Genetic mapping of hard real-time applications onto NoC-based MPSoCs – a first approach

Paris Mesidis, Leandro Soares Indrusiak
Real-Time Systems Group, Department of Computer Science
University of York
York, United Kingdom

Abstract— **Despite its significance to embedded systems industry and research communities, little research has been done on providing guarantees for hard real-time applications running over multicore processors based on wormhole Networks-on-Chip (NoCs). This work takes advantage of recent work on schedulability analysis that is tailored to such platforms, and uses it as a ranking function in a genetic algorithm that is able to evolve task mappings which allow all tasks and communication flows to meet their deadlines in all possible scenarios.**

Keywords-network-on-chip; multiprocessing; task mapping.

I. INTRODUCTION

Network on chip have been introduced recently as a solution to the limitations of complex multiprocessor systems on chip [8]. The performance of a NoC system running a highly parallel application is mostly communication-dominated, therefore the performance of the system also depends on the topological mapping of the application's tasks on the available processors. As different mappings would result in different communication patterns over the network. Performance and cost metrics such as overall execution time, energy consumption and network packet latency depend on the different communication patterns. The mapping problem for NoC-based MPSoCs is to decide how to place each application tasks onto the processing elements (PEs) of the network so that the metrics of interest are optimized.

Applications that are likely to benefit from networks on chip are complex embedded applications with many communicating subsystems and high computing power requirements mainly because their constrains on power consumption and size can be better met by on chip integration. A significant portion of embedded applications have real-time requirements. Applications that fall in the real-time domain have strict timing requirements, meaning that tasks must finish computation in time to meet specific deadlines in every possible situation. Applications that run on NoCs are typically composed of periodic or sporadic tasks that communicate their results by sending messages over the network. For the timing requirements of all the individual tasks to be met the end-to-end latency of the communications between them has to be bounded. A stream of successive packets from the same source heading to the same destination is termed a traffic flow and it is a means of describing messages of various sizes between tasks. Communication deadlines are imposed on traffic flows. In a packet based network the time a packet would take to reach a destination node is the network packet latency. This latency is directly dependent on the distance between communicating nodes and the interference from other traffic flows hence it also depends on the current topological mapping of the tasks onto the processing elements and the resulting communication patterns.

Ever since the introduction of NoCs there has been significant research in the field of task mapping, usually with the goal of optimizing performance and cost. In this paper the proposed approach considers real-time applications where the main objective is to ensure that all the tasks and traffic flows meet their deadlines in every possible scenario. To do that, it applies a genetic algorithm that uses schedulability analysis as its ranking function.

This paper is organized as follows: Section II contains a review of relevant work in literature on which the rest of the work is based. Section III explains the architecture of the NoC platform used in our approach. Section IV explains the proposed genetic mapping algorithm and the last two sections describe the experimental results and conclude on the paper.

II. RELATED WORK

A. Genetic Algorithms

Genetic and evolutionary algorithms in general have been used in a variety of application domains such as engineering, chemistry and finance and are able to find near-optimal solutions even over very large and/or multidimensional solution spaces. A genetic algorithm is an iterative search procedure inspired by the theory of evolution. It usually starts with a population of random solutions and uses a ranking function to keep or discard a part of the population. Then, it uses mutation and crossover operators to generate additional solutions out of those that were kept from the first population. In essence it is a random search guided by a ranking function. In its simplest form a genetic algorithm applies the crossover and mutation operations in a random way. In some cases however these operations can be used in a problem specific manner in order to converge more quickly towards a solution. Genetic and evolutionary algorithms have been applied successfully in the domain of VLSI design [11], [10] and also to the task mapping problem mentioned earlier [9] [15].

B. Task Mapping Algorithms

The task mapping problem can be generally described as the problem of finding a topological placement of the application tasks onto the processing elements (PEs) of the

system, so that the metrics of interest are optimised. For an application that is partitioned into *n* tasks the problem of mapping those tasks on a multiprocessor with *n* PEs would have *n*! possible solutions. The solution space increases factorially with the problem size, so it is infeasible to exhaustively search it.

Static mapping algorithms are used for closed multiprocessor systems, when the application characteristics are known at design time and are not supposed to change during the system's lifetime. They are also used to optimise the initial mapping of tasks in dynamic systems. Because static algorithms run at design time their running time is not a limiting factor and different heuristics have been proposed in the literature also considering different objective functions.

In [12] Hu and Marculescu propose a branch-and-bound algorithm for the mapping problem which aims to minimize the total communication energy and ensure that the application traffic does not exceed the available bandwidth on the links so that all the traffic flows can be serviced. The solutions in this approach are evaluated using a specific energy model. According to that model the energy spent sending a traffic flow is proportional to the number of bits in the flow and the distance between the communicating nodes, measured in hops (distance between adjacent routers).

In [13] a similar approach is used aiming to minimize the communication energy over a 2D mesh NoC with XY routing. Here two greedy algorithms produce solutions which are used as seeds to a simulated annealing algorithm. Among different heuristics genetic algorithms have been used as well. This is the case in [15] where a two stage genetic algorithm is used with the objective of optimizing the execution time of the application by minimizing the time spent on communication however the effects of network contention between traffic flows are not taken into consideration because a platform with adaptive routing is used. Similarly in [14] a genetic algorithm has been used, but in this case the algorithm tries to optimize multiple objectives at once by searching a Pareto front of solutions and trying find a solution from the optimal set. The NoC architecture in this case is a 2D-mesh with static XY routing and wormhole switching. It is abstracted as a graph that contains information about the functional behavior of each element in the NoC as well as timing and power consumption parameters, while the applications are either statistically synthesized or come from real traces. Solutions are evaluated using simulations instead of analytical models at the cost of simulation time and the simulation framework is capable of evaluating many different metrics such as energy and processing time. In this approach the genetic algorithm is not entirely random; the genetic operations are applied on random chromosomes/solutions of each population the mutation operations is modified so that it reduces the hop count between two communication tasks in order to be more effective guiding towards a solution faster.

Most of the works in literature try to minimise the energy spent on inter task communication and improve performance by reducing congestion on the networks links and packet latencies. This work tries to find a mapping where regardless of these cost and performance metrics all the traffic flows will meet their respective deadlines.

C. *Schedulability in On-chip Multiprocessors*

Motivated by the need for NoC platforms that can provide real-time services, a schedulability analysis approach was presented in [1] based on which the worst case network packet latency can be calculated. It can be applied to NoCs with wormhole switching, which are famously hard to predict. However, it is constrained to wormhole NoCs with deterministic routing and priority-preemptive virtual channels.

In priority pre-emptive virtual channels, if any number of traffic flows compete for the same link, the one with the highest priority gains access to the link while all others are blocked. Because of this fact, it is possible to calculate the maximum amount of time a particular flow will have to wait before it can transmit its payload completely.

The schedulability analysis from [1] assumes that all traffic flows τ_i are either periodic or sporadic and have a period T_i, they are assigned a priority P_i and have a deadline D_i. Another parameter that is defined for traffic flows is the release jitter J_j^R which is the maximum deviation of the release of packets from the period of their traffic flow. Another important attribute of a traffic flow is the basic network packet latency C_i, which is the time a packet of a given traffic flow would take to reach its destination node when no contention occurs. Basic latency is calculated by:

$$C_i = \left\lceil \frac{Li}{f} \right\rceil \cdot \frac{f}{B} + H \cdot S \quad (1)$$

Where L_i is the packet size, f is the flit size, B is the link bandwidth. The distance the packet has to traverse is H number of hops and S is the processing time each router takes.

Each traffic flow in the system will be assigned a specific route generated by a routing algorithm. The route of a traffic flow τ_i is a set of links β_i. If a traffic flow's route β_i shares at least on link with another flow's route β_j then the two traffic flows τ_i and τ_j have a direct competing relation in which case the higher priority flow will pre-empt the lower priority one. For a given traffic flow, the direct interference set S_i^R is defined as the set of all the traffic flows that have a direct competing relation with τ_j and have higher priority.

In the case where two flows τ_i and τ_j do not share any links, but there is at least one traffic flow τ_k that has a direct competing relationship with both of them and its priority is such that $P_j > P_k > P_i$ then P_i will have an indirect competing relationship with P_j. An indirect interference set S_i^I can be defined so that it contains all the traffic flows that have an indirect competing relationship with τ_i.

The analysis proposed in [1] calculates the worst case network latency R_i for each traffic-flow The upper bound of the network latency depends both on the basic maximum packet latency and the time a flow is blocked due to interference from higher priority traffic flows.

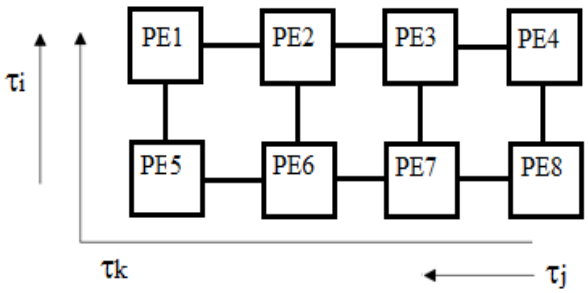


Figure 1. Indirect interference between τ_i and τ_j where $P_j > P_k > P_i$.

Under the assumption that for all traffic flows $D_i < P_i$ the network latency upper bound R_i for a traffic flow τ_i is given by:

$$Ri = \sum_{\forall \tau j \, \epsilon S_i^R} \left\lceil \frac{\mathrm{Ri} + \mathrm{Rj} + J_j^I - Cj}{\mathrm{Tj}} \right\rceil Ci + Cj \quad (2)$$

For further details, we refer to [1], but the main intuition is that the worst case latency of a traffic flow depends on the distance between its source and destination and on the interference it may experience from higher priority traffic flows, which in turn depends on the communication patterns over the network. The correctness of this analysis is proved in [1] and is also backed up by simulation results in [2].

III. Interconnect Architecture

Several NoC architectures can provide guaranteed throughput to some of the transmitted packets, while providing a best effort service to all other packets. In this paper, we are interested in applications with hard real-time constraints, so we can expect them to produce a significant number of packets that must meet their deadlines even in the worst case scenario. Latency guarantees should be the rule, and not the exception. Therefore, we adopt wormhole NoCs with priority pre-emptive virtual channels. Such NoCs provide a good trade-off between latency guarantees and hardware overhead. By 33 priorities to packets, and by allowing high priority packets to pre-empt the transmission of low priority ones, network contention becomes predictable. In such architecture, the schedulability of packets can be analysed through the approach presented in [1] and the actual packet latencies can be accurately obtained using transaction level simulation models [2].

Fig. 1 shows the internal structure of a NoC router using such architecture, which is similar to QNoC [3] and HERMES [4]. 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 [5] guarantees that data is only forwarded from a router to the next when there's enough buffer space to hold it.

At any time, a flit of a given packet will be sent through its respective output port if it has the highest priority among the packets being sent out through that port, and if it has credits (that is, buffer space on the respective buffer of the neighbouring node connected to that output port). If the highest priority packet can't 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 architectures such as [6] and [7], where GT traffic gets a pre-assigned time-slot to use resources. Priority pre-emptive arbitration does not unnecessarily reserves resources, so low priority traffic can always use the NoC if there are no requests from GT flows.

For the experiments in section V, we adopt a mesh topology and XY deterministic routing, but the proposed mapping technique can be applied to other topologies and routing algorithms.

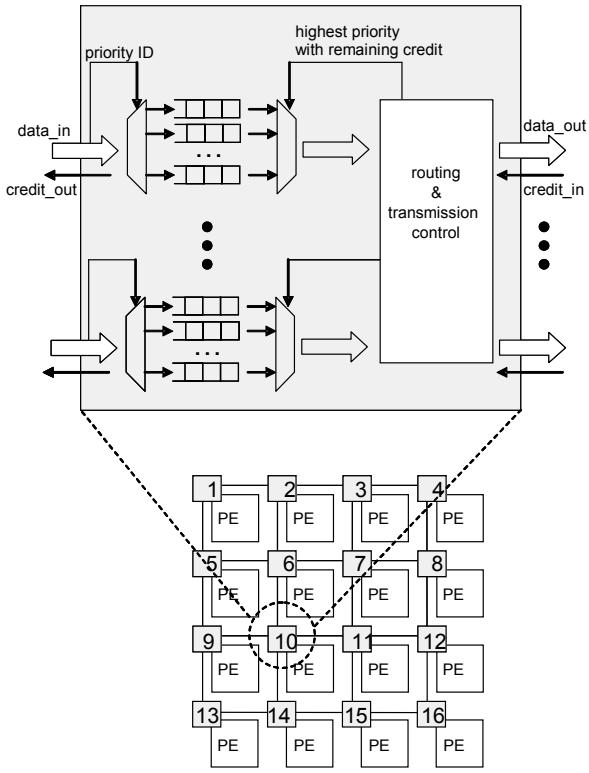


Figure 2. NoC architecture with detail of the router structure.

IV. Genetic Mapping of Hard Real-Time Applications

The worst case latency of a traffic flow depends on the distance it has to travel and on the interference it may experience. The direct and indirect interference that occurs between flows depends on the current communication pattern which is a result of the current topological mapping of the application's tasks onto the PEs.

Motivated by this fact we employ a genetic algorithm with the objective of finding a mapping where all the traffic flows in the application are able to finish transmission before their respective deadlines even in the worst case scenario. The inputs to the proposed algorithm are the models of the application and of the NoC-based platform. The application

model describes its tasks with attributes such as priority, period, computation time, deadline and release jitter. Tasks communicate with each other through traffic flows, which inherit some attributes of the tasks that generate them (period, priority, deadline) and also have their own attributes, such as the amount of data they transmit and the release jitter that they suffer (which is in derived from the worst case execution time of the respective task). The attributes of the traffic flows are then used to carry out the worst case latency analysis for each different mapping.

The model of the NoC platform describes its topology and major components: processing elements, routers, buffers, links. In this paper, we consider only 2D mesh topologies with homogeneous processing, but nothing prevents the application of the proposed approach to other architectures. A solution to the mapping problem is an assignment of all tasks of the application model to processing elements of the platform model.

Each solution in the algorithm can be denoted by the pairing of tasks and processing elements in a many-to-one cardinality (a task can be assigned to only one processing element, which can also be mapped other tasks). Each solution can be ranked by the number of traffic flows that cannot meet their respective deadlines under this mapping. An optimal solution should have a score of zero. We use the analysis presented in [1] as our ranking function: the score of each mapping is calculated using equation (2) for every traffic flow. In case a mapping with score 0 is found, which means that all traffic flows meet their deadlines under that mapping, the algorithm halts. The genetic operations are purely random: the crossover operation combines two mappings to produce a new one and the mutation operation just perturbs an existing mapping.

The algorithm begins by generating a population of random mappings. The selection step of the algorithm calculates the score of each mapping and the average score of all the mappings in the current generation, it then discards all the mappings with score higher than average. Next the current generation is repopulated using crossover. Finally each member of the population is mutated with a probability of 50% (similar to mutating a fixed percentage of the population). The mutation operation in this instance is just a pair-wise mapping swap of two tasks picked at random. These steps are iterated until a suitable solution is found.

```
1   Generate an initial population of random mappings of size n
2   while (!solutionFound){    // mapping with no deadline misses
3           for all mappings in population{
4               if  (mapping.DeadlineMisses >AvgDeadlineMisses)
5                       {Remove mapping}}
6           while(populationSize!= initialSize) {
7               Mapping m1 = random mapping from population
8               Mapping m2 = random mapping from population
9               Mapping m3 = randomCrossover(m1,m2)
10              population.add(m3) }
11          for all mappings in population{
12              mutate with probability 50% }
13          evaluate(population)   //calculate the score of mappings
14  }
```

Listing 1. Pseudo-code of the genetic algorithm.

The current approach was chosen because it was relatively straightforward to implement in addition to the fact that the computation time is not a limitation in static mapping. As a first approach, the proposed algorithm applied genetic operators randomly as this will help with comprehending the problem complexity and the size of the solution space and will give us a point of reference to compare with more sophisticated algorithms.

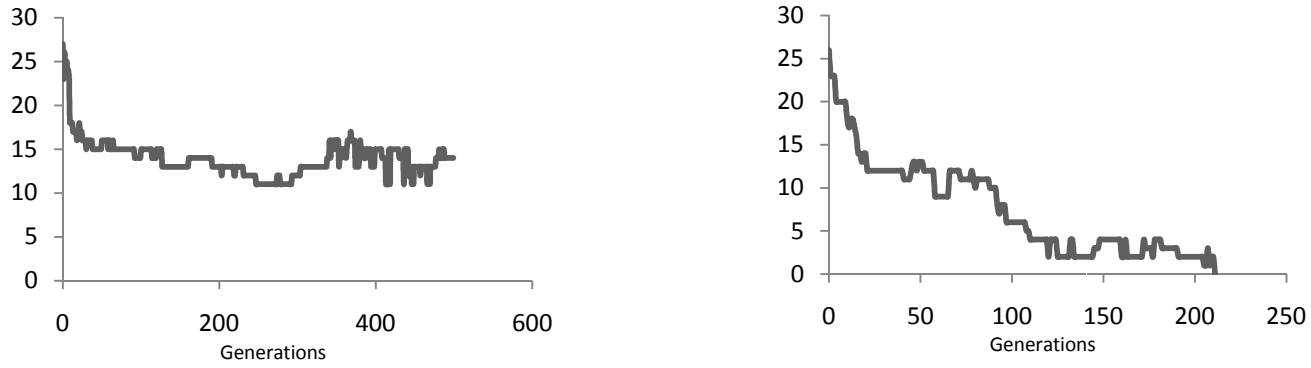
V. EXPERIMENTAL RESULTS

In order to validate the proposed approach we developed a framework that implements both application and platform models described earlier, as well as the schedulability analysis from [1]. Unlike most of the related work on task mapping our approach did not require cycle accurate simulation as it is based on an analytical model. The software framework provided us with a NoC system model where an application can be mapped onto.

Such model describes the topology of the NoC using collections of PEs and links on a 2D coordinate system interconnected to each other. It also implements a routing function. This system model provides all the information needed to carry out the analysis described in the previous sections.

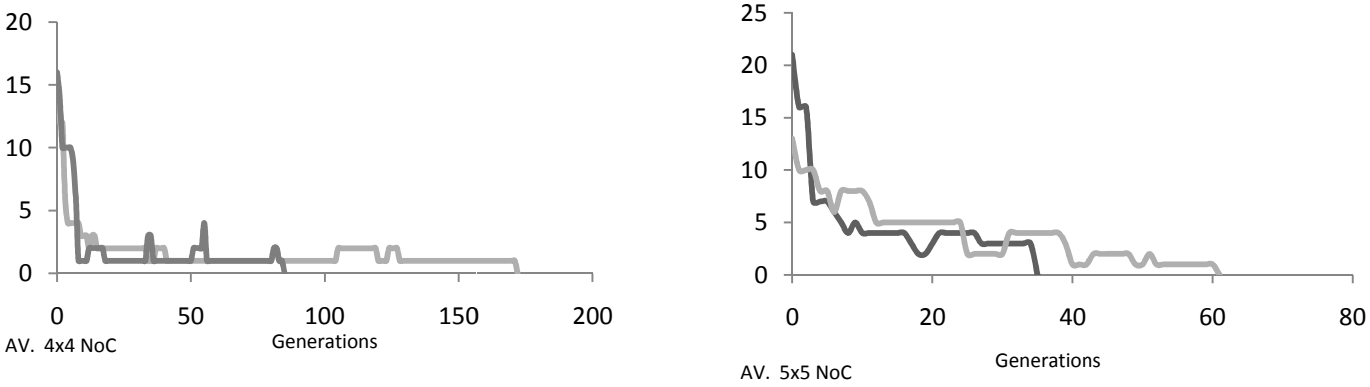
Two application models were used for this case study. The first one is the controller of an autonomous vehicle, which is a good example of a highly parallel application running on an embedded system. This application was introduced in [2] and it is very communication intensive, because of the successive processing of different video streams for stereophotogrammetry and visual odometry. The subsystems of the application are those of video processing, navigation and stability control. In total it comprises of 33 tasks and 38 inter-task fixed-priority communication flows. Each traffic flow has all the attributes mentioned in the previous sections such as period and deadlines. In addition all the traffic flows have fixed-size payloads, ranging from 7kbits to 525kbits. A second application was used which was completely synthesized. It was composed of more tasks and traffic flows hence it would be harder to find a solution for a same size NoC. The synthetic application has 50 tasks and 40 inter-task communication flows again with all traffic flows having fixed-size payloads, ranging from 42kbits to 2400kbits, and shorter periods so that this application is much more communication intense than the previous one.

The following figures show the results obtained from running the proposed algorithm with the applications and NoC-based platform described previously. The graphs show the minimum (best) score achieved in every iteration of the algorithm. The solutions tend to get closer to optimal as the algorithm runs. We applied the algorithm with a population size of 100 solutions. The algorithm tried to map the autonomous vehicle and the synthetic applications on 5x5, 4x4, 4x3 and 3x3 NoC platforms. The algorithm was executed 15 times for each scenario, and each graph shows the quickest and the slowest run of the algorithm which could find a solution.



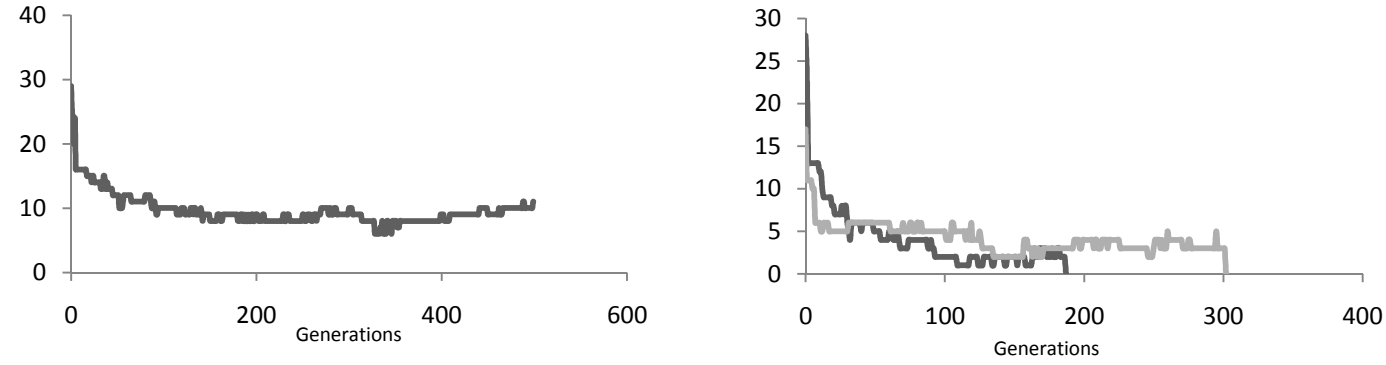
(i) (ii)

Figure 3. Mapping of Autonomous Vehicle application onto a NoC platform with topologies (i) 3x3 and (ii) 4x3. Vertical axis shows the number of unschedulable traffic flows of the best mapping of each generation of the genetic algorithm execution (only the best of 15 executions is shown).



(i) (ii)

Figure 4. Mapping of Autonomous Vehicle application onto a NoC platform with topologies (i) 4x4 and (ii) 5x5. Vertical axis shows the number of unschedulable traffic flows of the best mapping of each generation of the genetic algorithm execution (dark gray line shows the fastest of 15 executions, light gray line shows the slowest execution that succeeded to find a solution).



(i) (ii)

Figure 5. Mapping of Synthetic application onto a NoC platform with topologies (i) 4x4 and (ii) 5x5. Vertical axis shows the number of unschedulable traffic flows of the best mapping of each generation of the genetic algorithm execution (dark gray line shows the fastest of 15 executions, light gray line shows the slowest execution that succeeded to find a solution).

It can be seen that for each application, it becomes harder to find a mapping as the platform gets smaller, a fact that is expected because the interference between traffic flows increases. The algorithm was allowed to run to a maximum of 500 generations, hoping that this would allow the algorithm to visit enough solutions. The algorithm would also run for 15 times for each scenario so as to make the overall solution pool independent on the initial random population's quality.

The synthetic application, which was more communication-intensive, could only be mapped on a 5x5 NoC. The autonomous vehicle application was mapped successfully on a 5x5, 4x4 and a 4x3 NoC but could not be mapped on a 3x3 platform by this algorithm. In the case of the 4x3 NoC the algorithm found a solution only once in 15 runs.

VI. CONCLUSIONS

In this paper we proposed an approach for mapping hard real time applications on NoCs aiming ensure that all the traffic flows meet their deadlines in all possible scenarios. Since this was a first approach to this problem, we chose to use a straightforward evolutionary algorithm that uses an analytical method as a ranking function. This algorithm, as the results show, was able to find optimal solutions in some cases and could be used as a reference point for future research. However, there is no guarantee that there are no existing solutions for those cases where the proposed algorithm could not find one (such as mapping the autonomous vehicle application onto a 3x3 NoC or the synthetic application onto a 4x4 NoC).

Some points of improvement that are currently underway include the use genetic operations that also manipulate the priorities of the traffic flows, as well as the application of different genetic operations that may lead to a faster convergence. Future research will mainly focus on developing a mapping algorithm that will specifically try to minimize the interference between the traffic flows.

REFERENCES

[1] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, 2008, pp. 161-170.

[2] Zheng Shi, Alan Burns, Leandro Soares Indrusiak: Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching. IJERTCS Vol. 1Issue (2): pp.1-22. 2010

[3] L. S. Indrusiak and O. M. dos Santos, "Fast and Accurate Transaction-Level Model of a Wormhole Network-on-Chip with Priority Preemptive Virtual Channel Arbitration," in *Proc Design Automation and Test in Europe (DATE)*, Grenoble, France, 2011, pp. 1089-1094.

[4] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, no. 2-3, pp. 105-128, Feb. 2004.

[5] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual channels in networks on chip: implementation and evaluation on Hermes NoC," in *Proc 18th Symposium on Integrated Circuits and Systems Design*, 2005, pp. 178-183.

[6] T. Bjerregaard and J. Sparso, "Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip," in *Norchip Conference, 2004. Proceedings*, 2004, pp. 269-272.

[7] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414-421, 2005.

[8] M. Schoeberl, "A Time-Triggered Network-on-Chip," in *Int Conf on Field Programmable Logic and Applications,* 2007, pp. 377-382.

[9] L. Benini, G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, n. 1, pp. 70–78, 2002.

[10] T. Lei and S. Kumar, "A two-step genetic algorithm for mapping task graphs to a network on chip architecture," in Euromicro Symposium on Digital Systems Design, 2003.

[11] J. Lienig, "A parallel genetic algorithm for performance-driven VLSI routing," *IEEE Trans Evolutionary Computation*, vol 1, n. 1, pp. 29–39, 1997.

[12] P. Mazumder and E. M. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall, 1999.

[13] J. Hu and R. Marculescu, "Energy- and performance aware mapping for regular NoC architectures," *IEEE Trans CAD Integrated Circuits*, vol.24, no. 4, pp. 551–562, 2005.

[14] César A. M. Marcon, Edson I. Moreno, Ney L. V. Calazans and Fernando G. Moraes, "Evaluation of Algorithms for Low Energy Mapping onto NoCs," in *Proc IEEE International Symposium on Circuits and Systems*, 2007.

[15] Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi, "A Multi-objective Genetic Approach to the Mapping Problem on Network-on-Chip," *Journal of Universal Computer Science*, vol. 12, n. 4, 2006, pp. 370-394.

[16] Tang Lei, Shashi Kumar, "Algorithms and Tools for Network on Chip Based System Design," in *Proc 16th Symposium on Integrated Circuits and Systems Design*, 2003, pp. 163-168.