DTFM: a Flexible Model for Schedulability Analysis of Real-Time Applications on NoC-based Architectures

Mourad Dridi*, Stéphane Rubini*, Frank Singhoff*, Jean-Philippe Diguet†,
*Lab-STICC, CNRS, UMR6285, Univ. Bretagne Occidentale, 29100 Brest, France
†Lab-STICC, CNRS, UMR6285, Univ. Bretagne Sud, 56100 Lorient, France
{mourad.dridi, stephane.rubini, frank.singhoff}@univ-brest.fr, jean-philippe.diguet@univ-ubs.fr

***Abstract*—Many-core processors are expected to be hardware targets to support the execution of real-time applications. In a many-core processor, cores communicate through a Network-On-Chip (NoC), which offers high bandwidth and scalability, but also introduces contentions leading to additional variability to task execution times. Such contentions also strongly increase the pessimistic trend of worst case execution time estimation. Consequently, modeling and analysis of network contentions interferences on many-core processors is a challenge to support real-time applications. In this article, we formalize a dual task and flow model called DTFM. From the specification of a real-time application composed of a set of tasks and their communication dependencies, DTFM allows us to compute flow requirements and to assess predictability of the tasks. DTFM is extensible enough to be adapted to various NoCs and task models, allowing designers to compare candidate software and NoC architectures. Furthermore, we introduce an original validation approach based on the cross-use of a task level real-time scheduling analysis tool and a cycle-accurate SystemC NoC simulator.**

I. Introduction

NoCs are widely used in many-core systems since they provide scalability, modularity, and communication parallelism. Many-core allows multiple applications to run at the same time in the same processor. These applications exchange messages through the communication infrastructure. Real-time applications have very stringent communication service requirements. The correctness of real-time communications depends not only on the communication result but also on the completion time bound [1].

In this article, we address temporal failures which occur when a real-time application is unable to meet its deadlines. To predict whether a given application is able to meet its deadline, usual schedulability analysis methods require to know the Worst Case Execution Time (WCET) of each task [2]. Then, schedulability analysis methods allow designers to investigate how the tasks will be scheduled on the hardware platform and how task executions will be delayed due to hardware contentions.

A. Problem Statement

Many task models were proposed to deal with real-time applications [2]. All these models are not immediately suitable with NoC architectures. They do not take into account the communications through the network, the task mapping and the latencies introduced by this new type of shared resources. Many traffic-flow models are also proposed to analyze the real-time communications for NoC architectures. All these models focus on real-time traffic-flows and they do not take into account the schedulability of tasks. A schedulable traffic-flow does not allow us to conclude that the whole system is schedulable. Moreover, they do not offer the flexibility, which is necessary to consider different communication protocols. Considering the incomplete state of the art, we investigate how to formalize and implement task and flow models that will provide a common platform for the evaluation of different solutions. Such a platform is required to run simulations and perform schedulability analysis with real-life cases in order to compare different solutions at hardware, software, scheduling or protocol levels.

B. Contributions

We first propose DTFM, a Dual Task and Flow Model, that allows designers to perform a complete schedulability analysis in order to assess the predictability of real-time tasks running over NoC-based architectures.

The second contribution is a unique validation environment, which combines a tick accurate real-time simulator and analysis tool that implements the DTFM model and produces a task scheduling, which is then used to feed a SystemC cycle accurate NoC simulator. Such a validation environment could be applied to investigate the temporal behaviour of real-time applications running on various other hardware components such as cache hierarchies. The analysis tool, that implements DTFM, allows us to evaluate and compare different NoC architectures, which can be standard or designed for mixed criticality systems. It also allows us to consider different types of task communication protocols.

The remainder of the paper is organized as follows. The next section presents related works. Section III introduces background elements about the NoC we consider. Then, section IV proposes our dual flow and task model. Implementation and evaluation of DTFM are explained in section V and section VI concludes the article.

II. Related Works

Several schedulability analysis approaches for real-time on-chip communication have been explored.

In [3], the authors propose an analysis approach for real-time on chip communication with wormhole switching and fixed priority scheduling. [1] focuses on real-time communication service with a priority share policy.

In all these works, the scheduling analysis of the real-time applications is computed for a given flow model and for a given task model and they do not take into account the schedulability of tasks. While in our work, we expect to compute the flow model from a given task model so we examine simultaneously task and flow schedulability. Moreover we provide a flexible model in order to consider different communication protocols and different NoC architectures.

There are also several existing tools and frameworks which support mapping of applications onto NoC-based architectures. [4] proposes a simulator developed in SystemC with a C++ plugin,

in order to model concurrent hardware modules at the cycle-level accuracy.

There are also other simulators such as [5] which allows users to setup and simulate a broad range of network configurations defined by network size, buffer size and routing algorithm.

In order to evaluate the usage of Many-Core architecture for critical applications, authors in [6] propose a simple one task to one core execution model and develops test drive programs. The developed programs evaluate the NoC latency performance.

All of these simulators provide Transaction-Level Model of NoC architectures, without any consideration for Task schedulability. On the other hand, DTFM, with the tools that support it, allows us to analyze the schedulability of real-time systems along with a cycle-accurate implementation of communications so that a real-time communication analysis can also be performed.

III. Background

In this section, we introduce the NoC we assume in this work. Then, we identify the timing issues a real-time application faces when running over NoC architectures.

A. NoC

A NoC is a network of nodes which can be a processor, a memory a peripheral or a cluster. A node can access the network through a network interface (NI) that usually transmit and receives data in the form of packets.

Wormhole is largely adopted in NoCs because it does not require large capacity buffers. In a wormhole network, the packet is divided into a number of fixed size flits [7]. The packet is split into a header flit, one or several body flits and a tail flit. The header flit stores the routing information and builds the route. As the header flit moves ahead along the selected path, the remaining flits follow in a pipeline way and possibly span a number of routers. If the chosen link is busy, the header flit is blocked until the link becomes available. In this case, all the packets will be blocked in routers along the selected path. Then the network will introduce a new latency for the task that we cannot ignore during schedulability analysis.

We assume a standard NoC based on packet-based wormhole switching, with a 2D-mesh topology and XY deterministic routing. In this study we do not consider NoC with flit level preemption or with any specific protocol for real-time traffic.

B. Types of delays in a NoC

For each NoC configuration, we have different types of latency. In this section we focus on latency introduced by the type of NoC assumed in this paper. Let's see the different types of delays that we must understood to assess schedulability of real-time tasks deployed on such NoC [3], [1].

1) Path delay: The first type of delay is straightforward and can be called the Path Delay:

Definition 1: The Path Delay DP is the time required to send the packet from the transmitter to the receiver when no traffic flow contention exists.

We assume here that packets of a flow do not share any physical link with other flows along the path. This delay is variable and depends on several parameters such as the flow rate and the size of the message [3]. Note that with the path delay, we consider the worst case latency introduced by the network without any contention. If the real-time flow meets its deadline with the worst case then it will meet the deadline for any other scenario.

2) Delay due to Direct Interference: A second kind of delay occurs due to direct interference:

Definition 2: There is a direct interference when two flows share the same physical link at the same time. We called the delay caused by a direct interference between flows a Direct Delay DD.

In all situations, the direct interference can be presented as a non-deterministic source of delay. In our analysis, we consider the worst case latency introduced by the direct interference. We assume the packet from the observed traffic-flow is arrived just after other packets for each shared link. We assume also a maximum size of the packets [1].

3) Delay due to Indirect Interference: The last delay we address in this article is called Indirect Interference:

Definition 3: There is an indirect interference when two flows do not share the same physical link but they interfere directly with the same flows. We call the delay caused by an indirect interference, an Indirect Delay, noted DI.

We can have many cases where a flow is blocked by another flow which is also blocked by others and so on. In this case, the first flow must wait for a non-deterministic delay. Like the direct delay, an indirect delay is not deterministic because it depends on the history of network, on the flows rate, the messages size and the links states [1].

In this paper, we compute path delay and direct delays according to equations given in [1] and we consider that indirect interferences are unpredictable delays, we then do not handle indirect delays.

IV. The Dual Task and Flow Model

In this section, we introduce our task and flow model. We also show how to calculate the flow model from the task model. A simple example is presented in order to illustrate those models. Finally, we explain how to compute flow parameters for a given communication protocol.

A. Presentation of the approach

The figure 1 shows the overall approach we propose in this article. From the task model, the mapping of the tasks on the processing units and the NoC model, DTFM allows us to generate the associated flow model. The flow model specifies the messages that have to be transmitted in the NoC to enforce the communications between the tasks. From a flow model and after identification of the different delays elapsed in the NoC, DTFM checks the predictability of the flows, i.e. detects when unpredictable delays may occur. If there is no unpredictable flows, the analysis tool implementing DTFM can compute the deadline and the communication delays of each message, and then, decides whether the system is schedulable. In the sequel, we introduce the task and flow notations used in this article.

B. Task and flow Models

The task model Γ comprises n periodic tasks: $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$

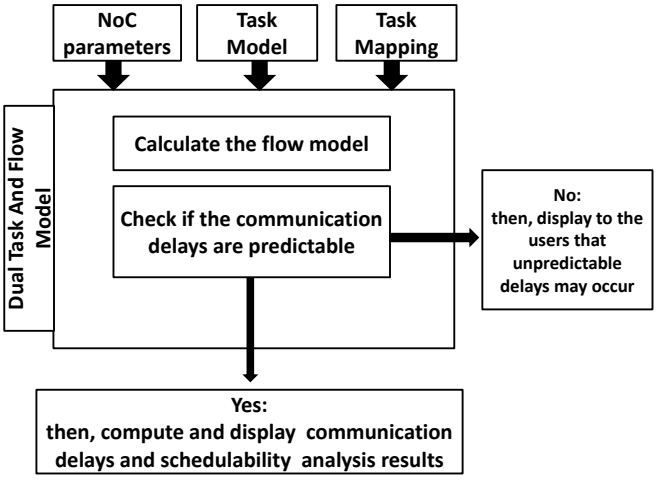

Fig. 1: The Dual Task and Flow Model

Each task leads to a sequence of jobs. Each task τ_i has a set of properties and timing requirements as follows:

$\tau_i = \{ O_i, T_i, C_i, D_i, \Pi_i, Node_i, E_i \}$

where:

- $O_i \in \mathbb{N}^+$ is the first release time of the task τ_i, i.e. the release time of the first job of τ_i.
- $T_i \in \mathbb{N}^+$ is the period of the task, i.e. the fixed delay between two successive jobs of the task.
- $C_i \in \mathbb{N}^+$ specifies the worst case execution time of a job of the task.
- $D_i \in \mathbb{N}^+$ is the deadline. We note that each task deadline must be less than or equal to its period.
- $\Pi_i \in \mathbb{N}^+$ is the fixed priority of the task. The value 1 denotes the highest priority level while a larger value is a lower priority.
- $Node_i \in \mathbb{N}^+$ identifies the node running the task. This parameter allows us to introduce the mapping configuration, i.e. on which processing unit each task is assigned to.
- $E_i : \Gamma \longrightarrow \Gamma^p$
  $\tau_i \longrightarrow E(\tau_i) = \{\tau_j, \ldots, \tau_k \}$
  The function E allows us to introduce the precedence constraints in the task model. For a given task τ_i, the function E determines all the tasks τ_j that receive messages from the task τ_i. Furthermore, we also assume that tasks use a given communication protocol to exchange their messages.

The flow model comprises m periodic traffic flows: $\psi = \{\rho_1, \rho_2, \ldots, \rho_m\}$

Each flow ρ_i raises a sequence of messages in a similar way that a task raises a sequence of jobs. The flow is defined as follows:

$\rho_i = \{ O_i, T_i, D_i, \Pi_i, NodeS_i, NodeD_i, F_i \}$

where:

- $O_i \in \mathbb{N}^+$ is the release time of the messages, i.e. the first time when a message of the flow becomes ready to be transmitted.
- $T_i \in \mathbb{N}^+$ is the period of the message. It is the fixed delay between releases of successive messages.
- $D_i \in \mathbb{N}^+$ is the deadline of the message, i.e. the time instant which allows both receiver and transmitter tasks to complete their execution prior their deadlines. Our model has the same restriction as [1] and [3]: each flow deadline must be less than or equal to its period.
- $\Pi_i \in \mathbb{N}^+$ is the priority of the messages. The value 1 denotes the highest priority level while a larger value indicates a lower priority.
- $NodeS_i \in \mathbb{N}^+$ is the node running the transmitter task.
- $NodeD_i \in \mathbb{N}^+$ is the node running the receiver task.
- $F_i : \psi \longrightarrow \Omega^p$
  $\rho_i \longrightarrow F(\rho_i) = \{e_{p,k}, \ldots, e_{i,j}\}$
  where Ω is the set of physical links in the NoC. If a and b are neighbor routers, $e_{a,b}$ (resp. $e_{b,a}$) is the physical link from the router a (resp. b) to the router b (resp. a).
  F_i is a function that computes the links used by a flow. Said differently, F_i identifies the physical links that will be used by a given message.
  The function F_i helps us to understand the relationships between the various flows that transit through the network and to determine the interferences between messages sent by the tasks.

From the task model, we can compute the corresponding flow model. For each task given by the E function, we have a flow ρ where the transmitter is the task antecedent and the receiver is the task image.

We define the function G with two variables to calculate the flow model from the task model:

$\mathrm{G} : \Gamma^2 \longrightarrow \psi$

$(\tau_i, \tau_j) \longrightarrow \mathrm{G}((\tau_i, \tau_j)) = \rho$

i $\in$ [1 .. n]; j $\in$ [1 .. n]

If E $(\tau_i) = \tau_j$ then $\mathrm{G}((\tau_i, \tau_j)) = \rho = \rho_{i,j}$

with:

$\Pi(\rho_{i,j}) = \Pi(\tau_i)$

$\mathrm{NodeS}(\rho_{i,j}) = \mathrm{Node}(\tau_i) = Node_i$

$\mathrm{NodeD}(\rho_{i,j}) = \mathrm{Node}(\tau_j) = Node_j$

Function G is surjective. Every element of the codomain is mapped to at least one element of the domain.

$\forall\ \rho \in \psi; \exists\ (\tau_i, \tau_j)$ such that $\rho = \mathrm{G}((\tau_i, \tau_j))$

You should notice that G depend on the communication protocol used by tasks to exchange their messages.

Task	O(s)	T(s)	C(μs)	D(s)	Π	Node	E
τ_1	1	6	100	2	1	3	τ_2 τ_3
τ_2	3	2	100	2	2	5	τ_5
τ_3	3	2	300	2	1	10	τ_4 τ_5
τ_4	5	2	500	2	2	8	τ_5
τ_5	7	2	100	2	1	16	τ_1

TABLE I: Task Model

Flow	Π	NodeS	NodeD	F
$\rho 1$ = $\rho_{1,2}$	1	3	5	e(3,2) e(2,1) e(1,5)
$\rho 2$ = $\rho_{1,3}$	1	3	10	e(3,2) e(2,6) e(6,10)
$\rho 3$ = $\rho_{2,5}$	2	5	16	e(5,6) (6,7) e(7,8) e(8,12) e(12,16)
$\rho 4$ = $\rho_{3,4}$	1	10	8	e(10,11) e(11,12) e(12,8)
$\rho 5$ = $\rho_{3,5}$	1	10	16	e(10,11) e(11,12) e(12,16)
$\rho 6$ = $\rho_{4,5}$	2	8	16	e(8,12) e(12,16)
$\rho 7$ = $\rho_{5,1}$	1	16	3	e(16,15) e(15,11) e(11,7) e(7,3)

TABLE II: Flow Model computed from the task model

C. Example of a real-time system over NoCs

The following example illustrates our task model and shows how we can generate the corresponding flow model. For this example, we use a mesh 4 $\times$ 4 NoC and the routing algorithm is a XY routing. We consider a real-time application composed of 5 tasks as shown in Figure 2.

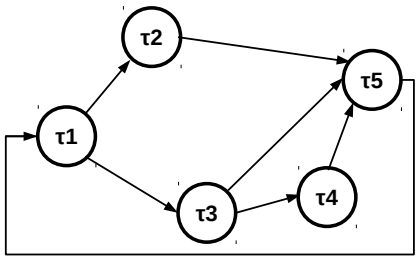

Fig. 2: Task Graph Model

Table I presents the task model. As shown in Table I, the task τ_3 runs on node 10. Its parameters are priority $\Pi = 1$, period $T = 2$ and deadline $D = 2$. E(τ_3) indicates that τ_3 sends messages to τ_4 and τ_5.

From this task model and with the G function, we can compute the flow model. As shown in Table II, the flow ρ_4 is sent from τ_3 to τ_4, then it is sent from the node 10 to the node 8 because NodeS(ρ_4) = Node(τ_3) = 10 and NodeD(ρ_4) = Node(τ_4) = 8. Then it will use the links e(10,11), e(11,12) and e(12,8).

D. Example of flow parameters for a given task communication protocol

You should notice that O_i, T_i and D_i parameters of flow ρ_i depend on the communication protocol used by tasks to exchange their messages. Then, to produce those parameters from the task model, the behavior of the task communication protocol has to be considered.

In this paper, we assume immediate data connection protocol which is defined in Architecture Analysis and Design Language (AADL) [8]. This protocol works as follow: tasks send their packets at the end of their execution and the packets are read before the execution of the receiving tasks. We assume the period of the flow is equal to the period of the transmitter task.

Let consider an example of the two tasks τ_1 and τ_2. τ_1 sends a flow ρ_1 to τ_2.

τ_1 = { O_1, T_1, C_1, D_1, Π_1, $Node_1$, E_1 }

τ_2 = { O_2, T_2, C_2, D_2, Π_2, $Node_2$, E_2 }

ρ_1 = { O_1, T_1, D_1, Π_1, $NodeS_1$, $NodeD_1$, F_1 }

For such flows, the parameters O_i, D_i and T_i can be computed as follows:

$O_{\rho_1} = O_{\tau_1} + C_{\tau 1} = O_1 + C_1$

$T_{\rho_1} = T_{\tau_1}$

$D_{\rho_1} = D_{\tau_2}$ - ($C_{\tau_1} + C_{\tau_2} + O_{\tau_1}$ - O_{τ_2})

V. Implementation and Evaluation of DTFM

In this section, we first explain the implementation of DTFM. Second, we present its evaluation.

A. Implementation into Cheddar

We implemented DTFM into Cheddar [9][1]. Cheddar is a GPL framework that provides a scheduling simulator, schedulability tests and various features related to the design and the scheduling analysis of real-time systems. To perform analysis, Cheddar handles an architecture design language, called *CheddarADL* [10]. *CheddarADL* allows the users to describe both the software and the hardware parts of the system they expect to analyze. To implement DTFM into Cheddar, we extend *CheddarADL* to model the NoC and its size, the processor locations in the mesh, usual scheduling parameters, the tasks parameters, the tasks mapping and the dependencies between tasks. From the task set and the NoC model, DTFM produces the associated flow model, checks if deterministic communication delays can be produced by the NoC, and finally, computes those communication delays.

B. Evaluation with a multiscale toolset

We have produced several experiments in order to evaluate the correctness and the scalability of DTFM.

To perform such evaluations, we have implemented and used a multiscale toolset composed of the tick accurate real-time scheduling simulator of Cheddar that simulates the behavior of the task set, and of SHOC [11], a NoC cycle accurate SystemC-TLM simulator that allows us to observe the traffic in the NoC. SHOC supports different types of traffic generators and consumers; In our study we consider ad hoc producers and consumers. SHOC provides all NoC components, and IPs required for MPSoC simulations. All components can be parameterized. In this experiment we have designed and simulated

[1]DTFM is available at http://beru.univ-brest.fr/svn/CHEDDAR/.

	U = 0.3 DP(ns) (SHOC)	U = 0.3 DP(ns) (DTFM)	U = 0.5 DP(ns) (SHOC)	U = 0.5 DP(ns) (DTFM)	U = 0.7 DP(ns) (SHOC)	U = 0.7 DP(ns) (DTFM)
ρ_1	300	303	300	303	300	303
ρ_2	297	298	297	298	297	298
ρ_3	290	293	290	293	290	293
ρ_4	295	298	295	298	295	298
ρ_5	292	293	292	293	292	293

TABLE III: Evaluation of Path delay

the temporal behavior of a 16-cores MPSoC that is interconnected through a 4x4 mesh-based NoC with XY-routing.

In order to perform the NoC simulations, SHOC requires the release times of each message. So for each experiment, Cheddar produces a task scheduling which is then used to feed the SystemC NoC simulator. Cheddar computes each task completion time and we do the assumption that any message is sent on completion time of the sender task, according to the protocol *immediate data connection* as defined in the AADL standard.

1) Correctness of the path and direct delays computed by DTFM: For this first experiment, we handle task sets with 64 periodic and dependent tasks. On each processor we run 4 tasks. Tasks communicate between each other with messages. Each message contains one packet and the packet size is about 5 flits.

In this section, we focus on different types of delay in the NoC. For each experiment, we compare the results computed by DTFM with the result of SHOC using the same settings.

We first evaluate the path delays computed by DTFM. In this experiment, we use a manual task mapping to enforce that flows generated by DTFM do not share any physical link. The utilization of each processor is set from 0.1 to 0.9 with a step of 0.2. For each processor utilization value, we generate 10 task mappings, and for every configuration we vary the period of tasks from 1000 ns to 1500 ns with a step of 20 ns. To generate each task set, we apply UUniFast [12].

Table III compares the delays produced by SHOC and DTFM for some representative analyzed task sets. We can see that DTFM is able to predict an upper bound of the path delays given by SHOC. Such results are consistent as the variation of processor utilization has no effect in the path delays, which is an expected behavior.

We now evaluate direct delays computed by DTFM. In this new experiment, we use a task mapping with more flows than the previous experiment. Then, some flows may share physical links. In order to study the delays due to direct interferences, a physical link is used at most by two flows. We generate 10 different task mappings with direct interferences. For every configuration, we vary the period of tasks from 1000 ns to 1500 ns with the step of 20 ns. Again, we apply UUniFast.

In this experiment, we change the release time of tasks in order to study different situations of contention between flows. We present in Figure 3 the difference between SHOC simulations and DTFM computations for some of the analyzed task sets. Each curve represents the difference for the same flow with different release times. We can see that DTFM is able to predict a upper bound of the direct delays given by SHOC.

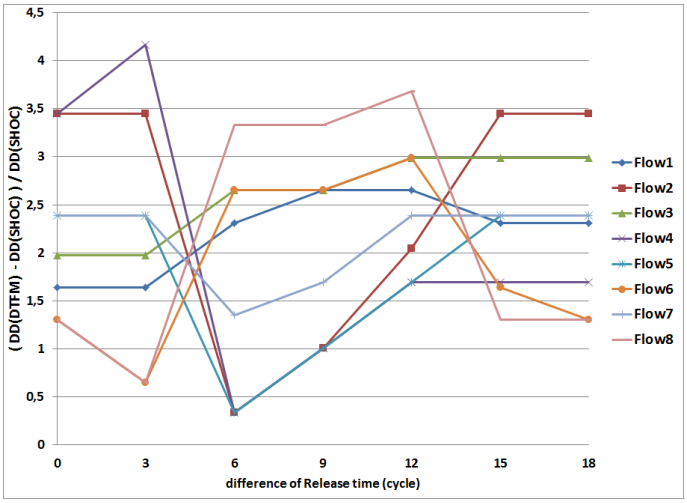

Fig. 3: Direct interference evaluation

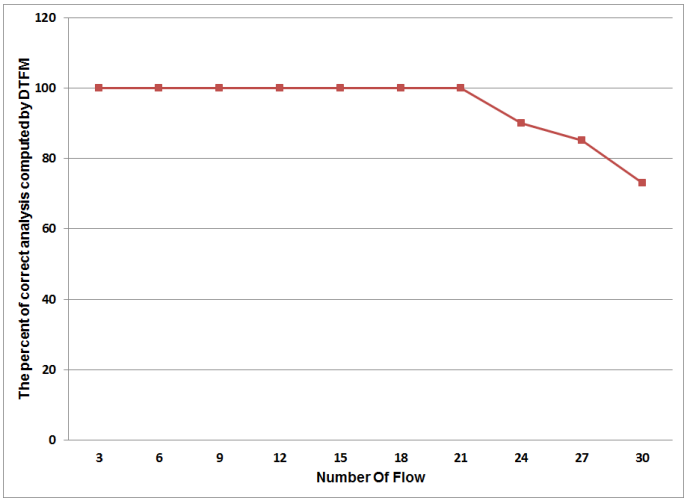

Fig. 4: Correctness of DTFM

2) Correctness of DTFM evaluations : In this experiment, we verify that any task set analyzed as schedulable by DTFM is actually schedulable.

To verify if a task set is actually schedulable, again, we run SHOC simulations. We vary the period of tasks from 1000 ns to 1500 ns with the step of 20 ns. For each configuration, we varied the task mapping such that the number of flows in the network changes (again, the more we have flows, the more we should have contentions). The number of flows in the network is selected from 3 to 30 with a step of 3.

Figure 4 presents the percent of correct analysis computed by DTFM. We can see that, if there is no indirect delays, any task sets said schedulable by DTFM are also said schedulable by SHOC. We also notice that flows with indirect delays are detected and rejected by DTFM: the corresponding task sets are said unschedulable by DTFM. But in some cases, indirect interference may only add a few extra delays and the system could still be schedulable. In such cases, DTFM

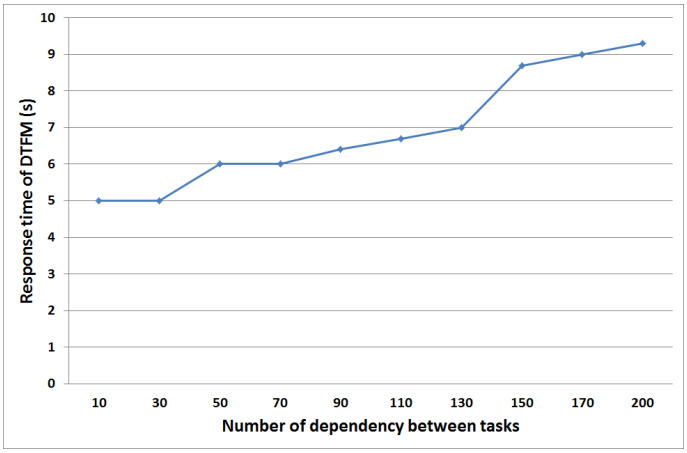

Fig. 5: Response time of DTFM

can make too conservative decisions by considering unschedulable cases which are schedulable.

3) Evaluation of the scalability of DTFM : In order to evaluate the scalability of DTFM, we measured the response time of DTFM to perform the analysis of a given task set.

For such evaluations, we call DTFM for several task sets with different numbers of dependencies ranging from 10 to 200. Figure 5 presents the response times of DTFM and shows how much time the tool needs to verify a task set. As shown in this figure, DTFM takes 5 seconds to analyze a task set with 10 dependencies and up to 9 seconds with 200 task dependencies. Cycle-accurate simulation of a few seconds of the system's execution can take several hours, while the proposed model decides about the schedulability of the whole system faster with cycle accuracy limited to NOC communications.

VI. CONCLUSION

In this article, we introduced DTFM, a dual task and flow model in order to improve the predictability of real-time applications over NoC architectures. DTFM allows us, from the task model and the task mapping, to check the schedulability of the system. Sharing resources between tasks, contentions of the networks and interferences lead to non-deterministic communication delays which make complex the schedulability analysis of tasks. In DTFM, we take into account all of these factors in order to assess the schedulability of tasks.

We use a standard NoC architecture to illustrate DTFM. Such NoCs are unsafe platforms for real-time applications because they introduce various types of latencies which may prevent tasks to meet their deadlines.

We have implemented DTFM into Cheddar. From a set of tasks and their dependencies, we can automatically check and compute the various latencies in the NoC and perform schedulability analysis. Furthermore, when a task set leads to flows with indirect delays, such flows are detected by DTFM and the task set is said unschedulable. Besides DTFM and its corresponding tool, a second contribution is the tool-based validation approach which is based on a multiscale simulator toolset. The validation tool is composed of the Cheddar real-time scheduling simulator and SHOC, a SystemC NoC simulator. The task scheduling produced by Cheddar is used as an input of the SystemC simulator, which is used to measure delays in the NoC. Such multiscale toolset simulator could be applied to investigate the temporal behavior of real-time applications running on various other hardware components such as cache hierarchies. With this multiscale simulator, we made experiments that evaluate DTFM correctness and scalability.

In future work, we will enrich DTFM with new delay models for different and usual task communication protocols and NoCs architectures. We also intend to use DTFM and its associated tools to investigate task mapping and arbitration techniques.

VII. ACKNOWLEDGEMENT

The authors would like to thanks Dr Martha Johanna Sepúlveda for her help on the SHOC simulator. This work and Cheddar are supported by BMO, Ellidiss Technologies, CR de Bretagne, CG du Finistère, and Campus France.

REFERENCES

[1] Z. Shi and A. Burns, "Real-time communication analysis with a priority share policy in on-chip networks," in *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, July 2009, pp. 3–12.

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[3] Z. Shi and A. Burns, "Real time communication analysis for on-chip networks with wormhole switching," in *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, Nov 2008, pp. 161–170.

[4] A. T. Tran and B. Baas, "NoCTweak: a highly parameterizable simulator for early exploration of performance and energy of networks on-chip," VLSI Computation Lab, ECE Department, University of California, Davis, Tech. Rep. ECE-VCL-2012-2, July 2012, http://www.ece.ucdavis.edu/vcl/pubs/2012.07.techreport.noctweak/.

[5] S. P. Azad, P. E. Behrad Niazmand, J. Raik, G. Jervan, and T. Hollstein, "SoCDep^2: a framework for dependable task deployment on many-core systems under mixed-criticality constraints," in *Proceedings of 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, June 2016, pp. 1–6.

[6] B. d'Ausbourg, M. Boyer, E. Noulard, and C. Pagetti, "Deterministic execution on many-core platforms: application to the scc," in *Many-core Applications Researc Community Symposium (MARC)*, Dec 2011.

[7] Z. Shi, "Real-time communication services for networks on chip," Ph.D. dissertation, University of York, Nov 2009.

[8] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[9] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real-time scheduling framework," *ACM SIGAda Ada Letters*, vol. 24, no. 4, pp. 1–8, Dec 2004.

[10] C. Fotsing, F. Singhoff, A. Plantec, V. Gaudel, S. Rubini, S. Li, H. N. Tran, L. Lemarchand, P. Dissaux, and J. Legrand, "Cheddar architecture description language," *Lab-STICC Technical report*, 2014.

[11] M. Sepúlveda, M. Strum, and W. Chau, "Performance impact of QoSS (quality-of-security-service) inclusion for NoC-based systems," in *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2009, pp. 12–14.

[12] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.