Survey of Hard Real-Time Scheduling for Multiprocessor Systems”. ACM Computing Surveys, vol. 43, no. 4, 35-35, 2011.

[10] L. S. Indrusiak. “End-to-end tests for multiprocessor embedded systems based on network-on-chip with priority-preemptive arbitration”. Journal of Systems Architecture, vol. 60, issue 7, 553-561, 2014.

[11] N. J. Nilsson. “Principles of Artificial Inteligence”. Springer-Verlag, 1982.

[12] N. C. Audsley, A. Burns, M. F. Richardson, A. J. Wellings. “Real-Time Scheduling: The Deadline-Monotonic Approach”. In Proc. IEEE workshop on Real-Time Operating Systems and Software, 1991, 133-37.