End-to-End Schedulability Tests for Multiprocessor Embedded Systems based on Networks-on-Chip with Priority-Preemptive Arbitration

Leandro Soares Indrusiak
Real-Time Systems Group - Department of Computer Science
University of York - York, United Kingdom
lsi@cs.york.ac.uk

ABSTRACT

Simulation-based techniques can be used to evaluate whether a particular NoC-based platform configuration is able to meet the timing constraints of an application, but they can only evaluate a finite set of scenarios. In safety-critical applications with hard real-time constraints, this is clearly not sufficient because there is an expectation that the application should be schedulable on that platform in all possible scenarios. This paper presents a particular NoC-based multiprocessor architecture, as well as a number of analytical methods that can be derived from that architecture, aiming to allow designers to check, for a given platform configuration, whether all application tasks and communication messages always meet their hard real-time constraints in every possible scenario. Experiments are presented, showing the use of the proposed methods when evaluating different task mapping and platform topologies.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems.

General Terms

Performance, Design.

1. INTRODUCTION

Embedded systems typically have to fulfil timing constraints that are related to their application domain and usage scenarios. Constraints are usually specified as the deadline to perform specific functions. For example, a high-definition video recorder must be able to capture, compress and store 25 video frames per second. In safety-critical applications, such constraints are said to be hard real-time constraints, as there is an expectation that they have to be met by the system in all possible scenarios. Therefore, embedded systems designers must be able to evaluate which design alternatives can fulfil those constraints and, for safety-critical applications, guarantee real-time behaviour.

In this paper, we present analytical methods to evaluate whether a multicore embedded system based on a Network-on-Chip (NoC) can fulfil all its timing constraints. A NoC-based system can have tens to hundreds of processing cores interconnected by an on-chip packet-switching network that allows data to be transferred between the local caches of each core and from/to external memory. Section 2 of the paper provides more detail on this type of system architecture. It will then become clear that the performance of the NoC interconnect is as critical as the performance of the processing cores when it comes to meet timing constraints.

Throughout this paper, we will use the terms *end-to-end timing constraint* or *end-to-end deadline* of an *application task-chain*. Those terms denote constraints derived from the application domain (e.g. every video frame must be processed in 40 ms or less) that must be met by specific components of the application (i.e. chains of communicating tasks). Our goal is to establish whether all task-chains of an application have their end-to-end deadlines met by a particular NoC-based platform configuration, and this problem is referred in this paper as *end-to-end schedulability test*. Such test must consider the *end-to-end latency* of each task of a task-chain: the time it takes for a processing core to execute that task (*computation latency*) plus the time it takes for the NoC to transfer all data produced by that task to the next one on the chain (*communication latency*). In Section 3, precise definitions of all those concepts will be given, followed in Section 4 by formulations of end-to-end schedulability tests that are tailored to NoC-based multicores with priority arbitration.

Some of the schedulability tests presented in this paper are based on classic Response Time Analysis (RTA) [1] and on NoC traffic flow schedulability analysis [2]. Individually, those analyses cannot be used to evaluate and improve the schedulability of a NoC system. For example, the traffic flow schedulability analysis from [2] has been used in [3] to produce fully schedulable task mappings, but authors had to artificially limit the number of tasks mapped to each core, as the analysis does not directly consider the different interference patterns resulting from mapping the source of the traffic flows to different cores. Without a limitation on the maximum number of tasks per core, the mapping optimisation process would lead to solution with all tasks mapped to the same core (so all communications are local, instantaneous and therefore schedulable). Likewise, the evaluation of NoC schedulability using only RTA would be oblivious to the delays caused by network contention. Therefore, in this paper we discuss how to compose those two analytical methods to achieve correct upper bounds to the end-to-end latency, and show that the resulting analytical model is useful as a test to evaluate whether a specific task mapping is schedulable.

Schedulability tests are not always used in industry and academia. Often, system designers address the schedulability problem by simulating the system under different scenarios and checking if the obtained figures for computation and communication latencies meet the constraints. There are two main limitations to that approach. Firstly, for a complex multicore embedded system, the simulation of a few seconds of an application's execution may take hours or days [4], limiting the number of design alternatives

that can be evaluated and the portion of the application lifetime that can be considered. Secondly, simulation can only verify whether constraints are met within the scenarios that are explicitly simulated. In complex embedded systems, the set of possible scenarios is too vast to be exhaustively covered, so it is not possible to check whether constraints are always met. For example, if application tasks can suffer release jitter, it would be necessary to simulate each and every possible value of jitter for each task in order to make sure that the timing constraints are met in every case. In Section 5, we use a number of benchmarks to evaluate the proposed schedulability tests, we compare the obtained figures with those obtained with simulation, and propose a design flow that benefits from the joint use of both techniques.

2. NOC-BASED MULTICORES

NoCs are a common architectural template for processors with dozens of cores, and it has the potential to scale with the increase of the core count up to hundreds or thousands. Figure 1 shows a simplified representation of a NoC architecture. It has 16 cores, each of them represented together with their own local cache as a white rectangle. Cores are directly connected to NoC switches (grey rectangles), which route data packets towards a destination (which may be another core, an interface to off-chip memory, a custom hardware accelerator, etc.).

Many components of the NoC template can be parameterized to better meet design goals, such as the number and type of cores, buffering, routing and arbitration policies, among others. In this paper, our choice of a specific subset within such a large number of alternatives was based on three criteria: (i) adopt architectural features that are widely used in industry and academia, (ii) use on-chip resources efficiently, and (iii) privilege techniques that are amenable to the type of schedulability tests we are investigating.

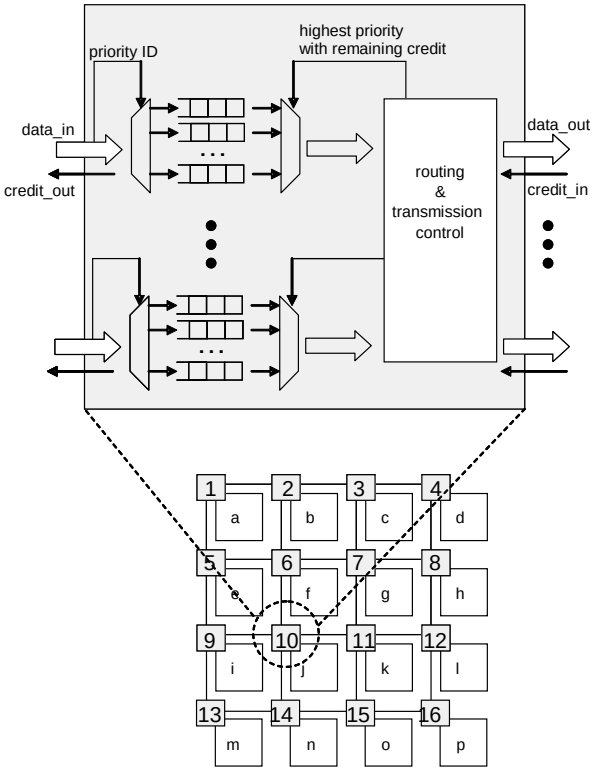

Figure 1. NoC architecture with detail of the router structure

Following criterion (i), we concentrate on the widely used 2D mesh topology [5][6][7][8]. Criterion (ii) motivates the use of wormhole switching, as its buffer overhead is much smaller than store-and-forward (SAF) approaches, and its link allocation is more efficient than circuit switching approaches: there is no need to reserve the complete path of a packet, and NoC links are only allocated on the segments of the path where there is data ready to be transferred. Finally, criterion (iii) requires some level of predictability on resource sharing policies, so we limit our approach to NoCs with non-adaptive routing and priority arbitration such as QNoC [7] or Hermes [9]. The most common implementation of priority arbitration is based on virtual channels (VCs) [10], which allow packets with higher priority to preempt the transmission of low priority ones, making it easier to predict the outcome of network contention scenarios. Figure 1 shows a detailed view of a NoC switch with priority-arbitrated VCs: in each input port, a different FIFO buffer stores data words (flits) of packets arriving through different VCs (one for each priority level). The routing component assigns an output port for each incoming packet according to their destination. A credit-based approach [10] guarantees that data is only forwarded from a router to the next when there's enough buffer space to hold it at the right VC. 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.

3. SYSTEM MODEL AND NOTATION

In this paper, we investigate ways to determine whether application tasks executing and communicating over a specific NoC-based multicore can meet all application-specific timing constraints. Therefore, we need a system model that covers the application as well as the NoC-based platform and its configurations.

For the application model, we recall the sporadic task model and define an application as a taskset $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ where each task τ_i is a 6-tuple $\{C_i, T_i, D_i, J_i, P_i, \varphi_i\}$ indicating respectively its worst case computation time, period (i.e. minimum inter-release time interval), deadline, release jitter and priority. The sixth element of the tuple is the only proposed addition to the sporadic task model, and represents a communication message sent by τ_i. Our initial assumption is that each task produces a single message φ_i which is sent immediately after it finishes its computation. The message is defined as a 3-tuple $\{\tau_d, Z_i, K_i\}$ representing its destination task, size and maximum release jitter. A task-chain $X = \{\tau_1, \tau_2, \ldots, \tau_x\}$ is an ordered subset of Γ where each task sends a message to the subsequent task in X, and all of them have the same period T_x. We assume that all task-chains in a particular application Γ are disjoint subsets of Γ, and that loops are not allowed (i.e. the sixth element of the tuple of the final task τ_x of every task-chain is the empty set ø).

The model of the NoC platform is composed of a set of processing cores $\Pi = \{\pi_a, \pi_b, \ldots, \pi_z\}$, a set of switches $\Xi = \{\xi_1, \xi_2, \ldots, \xi_m\}$, and a set of unidirectional links $\Lambda = \{\lambda_{a1}, \lambda_{1a}, \lambda_{12}, \lambda_{21}, \ldots, \lambda_{zm}, \lambda_{mz}\}$. Links can connect cores to switches, or switches with each other, allowing for all possible direct and indirect NoC topologies. For example, the architecture shown in Fig. 1 has 16 cores $\pi_a \ldots \pi_p$, each of them connected to one of the 16 switches $\xi_1 \ldots \xi_{16}$ via two unidirectional links (e.g. λ_{a1} and λ_{1a}). The switches, in turn, are

connected to each neighbouring switch by two links (e.g. λ_{21}, λ_{12}, λ_{23}, λ_{32}, λ_{26} and λ_{62} are the links attached to switch ξ_2).

NoCs forward packets from source to destination according to a routing algorithm. We define a function $route(\pi_a, \pi_b) = \{\lambda_{a1}, \lambda_{12}, \ldots, \lambda_{mb}\}$ denoting the subset of Λ used to transfer packets from core π_a to core π_b. A route will include links connecting the source and destination cores to their respective switches, and all the links between switches along the way. The cardinality of a route is defined as $|route(\pi_a, \pi_b)|$ and will be informally referred as its hop count. For the example in Fig. 1, $route(\pi_e, \pi_g) = \{\lambda_{e5}, \lambda_{56}, \lambda_{67}, \lambda_{7g}\}$ and $|route(\pi_e, \pi_g)| = 4$ for most commonly used routing algorithms.

Task mapping is a critical part of the design of multicore systems. It defines which application tasks should be mapped onto which processing core (i.e. on which core each task will execute). Many different approaches to task mapping have been proposed, taking into account the time when the mapping occurs, whether tasks can be remapped (or migrated) during execution, and which metrics should be considered when making a mapping decision. We therefore define a surjective function $map(\tau_i) = \pi_a$ to denote the core onto which a task is mapped. Its inverse is defined as $map^{-1}(\pi_a) = \{\tau_i, \ldots, \tau_n\}$ and represents the tasks mapped to a given core. Likewise, the mapping of a message $map(\varphi_i) = route(map(\tau_i), map(\tau_d))$ denotes the route of its packets over the NoC, and the inverse $map^{-1}(\lambda) = \{\varphi_i, \ldots, \varphi_n\}$ represents the messages mapped over a given link.

Once the mapping of all tasks of Γ is defined, it is possible to calculate the basic communication latency L_i of every message φ_i. It represents the time taken by the message to be completely transferred from its source to its destination, assuming no contention over the NoC links (i.e. as if the message is the only one using the NoC). The actual value of L_i will depend on implementation-specific characteristics of the NoC (e.g. link width, time required for a packet header to cross a router, and for a flit to cross a link). A common formulation is the following: $L_i = |map(\varphi_i)| \cdot l_{hop} + (|map(\varphi_i)| - 1) \cdot l_{router} + (Z_i / width) \cdot l_{hop}$, where the first term represents the time it takes for the packet header to traverse all the NoC links, expressed as the product of the message hop count and the latency l_{hop} for the header to traverse a single link; the second term represents the time it takes for the packet header to traverse all NoC routers, and is expressed as the product of the number of routers along the path (which is usually the number of hops minus one in most direct networks) and the latency l_{router} for the header to traverse a router; the third term represents the time taken by the packet payload to follow the header in a wormhole fashion all the way to the destination, expressed by the message length Z_i (in bits) divided by the link width (which results in the number of payload flits), multiplied by the single link latency l_{hop}.

4. END-TO-END SCHEDULABILITY TESTS FOR NOC-BASED MULTICORES

A schedulability test is able to discern system configurations that are schedulable, that is, able to meet their timing constraints even in the worst case scenario. In this paper, we assume that a system is schedulable *iff* all its task chains meet their end-to-end deadlines. To check this property, we first revisit a number of existing techniques that can be used as necessary schedulability tests.

4.1 Schedulability of tasks over a processing core

A processor utilisation test can be used to check whether all tasks mapped to a particular core π_a do not exceed its capacity:

$$\sum_{\tau_i \in map^{-1}(\pi_a)} \frac{C_i}{T_i} \leq 1 \quad \textbf{(1)}$$

This test is necessary but obviously not sufficient because even though the core π_a may be capable to run all the tasks, it may not be able to run all of them within their deadlines. Response Time Analysis (RTA) [1] is the standard technique to evaluate how much the interference from higher priority tasks can delay the completion time of task τ_i:

$$R_i = C_i + \sum_{\forall \tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad \textbf{(2)}$$

where the function $hp(\tau_i)$ denotes the set of all tasks that can preempt τ_i: those mapped to the same core and that have a higher priority. Formally, $hp(\tau_i)$ includes every task $\tau_j \in \Gamma$ where $map(\tau_i) = map(\tau_j)$ and $P_i < P_j$. With the help of Eq. 2, it is possible to calculate the longest possible time interval between the release of τ_i and its termination. This is done by adding τ_i's computation time C_i and the computation times C_j of all releases of tasks τ_j that could preempt it. The result of that sum is referred as τ_i's worst case response time and is represented by R_i. As R_i appears in both sides of Eq. 2, an iterative solution was proposed in [1]. RTA has been widely used to test schedulability of uniprocessor and statically mapped multiprocessor systems with fixed priorities.

More advanced tests have been reviewed in [11], considering more advanced task models that support task migration (global scheduling), dynamic priorities and different constraints on deadlines. However, the tests described and referenced above do not explicitly consider inter-task communication. Instead, most assume that all communication latencies can be combined with the worst case computation time C_i of the respective tasks. For uniprocessor systems with uniform memory access times, such assumption can be acceptable as the communication overhead can be predictable and usually small compared with the computation time. In NoC-based systems, however, the communication latency introduced by the NoC when tasks access memory or exchange messages depends heavily on the task mapping, the application communication patterns and resulting network congestion (which is particularly hard to predict in the case of wormhole switching NoCs). This leads to high variability in communication latencies, which can be of the same order of magnitude of the computational time C_i of the tasks (or even higher). Therefore, we make a case to explicitly consider communication times when analysing schedulability of NoC-based systems.

4.2 Schedulability of packets over a NoC and end-to-end schedulability of communicating tasks

To address the schedulability of packets transmitted over a NoC, we rely on the work proposed by Shi and Burns [2], which in turn builds on RTA. Their work assumes that packets are released into the NoC sporadically, i.e. a series of packets (referred in [2] as a *traffic flow*) has a minimum inter-release interval which is known at design time. The maximum size of each packet is also known a priori. On the platform side, the main assumption is that the NoC

routers perform deterministic routing, and that the link arbiters can preempt packets when higher-priority packets request the output link they are using. Such assumption is valid for the type of NoC architectures described in Section 2. The worst case latency S_i of a packet transmitted over such a NoC can be found using Eq. 3, which has been rewritten from the original in [2] to follow the notation presented in Section 3. To simplify the notation, we assume that there is a one-to-one relationship between application messages and packets sent over the NoC, and therefore use the same symbol φ for both.

$$S_i = \sum_{\forall \varphi_j \in \text{interf}(\varphi_i)} \left\lceil \frac{S_i + K_j + K_j^I}{T_j} \right\rceil L_j + L_i \qquad (3)$$

The function *interf*(φ_i) denotes the direct interference set of φ_i, which is the set of all packets that can preempt φ_i, which are those whose routes at have at least one NoC link in common with φ_i's route and that have higher priority. Formally, *interf*(φ_i) includes every packet φ_j where *map*(φ_i) ∩ *map*(φ_j) ≠ ∅ and $P_i < P_j$. The intuition behind Eq. 3 is similar to what was presented for Eq. 2. The value of S_i can be found by adding φ_i's basic latency L_i and the latencies L_j of all releases of packets φ_j that could preempt it. The same iterative solution proposed in [1] can be used here.

It is worth noticing that the release jitter of φ_j can influence how many times it can preempt φ_i. In Eq. 3, we consider two types of release jitter: K_j which is caused by the execution of the task τ_j that releases φ_j, and K_j^I which is caused by indirect interference (i.e. packets that can preempt φ_j but cannot interfere on φ_i because they don't share any links, see [2] for a detailed definition).[2][1]

Since the value of K_j must be the maximum amount of time elapsed between the start of φ_j's period and its actual release, and since we have defined that a packet is released immediately after its respective task has finished computation, we can clearly state that $K_j = R_j$. Finally, from [2] we have that $K_j^I = S_j - L_j$.

Thus, the worst case end-to-end response time of a task τ_i is given by $EER_i = R_i + S_i$, which composes its worst-case computation response time and its worst case communication latency (Figure 2). Its end-to-end schedulability can be tested by checking whether $EER_i \leq D_i$.

4.3 End-to-end schedulability of task chains

To test the schedulability of a task chain X, we need to consider the individual end-to-end response times of all tasks $\tau_i \in X$. Before we can do that, we must discriminate three modes of execution for task chains over multiple processing elements: *sequential*, *parallel* and *pipelined*.

In a sequential execution, a task chain will be executed completely, in one or more processors, before it can be executed again. In other words, only a single task $\tau_i \in X$ can be executing at a given point in time.

In a parallel execution over multiple processors, there are no constraints over the execution of task chains, and arbitrarily many jobs of a task chain can be executing at the same time.

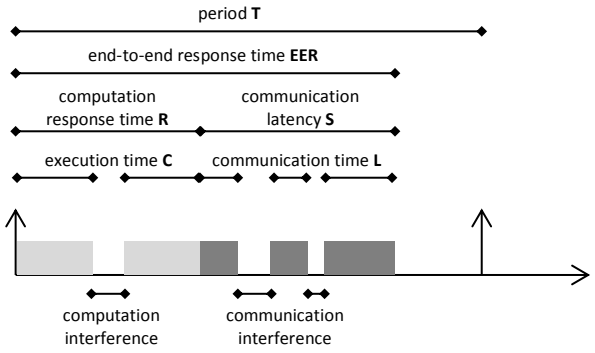

Figure 2. End-to-end response time of a communicating task

A pipelined execution is a special kind of parallel execution which allows multiple jobs of the same task chain to be executed simultaneously over different processors, but disallows the simultaneous execution of more than one job of the same task. A common pattern for pipelined execution is to have a number of jobs of a task chain X running concurrently, each of them released after T_x time units after the preceding one, in a phase-shifted way. We refer to this pattern as a synchronous pipeline. Figure 3 shows an example of a task chain executing as a synchronous pipeline. It includes three tasks τ_1, τ_2 and τ_3 running on separate cores (each of them represented on a separate timeline), their respective communications over NoC links (also shown over separate timelines), occasionally suffering interference from higher priority tasks and packets (not shown in the figure). Curved arrows show the functional dependencies between the computation and communication components of one chain, making it easier to see that those dependencies will always be satisfied as long as each task meets its end-to-end deadline constraint.

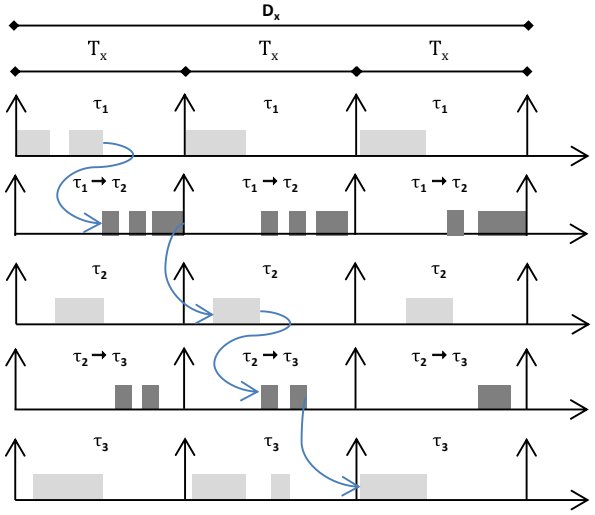

Figure 3. Example of a 3-task chain executed in a synchronous pipeline over 3 processors

In this paper, we concentrate on the synchronous pipeline case. We argue that it can be analysed by the end-to-end schedulability test described in subsection 4.2. Let us assume that the end-to-end deadline D_x of a synchronously pipelined task chain X, which is the maximum tolerated amount of time elapsed between the release of its first task and the delivery of the output of its last task, is equal to the number x of tasks in the chain multiplied by the chain period T_x. In a synchronous pipeline, we can partition

D_x equally among all tasks of the chain, so the end-to-end deadline D_i of each individual task is equal to T_x. This enables a task chain X with x tasks to produce its output x periods after its release, but once the pipeline is filled each of its jobs will produce an output at every period.

The schedulability test for this particular case is a simple check of whether $\forall \tau_i \in X, EER_i \leq T_x$. This assumes that there will be acceptable deadline misses for the first x jobs of the task chain X while the pipeline is being filled, and guarantees that the system will never miss a deadline after that. The intuition behind this approach is that each task of the chain will be triggered every T_x time units, and has to finish computing and communicating with the next task of the chain before the end of the period, so that the following task will have all the data it needs before it can run at the next periodic tick.

4.4 Link utilisation tests

Similarly to Eq. 1, a utilisation test can be applied to each of the NoC links, aiming to check whether the messages mapped to each of them will not exceed their bandwidth:

$$\sum_{\varphi_i \in map^{-1}(\lambda_{a1})} \frac{L_i}{T_i} \leq 1 \quad \textbf{(4)}$$

Again, this test is necessary but not sufficient because even though the link λ_{a1} may be capable to transmit all messages mapped to it without starvation, they might not meet their deadlines.

By considering the multi-hop nature of NoCs, we identify another utilisation test that addresses the direct interference set *interf(φ_i)* of a message φ_i:

$$\sum_{\varphi_j \in interf(\varphi_i)} \frac{L_j}{T_j} \leq 1 \quad \textbf{(5)}$$

The intuition behind this test is the following: if a message φ_j can interfere and hinder the progress of another message φ_i over the NoC, this happens regardless of the link where the contention happens. In other words, the complete route of a message can be seen as a single resource with exclusive access, and if a higher priority message needs to use any part of that route the whole transmission of φ_i will be halted. For example, if φ_i is routed over *n* different links and it suffers interference from φ_j which also uses one or more of those links and has a higher priority, the time φ_i waits for the shared link(s) will be the same if they share link λ_1, λ_2, …, λ_n, or any combination of them, as in every case φ_i will not be able to progress in a pipelined fashion towards its destination. The same intuition can be extended to other higher priority messages that share any possible combination of links with φ_i. Therefore, we conclude that the direct interference set *interf(φ_i)* determines all contenders for the route of a message φ_i, and the overall utilisation of that route has to be less than 1 due to the exclusive access.

The proof that this test is tighter than the test in Eq. 4 lies on the fact that the direct interference set *interf(φ_i)* is a superset of each of the sets including the messages that share any of the links φ_i is mapped to, and that can interfere with it: $\forall \lambda \in map(\varphi_i)$, $hp(map^{-1}(\lambda)) \in interf(\varphi_i)$, where $hp(\{\varphi_i \ldots \varphi_n\})$ denotes the subset of messages that have higher priority than φ_i. Actually, from the definition given in Section 4.2 it is easy to see that the direct interference set is actually the union of all those sets: $interf(\varphi i) = \cup_{\lambda \in map(\varphi i)} hp(map^{-1}(\lambda))$. Thus, the utilisation test given in Eq. 5 will cover, when applied to the lowest priority message of each link, the test given in Eq. 4.

In any case, both utilisation tests identified in this subsection are necessary, but not sufficient. While they are useful to discriminate unschedulable mappings, they cannot guarantee schedulability. They are nonetheless useful to prune large mapping spaces, as they are less computationally expensive than the tests described in subsections 4.2 and 4.3.

5. EXPERIMENTAL WORK

To evaluate the correctness and usefulness of the schedulability tests described in the previous section, we devised two types of experiment. In subsection 5.1, we will compare the figures for computation and communication response times found using the proposed schedulability tests with figures obtained through simulation of predefined configurations of a NoC-enabled embedded system. In subsection 5.2, we will then show that the proposed tests can be used as a fitness function within a search-based optimisation heuristic.

5.1 Joint end-to-end schedulability analysis and simulation

In this series of experiments, we analyse the schedulability of a benchmark application over a specific NoC-based embedded platform.

The platform follows the architecture described in Section 2, with homogeneous cores running priority-preemptive task schedulers, distributed memory, 2D-mesh NoC interconnect with XY dimension routing, credit-based flow control, 8 virtual channels with 3-flit input buffers per port and priority-preemptive link arbitration. It is worth noticing that the schedulability tests proposed in Section 4 would support alternatives on most of those architectural choices, but priority-preemptive arbitration at the cores and NoC links is a requirement.

The chosen benchmark application is based on the autonomous vehicle (AV) introduced in [12], including 39 communicating tasks performing functionality such as navigation control, vibration control and obstacle detection through stereo photogrammetry. Task periods vary between 0.04 to 1 second, and communication volumes vary between 1 and 76 kbytes.

To model the benchmark as task chains, a number of tasks of the original application had to be partitioned (i.e. to break tree-like structures when a task receives data from multiple sources). Furthermore, we had to introduce the notion of "sink tasks" to model DMA transfers to the local memory of the core where specific tasks are mapped to. In those cases, the destination task does not require any computation overhead (e.g. last task of a chain writes to a memory-mapped actuator, so it does not load the destination core, but used the bandwidth of the NoC links all the way to the destination interface).

To maintain the realism of the benchmark, we constrained all mappings used in the paper in such a way that all partitions of a task, as well as its respective sink, are mapped to the same core (so that only possible mappings of the application were considered). Table I shows the complete set of tasks, showing which chain they belong to (chains have lengths between 2 and 5 tasks), their names (first four letters indicate the original task name from [12], with an appendix if the task is a partition or a sink of one of the original tasks), destination task, computation

time (in milliseconds), period (in milliseconds), priority, and communication volume (in bytes).

Table I – Autonomous Vehicle benchmark

task	chain	name	dest task	comp (ms)	period (ms)	pri	comm (bytes)
1	A	POSI-A	2	5	500	31	2048
2	A	NAVC-A	3	10	500	32	4096
3	A	OBDB-A	42	150	500	33	32768
4	B	OBDB-B	33	150	1000	34	65536
5	C	NAVC-C	40	20	100	24	1024
6	C	SPES-C	5	5	100	25	1024
7	D	NAVC-D	40	10	100	26	2048
8	E	FBU3-E	47	10	40	1	76800
9	F	FBU8-F	48	10	40	2	76800
10	G	VOD1	42	20	500	3	1024
11	H	VOD2	42	20	500	4	1024
12	I	FBU1	20	10	40	5	76800
13	J	FBU2	21	10	40	6	76800
14	K	FBU3	22	10	40	7	76800
15	L	FBU4	23	10	40	8	76800
16	M	FBU5	24	10	40	9	76800
17	N	FBU6	25	10	40	10	76800
18	O	FBU7	26	10	40	11	76800
19	P	FBU8	27	10	40	12	76800
20	I	BFE1	28	20	40	13	4096
21	J	BFE2	43	20	40	14	4096
22	K	BFE3	43	20	40	15	4096
23	L	BFE4	43	20	40	16	4096
24	M	BFE5	29	20	40	17	4096
25	N	BFE6	44	20	40	18	4096
26	O	BFE7	44	20	40	19	4096
27	P	BFE8	44	20	40	20	4096
28	I	FDF1	30	10	40	21	16384
29	M	FDF2	51	10	40	22	16384
30	I	STPH	43	30	40	23	8192
31	Q	POSI-Q	43	5	500	35	2048
32	R	USOS	43	5	100	27	2048
33	B	OBMG-B	41	20	1000	37	8192
34	S	TPMS	36	5	500	36	4096
35	T	VIBS	38	5	100	28	1024
36	S	STAC-S	46	10	1000	38	4096
37	U	SPES-U	45	5	100	29	2048
38	T	STAC-T	44	10	100	30	2048
39	V	OBMG-V	41	0.5	1000	39	4096
40	sink	DIRC-X	-	-	-	-	-
41	sink	OBDB-X	-	-	-	-	-
42	sink	NAVC-X	-	-	-	-	-
43	sink	OBMG-X	-	-	-	-	-
44	sink	THRC-X	-	-	-	-	-
45	sink	STAC-X	-	-	-	-	-
46	sink	TPMS-X	-	-	-	-	-
47	sink	VOD1-X	-	-	-	-	-
48	sink	VOD2-X	-	-	-	-	-
49	sink	FDF1-X	-	-	-	-	-
50	sink	FDF2-X	-	-	-	-	-
51	sink	STPH-X	-	-	-	-	-

We selected one platform configuration (a 4x4 mesh) and three different task allocations, and applied equations 2 and 3 to find the worst-case end-to-end response time of each of the 39 communicating tasks under each mapping. Figures 5.a, 5.b and 5.c show the results for mappings M1, M2 and M3 respectively.

The worst-case end-to-end response time of each task is plotted with a brown cross, and their individual deadline is shown as a red horizontal line. Mappings M2 and M3 are fully schedulable, as all EER values are below the respective deadlines. M1, however, has a number of unschedulable tasks, denoted by the brown crosses plotted at the upper margin of Figure 5.a (the actual worst case response times in those cases were not found, as our implementation stops iterating towards a solution once a deadline is missed).

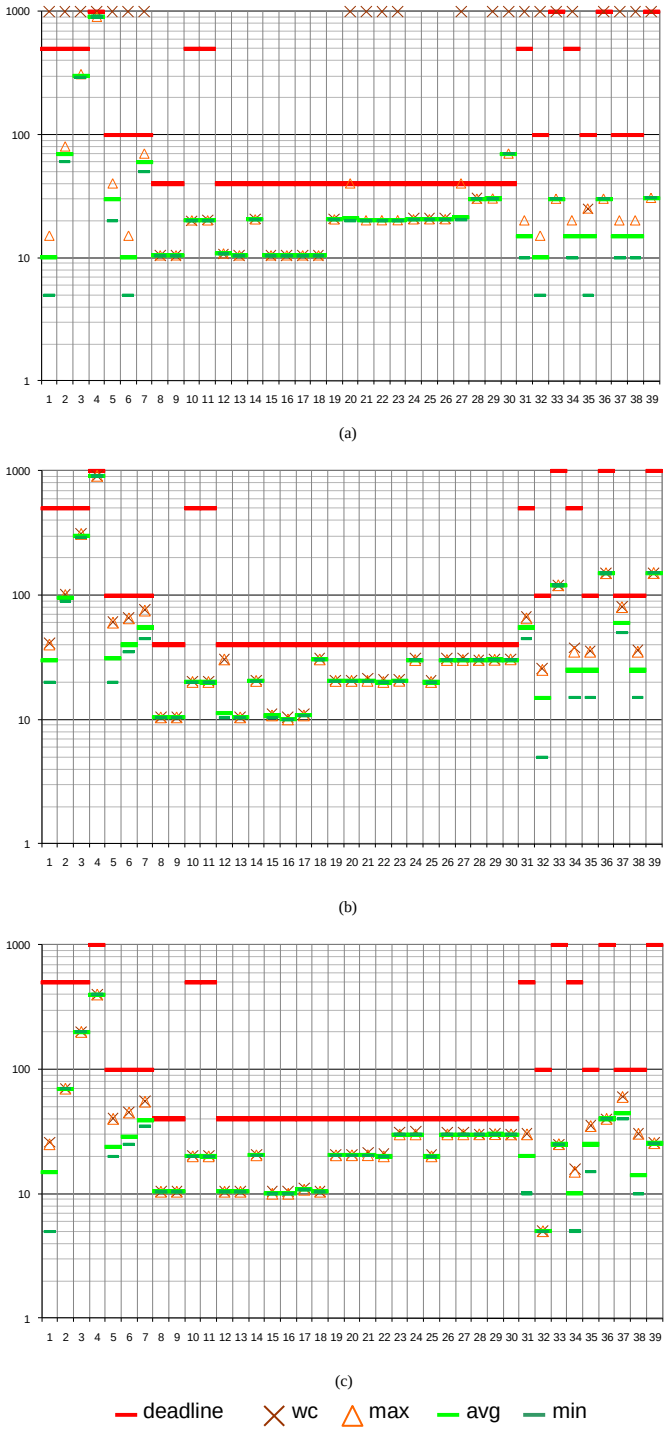

Figure 5. End-to-end response times (in ms) for all 39 communicating tasks under alternative mappings: (a) M1, (b) M2 and (c) M3.

We then used the tool flow presented in [13] and the simulation models presented in [14] to obtain latency figures for the execution of the benchmark application over the platform under all three mappings. We simulated each scenario for a target time

of 200 seconds, which allows for a good coverage of the application lifetime (the shortest period is of the video processing tasks, that must execute every 0.04 seconds to achieve 25 VGA frames per second, and the longest period is of 1 second for the tyre pressure control task). Figure 5 also shows, for each mapping, the best, worst and average end-to-end latency observed during simulation for each of the 39 communicating tasks. The plots show that the worst case response times found by our schedulability tests are effectively an upper bound to all results found with simulation. They also show that while M2 and M3 are fully schedulable mappings, M2 has higher worst case and maximum observed latencies (specially in communications 4, 7 and 37). Taking that into account, mapping M3 would be preferable as its results allow for larger safety margins.

5.2 Using end-to-end schedulability tests as fitness within search-based optimisation

As shown in Figure 5, the end-to-end timeliness of applications is affected by the way tasks and flows are mapped onto the NoC platform. Finding optimal mappings, or even acceptable ones, is a challenge in most NoC systems, and a significant amount of research was dedicated to this topic [15]. Heuristic search-based mapping is one of the mapping techniques reviewed by [15], and its distinctive feature is a fitness function to evaluate solutions over a given search space, aiming to converge towards solutions with increasing fitness. A common practice in such cases is to simulate the NoC platform with a given mapping for a specific amount of time, and use some aggregate of the latencies of all packets obtained through the simulation as the fitness of that mapping [16]. This process is then repeated for many different mappings across the search space until a mapping is found that fulfils the requirements (e.g. average latency of all packets below a given threshold).

In this subsection, we show that the schedulability tests described in Section 4 can be used to solve two problems found in search-based mappers with simulators as fitness function, specifically when it comes to optimise hard real-time systems. As discussed in Section 1, simulators cannot easily find worst-case packet latencies, and the time they take to run can be very high when evaluating complex NoCs. In search-based mappers, the second problem is particularly severe, because the search heuristic may have to simulate hundreds or thousands of different mapping before an acceptable solution can be found.

To solve those problems, we have implemented a search-based algorithm that uses the approach described in subsection 4.3 to find whether, given a particular mapping, how many of an application's tasks are end-to-end schedulable. Like in [16], our search-based heuristic follows an evolutionary approach, modelling a particular mapping as a chromosome representing on each gene the processing core where each task should be mapped (Figure 6.a). The evolution is performed across generations of a population of 100 individuals, each represented by one of such chromosomes. The initial population can be randomly generated, but subsequent generations are produced by applying crossover and mutation operations over the fittest chromosomes of the preceding one (Figure 6.b). In our implementation, crossovers were implemented by creating a new chromosome from the first and second halves of two existing chromosomes. Similarly, mutations created new chromosomes by swapping the contents of any two genes of an existing chromosome. The chosen chromosomes for crossover and mutation were those that, when evaluated using the technique described in 4.3, would have the lowest number of unschedulable tasks.

Ideally, after a number of generations the population will contain at least one individual with a chromosome representing a mapping that meets our constraints, i.e. has zero tasks that are end-to-end unschedulable. A detailed study on how different chromosome formats, population sizes, mutation and crossover styles and rates affect the convergence of the genetic algorithm towards a full schedulable solution can be found in [17].

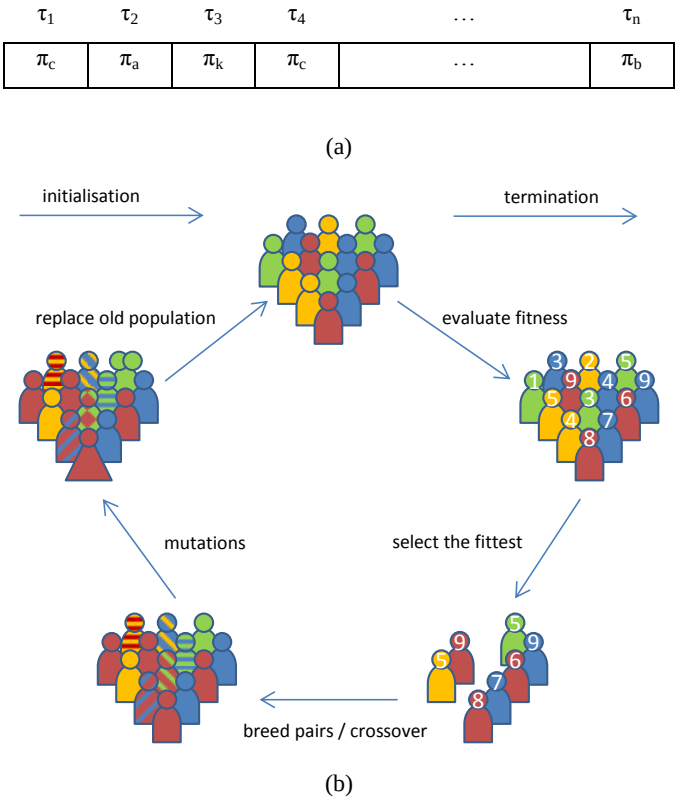

Figure 6. Evolutionary mapping: (a) chromosome format and (b) evolutionary search process.

We have used this evolutionary mapping algorithm to search for schedulable mappings of the AV application over 3x3, 4x4 and 5x5 mesh NoC platforms. Figure 7 shows the number of end-to-end unschedulable tasks of the best mapping of each generation. It can be seen that mappings for the 5x5 platform can be found very easily (as there are more resources and therefore less interference), reaching a fully schedulable mapping in 8 generations. For the 4x4 platform, the situation is slightly more difficult, but the evolutionary mapper is capable to find a fully schedulable mapping in 11 generations. Finally, for a 3x3 platform, the evolutionary mapping cannot find a fully schedulable mapping after 50 generations (because the utilization of the application exceeds the available capacity of the 3x3 platform), but it can clearly show improvements over generations, reaching a minimum of 12 end-to-end unschedulable tasks.

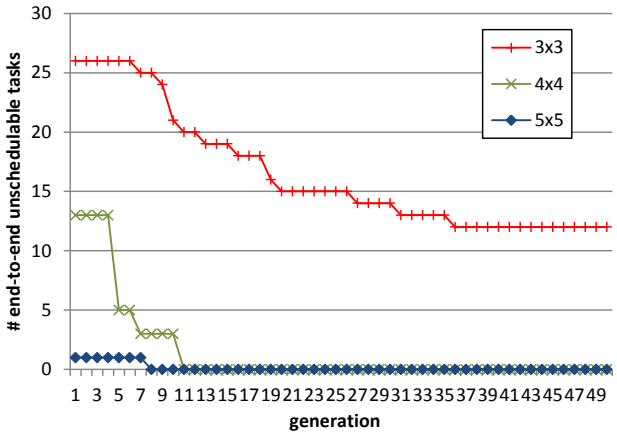

Figure 7. Number of end-to-end unschedulable tasks at each generation of the evolutionary mapping, for three different NoC platforms: 3x3, 4x4 and 5x5.

5.2.1 Performance comparison

The performance of the proposed schedulability test, when used as a fitness function of a search-based heuristic, shows a significant improvement over simulation-based fitness functions such as those used by [16]. A simple Java-based implementation of the proposed test takes 0.13 seconds to evaluate the schedulability of a single mapping of the AV application over a 4x4 NoC. This is at least one order of magnitude faster than simulation, as reported in [14], and consistently reinforced by our experiments reported in subsection 5.1. The time it takes to simulate 2 seconds of a single mapping of the AV application is 7.89 seconds, for a fast simulator operating at TLM (Transaction Level Modelling) level. For a cycle-accurate simulation of the same scenario, the time elapsed is 2895.69 seconds.

Such numbers show that even if it would be feasible to identify the worst-case release scenario for all tasks and packets, a state-of-the-art NoC simulator would take 60 times longer (or up to 20000 times longer, if full accuracy must be achieved) to evaluate the fitness of one specific mapping. Recalling that in a typical search-based mapping heuristic one must check the fitness of thousands of mappings, we can clearly see the advantage of the proposed approach (e.g. in the experiments described above we needed 1100 application of the fitness function to find a fully schedulable mapping for a 4x4 platform, i.e. 11 generations of a population of 100 individuals).

6. RELATED WORK

Besides RTA and its derivatives, other analytical models have also been used to evaluate schedulability in NoCs.

Beekooij et al. [18] have proposed an extension to dataflow analysis (originally proposed by Lee and Messerschmitt [19]) that can model the behaviour of a homogeneous synchronous dataflow (HSDF) application performing computation and communication over a specific type of NoC (i.e. statically scheduled time-division multiplexing of links). They assume that the worst-case computation time of each application task is known (just like in this paper, as referred as C_i in Section 3). However, due to the nature of their underlying NoC architecture, they can assume that there is no contention over NoC links, and thus the delay introduced by the NoC to each data transfer can be established independently for each task chain. Therefore, the worst case end-to-end latency of a task-chain can be found by dataflow analysis, which can calculate the latest arrival time of the data token at the output of the last task of the chain.

Qian et al. [20] proposed the use of network calculus [21] to calculate worst-case packet latency bounds in wormhole NoCs, as long as all traffic can be modelled as an arrival curve and all NoC routers can be modelled by a service curve. Such curves abstract the actual behaviour of the application and the NoC by the bandwidth required or provided, respectively, at each point in time. The calculation of latency bounds is done through algebraic operations over all arrival curves at a given router, as well as the router's service curve. The main challenge of this approach is to represent the behaviour of a sequence of specific routers (with their particular buffering and arbitration schemes) as a service curve. The modelling of the application traffic as arrival curves is also challenging, specially if the variations on the source task's execution are taken into account (e.g. execution time variability or interference from tasks running on the same core), and this is currently an open problem preventing the use of network calculus on the evaluation of NoC end-to-end schedulability.

Other approaches to evaluate NoC schedulability are surveyed by in [22], all of them based on dataflow analysis, network calculus or RTA. The survey also states the difficulty to compare different analytical methods based on distinct formalisms, as they have fundamentally different assumptions. Still, they provide a summary of strengths and weaknesses of each type of analysis. Their assessment of dataflow and network calculus models has similar views as the ones we provided above, emphasizing the restrictions that must be imposed on the application behaviour and the NoC resource sharing disciplines. Their assessment of RTA and its derivatives, however, states that the main weakness is the inability to represent dependencies between flows, which is an issue that we have directly addressed in this paper and solved for the restricted case of synchronous pipelines.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated ways to determine whether application tasks executing and communicating over a specific NoC-based multicore can meet all application-specific timing constraints. We have identified a number of schedulability tests, and have shown their utility within distinct steps of an embedded system design flow. By combining them with simulation, designers can obtain a more detailed understanding of the overheads that are needed to guarantee performance in the worst case, as opposed to the average case. And by using them as fitness in search-based optimisation, we enabled a faster coverage of the typically large design spaces given by the multiple design alternatives in this kind of system.

For the sake of simplicity, we assumed an application model where tasks require all data to be available locally before they execute, and can send a single message only after they finish their computation. While restrictive, this model supports the widely used Actor model (i.e. read-execute-write) and can represent applications based on task chains. A more general formulation that allows tasks to send an arbitrary number of messages can be easily derived, but was left to future work, and would enable the representation of and tree-like structures. Even in the case of simple task chains, we have only addressed the restricted case of

synchronous pipelines. Extensions to the presented tests to address general pipeline and sequential execution are currently under investigation, and so is the use of deadline decomposition approaches (such as in [23] and [24]) and schedulability tests supporting release offsets (such as [25]).

Additional future work can take advantage of the utilisation tests presented in subsections 4.1 and 4.4 to accelerate the design space exploration by quickly pruning away mappings with over-utilised cores or links. Such approach could improve even further the performance reported in subsection 5.2.1, where the substantially heavier schedulability test presented in subsection 4.3 was used throughout the whole optimisation.

Finally, the proposed platform model assumes homogeneous cores, switches and links. Interesting avenues of research can also be opened by lifting such restrictions.

8. ACKNOWLEDGEMENTS

The author would like to thank Zheng Shi, Alan Burns, Osmar Marchi dos Santos and Borislav Nikolic for the discussions on the tests presented in Section 4; and Paris Mesidis, Adrian Racu and Norazizi Sayuti for the discussions and help with the experimental work supporting subsection 5.2.

9. REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[2] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," in *ACM/IEEE Int Symposium on Networks-on-Chip (NOCS)*, 2008, pp. 161–170.

[3] Z. Shi and A. Burns, "Schedulability analysis and task mapping for real-time on-chip communication," *Real-Time Syst*, vol. 46, no. 3, pp. 360–385, Sep. 2010.

[4] N. Genko, D. Atienza, G. De Micheli, L. Benini, J. M. Mendias, R. Hermida, and F. Catthoor, "A novel approach for network on chip emulation," in *IEEE Int Symposium on Circuits and Systems (ISCAS)*, 2005, pp. 2365–2368 vol. 3.

[5] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, Oct. 2004.

[6] A. Agarwal, "The Tile Processor: A 64-Core Multicore for Embedded Processing," in *11th Annual Workshop on High Performance Embedded Computing (HPEC)*, Lexington, Massachusetts, USA, 2007.

[7] 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.

[8] D. Wiklund and Dake Liu, "SoCBUS: switched network on chip for hard real time embedded systems," in *Int Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[9] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual channels in networks on chip: implementation and evaluation on hermes NoC," in *18th Annual Symposium on Integrated Circuits and Systems Design (SBCCI)*, Florianopolis, Brazil, 2005, pp. 178–183.

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

[11] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011.

[12] Z. Shi, A. Burns, and L. S. Indrusiak, "Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching," *IJERTCS*, vol. 1, no. 2, pp. 1 – 22, Jun. 2010.

[13] L. S. Indrusiak, I. Quadri, I. Gray, N. Audsley, and A. Sadovykh, "A MARTE subset to enable application-platform co-simulation and schedulability analysis of NoC-based embedded systems," in *2012 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2012, pp. 1–7.

[14] 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.

[15] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for Network-on-Chip design," *Journal of Systems Architecture*, vol. 59, no. 1, pp. 60–76, Jan. 2013.

[16] G. Ascia, V. Catania, and M. Palesi, "A Multi-objective Genetic Approach to Mapping Problem on Network-on-Chip," *Journal of Universal Computer Science*, vol. 12, no. 4, pp. 370–394, 2006.

[17] A. Racu and L. S. Indrusiak, "Using genetic algorithms to map hard real-time on NoC-based systems," in *2012 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2012, pp. 1–8.

[18] M. Bekooij, R. Hoes, O. Moreira, P. Poplavko, M. Pastrnak, B. Mesman, J. Mol, S. Stuijk, V. Gheorghita, and J. Meerbergen, "Dataflow Analysis for Real-Time Embedded Multiprocessor System Design," in *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, 2005, pp. 81–108.

[19] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[20] Y. Qian, Z. Lu, and W. Dou, "Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip," in *3rd ACM/IEEE International Symposium on Networks-on-Chip, 2009. NoCS 2009*, 2009, pp. 44–53.

[21] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Heidelberg: Springer-Verlag, 2001.

[22] A. E. Kiasari, A. Jantsch, and Z. Lu, "Mathematical Formalisms for Performance Evaluation of Networks-on-chip," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 38:1–38:41, Jul. 2013.

[23] T. F. Abdelzaher and K. G. Shin, "Combined task and message scheduling in distributed real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 11, pp. 1179–1191, 1999.

[24] M. Saksena and S. Hong, "An engineering approach to decomposing end-to-end delays on a distributed real-time system," in *4th Int Workshop on Parallel and Distributed Real-Time Systems*, 1996, pp. 244–251.

[25] I. Bate and A. Burns, "Schedulability analysis of fixed priority real-time systems with offsets," in *9th Euromicro Workshop on Real-Time Systems*, 1997, pp. 153–160.