Proceedings of the

# 9th Junior Researcher Workshop on Real-Time Computing

## JRWRTC 2015

http://rtns2015.lifl.fr/jrwrtc2015/

**Lille, France**

**November 4-6, 2015**

RÉGION NORD-PAS DE CALAIS

CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille

Université de Lille 1 SCIENCES ET TECHNOLOGIES

# Message from the Workshop Chair

Welcome to the 9th Junior Researcher Workshop on Real-Time Computing, held in conjunction with the 23rd International Conference on Real-Time and Network Systems (RTNS) in Lille, November 2015. The workshop is a platform for junior researchers to present their work, their incursion into uncharted scientific territories, in a relaxed forum encouraging discussion between members of the real-time community.

I would like to first thank the authors for their submission to the workshop. Each has been reviewed and discussed by the members of the Program Committee, to which I express my gratitude for their dedication to the quality of the selected papers and the workshop. Credit is also due to the General Chair of RTNS Julien FORGET (Université de Lille, France), the local committee, Clément BALLABRIGA, Antoine BERTOUT and Giuseppe LIPARI (Université de Lille, France), and the program chairs, Liliana CUCU-GROSJEAN (INRIA Rocquencourt, France) and Nathan FISHER (Wayne State University, USA), for their help and support in organizing the workshop. I also tip my hat to Rob DAVIS (University of York, UK) and Sebastian ALTMEYER (University of Luxembourg, Luxembourg) for their numerous advices in the organisation of the event.

It is with immense delight that, on behalf of the Program Committee, I wish you a pleasant workshop. May the presentations be enjoyable, the posters enlightening, and the discussions fruitful.

Benjamin LESAGE (University of York)

## Program Committee

| | |
|---|---|
| Andreas ABEL | Saarland University, Germany |
| Alessandro BIONDI | Scuola Superiore Sant'Anna, Pisa, Italy |
| Sudipta CHATTOPADHYAY | Linköping University, Sweden |
| David GRIFFIN | University of York, UK |
| Zhishan GUO | University of North Carolina at Chapel Hill, USA |
| Martijn VAN DEN HEUVEL | TU Eindhoven, The Netherlands |
| Benjamin LESAGE | University of York, UK |
| Martina MAGGIO | Lund University, Sweden |
| Ernesto MASSA | State University of Bahia (UNEB), Brazil |
| Cristian MAXIM | Inria Paris, France |
| Mitra NASRI | TU Kaiserslautern, Germany |
| Borislav NIKOLIC | Polytechnic Institute of Porto, Portugal |

# Table of Contents

# Failure tolerance for a multicore real-time system scheduled by PD2

Yves MOUAFO
LIAS-ENSMA
Teleport 2, 1 av. clément Ader
BP 40109,86961
Futuroscope-Chasseneuil
yves.mouafo@ensma.fr

Annie GENIET
LIAS-ENSMA
Teleport 2, 1 av. clément Ader
BP 40109, 86961
Futuroscope-Chasseneuil
annie.geniet@ensma.fr

Gaëlle LARGETEAU
SXlim-SIC, Univ. Poitiers
BP 30179, 86962
Futuroscope-Chasseneuil
glargeteau@sic.univ-poitiers.fr

## ABSTRACT

This work addresses the problem of failure tolerance for real-time applications, running on a multicore hardware architecture. In fact, at any time during the scheduling process, a problem may occur on any of the processor cores, affecting the task that was running on it. We focus on systems composed of periodic independent tasks with simultaneous first release and implicit deadlines. The system is scheduled under the fair algorithm PD2.

Our approach is based on limited hardware redundancy: the system will run on a processor with one more core than required. Then we prove that, if the subtask running on the faulty core is not rescheduled, the application can keep running on the remaining cores without temporal or fairness faults. Moreover, a strategy of restriction and relaxation of tasks deadlines is proposed to ensure the validity and fairness in case of re-execution of the subtask.

## General Terms

Real-time systems, Pfair scheduling, Fault tolerance, limited redundancy, PD2 algorithm

## Keywords

Scheduling, failure, redundancy

## 1. INTRODUCTION

With the introduction of multicore system-on-a-chip architectures for embedded systems, failure tolerance is bound to become a major aspect in application design. In fact, it is well-known [1] that technology scaling exposes electronic devices to external disturbs. The overall effect is a probability that a core of the processor fails during the execution of the application. In this paper, we focus on permanent failures and adopt the classical modeling of a real-time application [2] which consists of a set of $n$ independent periodic tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task $\tau_i$ is submitted to hard temporal constraints and characterized by four temporal parameters: the first release date or offset $r_i$, the worst-case execution time (WCET) $C_i$, the period $T_i$ and the relative deadline $D_i$. Each task $\tau$ consists of an infinite set of instances (or jobs). An important characteristic of the task $\tau_i$ is its utilization $U_i = \frac{C_i}{T_i}$. For any system of tasks $\tau$, we denote $U = \sum_{i=1}^{n}(C_i/T_i)$ the load of the system. We assume that the temporal parameters are known and deterministic, the tasks have simultaneous first releases ($r_i = 0$) and implicit deadlines ($D_i = T_i$).Thus, we denote a task $\tau_i$ with a WCET $C_i$ and a period $T_i$ by $\tau_i < C_i, T_i >$. The system is scheduled under the Pfair algorithm PD2 [3]. Under these assumptions, a necessary and sufficient condition for feasibility is: $U \leq m$ [4] ($m$ denotes the number of processor cores) and PD2 is optimal.

The construction of a PFair scheduling involves dividing each task $\tau_i$ into unitary subtasks. Each subtask $\tau_i^j (j \geq 0)$ has a pseudo-release date $r_i^j = \lfloor \frac{j}{U_i} \rfloor$ and a pseudo-deadline $d_i^j = \lceil \frac{j+1}{U_i} \rceil$. The interval $[r_i^j, d_i^j)$ represents the *feasability window* of the subtask $\tau_i^j$. Subtasks are scheduled in increasing pseudo-deadline order and when the pseudo-deadlines are equal, PD2 uses two additional criterias to determine the priority order between subtasks.

At any time during the scheduling process, a failure may occur on any of the cores, affecting or not a task. It becomes necessary to reorganize the system so that the system keeps on running on the remaining cores without temporal or fairness failures. To provide a protection against a permanent failure, we propose an approach based on limited hardware redundancy, where only one core is added to those requiered. Thus, we prove that, using a fair algorithm such as PD2, if the affected subtask is not rescheduled, limited redundancy guarantees the validity and the fairness of scheduling despite the defection of one core. Moreover, a strategy of rectriction and relaxation of tasks deadlines is proposed to ensure validity and fairness in case of re-execution of the sutask.

The remainder of this paper is organized in four sections. In the next section (Section 2) a state of the art is introduced. Then our modeling of a failure is proposed, and the limited redundancy approach is presented (section 3). Finally, we present our results to ensure failure tolerance when the re-execution of affected subtask is not necessary (section 4) and when it is (section 5).

## 2. STATE OF ART

The classical way to provide failure tolerance on multicore platforms (which are generalized by multiprocessors) is to use time and/or space redundancy. Time redundancy can only protect systems against transient faults. However, space redundancy is useful for transient and permanent failures. The idea is to introduce redundant copies of the elements to be protected (processor or other components), and exploit them in the case of a fault. Authors who have dealt with fault tolerance in multiprocessor architectures [5]

[6] prefer to use the primary backup technique where each task has a primary copy and a backup copy. Both copies are scheduled. The backup is only activated when the primary has failed. This technique has drawbacks, due to the increase of the number of processors, induced by the duplication of tasks. Other authors [7] advocates the removal of some tasks after the failure to maintain a tolerable system load. However, this causes a decrease of system functionality. The number of processors used to implement space redundancy is determinant for the energy consumption of the processor and thus has an environnemental impact.

Our approach aims to minimize the number of redundant processors while maintaining the full functionality of the application despite the failure. Therefore, we use only one additional processor and no deletion of task is envisaged. Only the lost subtask might be re-executed. Instead of using the extra core only after the failure (which will be solved by replacing the damaged core by the backup one), we consider that it is used at the begining. This assumption will be necessary in case of re-execution of the lost subtask(s) and thus, we want to study the scheduling behavior in such context.

## 3. FAILURE MODELING AND TOLERANCE APPROACH

### 3.1 Failure model

We consider that while the application is running, one core of the processor failed. Moreover, the failure is permanent and affects only one subtask: the one that is running on the core which suffers the failure. All or part of this subtask will have to be re-executed. In [8], S. Malo distinguishes two possible scenarios:

- The failure occurs immediately after the context switch. In this case, the application should simply continue execution on the remaining cores. We only have to verify that the reorganization is possible and does not cause a task to miss it deadline;

- The failure occurs during the execution of a task. In this case, three policies are possible: (1) Re-execution of what has been executed since the last backup of the context; (2) The full task is re-executed; (3) The current subtask is simply abandoned and execution continues on the remaining cores.

Therefore it is necessary to distinguish the case where the re-execution of task is required (tolerant scheduling with re-execution) and the case where it isn't (tolerant scheduling without re-execution). *Figure 1* gives an illustration of the scenarios and draws the outline of the context of our contribution. First, we focus on tolerant scheduling without re-execution which means that no subtask is affected or that possibly affected subtask is abandoned. We must ensure that the application can be keep on running without temporal faults. Then, the case of the re-execution of the affected subtask is discussed in the last section of this paper.

### 3.2 Limited redundancy approach

Let S be a system of tasks. In the introduction it was established that S is schedulable if and only if $U \leq m$. To overcome a failure of a core, limited hardware redundancy is a method which consists in providing an additional core: instead of running S on $m$ cores, it will run on $m+1$ cores. Thus, when the failure occurs, the system will remain schedulable on the $m$ remaining cores.

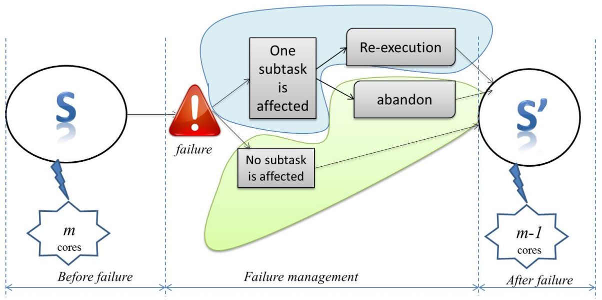


**Figure 1: Failure model**

Let $\tau$ be the system of tasks defined by $\tau = (\tau_1 < 2, 3 >, \tau_2 < 2, 6 >, \tau_3 < 3, 8 >, \tau_4 < 6, 8 >, \tau_5 < 5, 12 >)$. The load $U = 2.54 < 3$, thus $\tau$ is schedulable on a 3 cores processor.

*Figure 2* illustrates the use of limited redundancy to build a tolerant scheduling of the system $\tau$. A failure occurred on the CPU core 1 at time 6 affecting the subtask $\tau_1^4$.

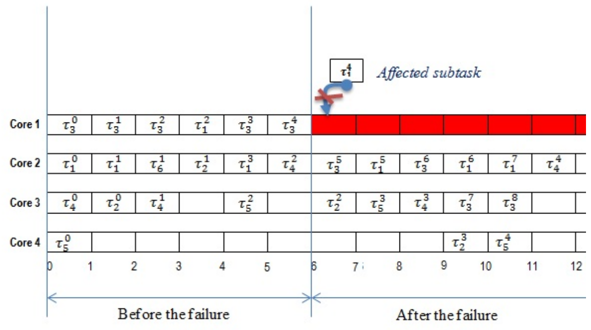


**Figure 2: Example of scheduling with limited redundancy method**

### 3.3 Experimentation

A software prototype FTA (Fault Tolerance Analyser) is designed to simulate scheduling with fault by Pfair algorithms. To get concluding results about our approach, a total of 550 random systems with various proportions of heavy task (i.e $U_i \geq 0.5$) has been generated and submitted to the simulation. The fault occurrence time and the failing core of the processor are randomly chosen and vary from one system to another. The experiment was repeated 50 times on the 550 generated systems. The obtained results show that, no matter the time the failure occurs or the affected core, no matter the system load or the number of heavy tasks ($U_i \geq 0.5$), scheduling with fault without reexecution guarantees the respect of validity and fairness constraints.

In the following section, we establish the proof of this result.

## 4. SCHEDULING WITHOUT REEXECUTION

In this section, we prove that the limited redundancy approach guarantees the validity and the fairness of a tolerant scheduling without re-execution. We introduce the notations and the lemmas needed. The idea of the proof is based on the observation that tasks are scheduled earlier on m+1 cores with failure than on m core without failure (see lemma 2).

## 4.1 Notations and Assumptions

**Notations:**
$Sched^m$: PD2-schedule on a m cores architecture, called m-schedule;
$Sched^{(m+1)\to m}$: PD2-schedule on an architecture with $m+1$ functional cores initially, and $m$ cores after a failure, called reorganized-schedule;
$Sched^m(t_i...t_j)$ : part of the schedule $Sched^m$ from time $t_i$ to the time $t_j$.
$C_m(t)$:List of pending subtasks at time t in $Sched^m$;
$C^u(t)$:List of subtasks released at time t;
$C^e_m(t)$:List of the subtasks scheduled at time t in $Sched^m$;
$C^r_m(t)$:List of the pending subtasks which are not scheduled at time t in $Sched^m$;
$t(\tau_i^j, Sched)$:Date by which the subtask $\tau_i^j$ is executed in $Sched$;
$t_p$: Failure time;
**Assumptions:**
$U \leq m$ thus $Sched^m$ and $Sched^{m+1}$ are valid and fair.

## 4.2 Lemmas of interest

Two lemmas are needed to demonstrate our result. For space reason, we state the two lemmas but provide a sketch of proof only for Lemma 2.

**Lemma 1: At any time t, the list of the pending subtasks in the reorganized-schedule is included in the list of the pending subtasks in the $m$-schedule.**

$$C_{(m+1)\to m}(t) \subseteq C_m(t)$$

This come from the fact that there are more tasks executed on m+1 cores than on m cores.

**Lemma 2:Any subtask is scheduled earlier in the reorganized schedule than in m-schedule.**

$$\forall \tau_i^j, t(\tau_i^j, Sched^{(m+1)\to m}) \leq t(\tau_i^j, Sched^m)$$

PROOF. Let $\tau_i^j$ be a subtask. At any time t,
if $\tau_i^j \in C^e_{(m+1)\to m}(t)$ then $\tau_i^j \in C_{(m+1)\to m}(t)$.
In fact, if a subtask is scheduled, then it was pending.
According the Lemma 1,
$C_{(m+1)\to m}(t) \subseteq C_m(t)$, therefore, $\tau_i^j \in C_m(t)$.
Since at time t a pending subtask is either scheduled or not, we have $C_m(t) = C^e_m(t) \cup C^r_m(t)$.
Two cases to consider:
First case: $\tau_i^j \in C^e_{(m+1)\to m}(t)$ and $\tau_i^j \in C^e_m(t)$.
i.e at time t, $\tau_i^j$ is scheduled in the reorganized schedule and is scheduled in the $m$-schedule. Then, $\tau_i^j$ is scheduled at time t in both schedules:
$t(\tau_i^j, Sched^{(m+1)\to m}) = t(\tau_i^j, Sched^m) = t$
Second case: $\tau_i^j \in C^e_{(m+1)\to m}(t)$ and $\tau_i^j \in C^r_m(t)$.
i.e at time t, $\tau_i^j$ is scheduled in the reorganized-schedule but is not scheduled in the $m$-schedule. In this case, $\tau_i^j$ will be scheduled later in in the $m$-schedule.
Thus, $t(\tau_i^j, Sched^{(m+1)\to m}) = t$ and $t(\tau_i^j, Sched^m) > t$
and then $t(\tau_i^j, Sched^{(m+1)\to m}) \leq t(\tau_i^j, Sched^m$. □

## 4.3 Validity and fairness proof

Our main result is given by Theorem 1 below.

THEOREM 1. ***Any system** $\tau = (\tau_1 < C_1, T_1 >, \tau_2 < C_2, T_2 >, ..., \tau_n < C_n, T_n >)$ **which consists of n periodic and independent tasks, with simultaneous first releases and implicit deadlines feasible under the fair algorithm PD2 on a** $m$ **cores processor and running on** $m+1$ **cores, remains feasible on** $m$ **cores, after the failure of one of the cores without rescheduling the impacted subtask.***

*In other words, the reorganized schedule $Sched^{(m+1)\to m}$ is valid and fair. i.e $\forall \tau_i^j, r_i^j \leq t(\tau_i^j, Sched^{(m+1)\to m}) < d_i^j$.*

PROOF. Let $\tau_i^j$ be a sub-task.
If $t(\tau_i^j, Sched^{(m+1)\to m}) < t_p$ then $t(\tau_i^j, Sched^{(m+1)\to m}) = t(\tau_i^j, Sched^{(m+1)})$, because before the failure, the reorganized-schedule and the (m+1)-schedule are the same.
Since $Sched^{m+1}$ is valid and fair according to the assumptions, we have
$r_i^j \leq t(\tau_i^j, Sched^{(m+1)\to m}) < d_i^j$.
Otherwise $t(\tau_i^j, Sched^{(m+1)\to m}) \geq t_p$:
- The algorithm PD2 can schedule subtasks only if they are pending and so, after their pseudo-release date. Therefore, $\forall \tau_i^j, r_i^j \leq t(\tau_i^j, Sched^{(m+1)\to m})$;
- To prove the second inequality, let us reason by contradiction. Suppose that $d_i^j \leq t(\tau_i^j, Sched^{(m+1)\to m})$.
According to Lemma 3, $t(\tau_i^j, Sched^{(m+1)\to m}) \leqslant t(\tau_i^j, Sched^m)$.
We would then have $d_i^j \leq t(\tau_i^j, Sched^m)$. i.e $\tau_i^j$ misses its pseudo-deadline and thus, $Sched^m$ would not be fair. That is contrary to our initial assumptions (see 4.1).
Conclusion: $\forall \tau_i^j, r_i^j \leq t(\tau_i^j, Sched^{(m+1)\to m}) < d_i^j$ □

## 5. SCHEDULING WITH RE-EXECUTION

To take into account the re-execution of the affected subtask, we introduce the "constrain and release" approach; coupled with limited redundancy, it may ensure tolerance. For the implementation, two methods for calculating the tasks deadlines are considered and an experiment is made. We assume, as a first step, that there is a mechanism which detects and locates the affected core [9] during the slot after its occurrence. Therefore only the lost time unit has to be resumed.

## 5.1 Constrain and release approach

This method consists in starting to scheduling the application on a system of tasks with constrained deadlines, and in releasing the deadlines after failure. Thus the execution begins with smaller feasability windows that will be expanded after the failure. Time margin that is created between deadline and period of a task can be exploited as a tolerance window to possibly re-execute a subtask. *Figure 3* illustrates this approach. We build from the starting system (S) (schedulable on m cores), a system with constrained deadlines (S') that runs on m+1 cores. When the failure occurs, the impacted subtask is re-executed in the tolerance window and deadlines are released. We obtain a system S" which continues running on the remaining m cores.

The problem is here to determine how to compute the constrained deadlines, and next how to reconfigure the failed system. In the following paragraphs, we propose some ideas.

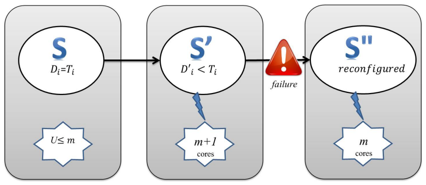


**Figure 3: Restriction and relaxation approach**

## 5.2 Deadlines calculation

Two approaches are possible. The first consists in exploiting the idle time units of the schedule. Those time units are distributed equally between all instances of all tasks. The proportion of time units allocated to an instance is derived from its period [10].
The second method consists in simulating the addition to each task of a further subtask which represents the re-execution of an affected subtask. The deadline of the task corresponds to the last but one subtask pseudo-deadline.

## 5.3 Reconfiguration protocol

When a failure occurs at time $t_p$, it affects a subtask $\tau_{i_0}^{'j_0}$ that belongs to the instance $k_0 = \lfloor \frac{j_0}{C_{i0}} \rfloor$ of the task $\tau_{i_0}$. Constrained deadlines in the system S' must be released for the remainder of the schedule and the failing subtask must be re-executed. Let denote by $r_i^{'j}$ and $d_i^{'j}$ respectively the pseudo-release date and the pseudo-deadline of a subtask $\tau_i^{'j}$ in the running constrained system S'. $r_i^j$ and $d_i^j$ are these settings for the corresponding subtask $\tau_i^j$ in the initial system with implicit deadlines. The reconfiguration happens as follows:

**- For the subtasks of non-affected tasks $\tau'_i (i \neq i_0)$:**

back to initial settings for non-yet executed instances

$$(r_i^{'j}, d_i^{'j}) \longmapsto (r_i^j, d_i^j) \text{ , } i \neq i_0$$

**-For the subtasks of the affected task $\tau'_{i_0}$:**

keep the settings for the current instance

$$j_0 < j < k_0 C_{i0} \Longrightarrow (r_{i_0}^{'j}, d_{i_0}^{'j}) \longmapsto (r_{i_0}^{'j}, d_{i_0}^{'j});$$

back to initial settings for next instances

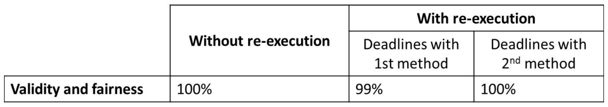
$$j \geq k_0.C_{i0} \Longrightarrow (r_{i_0}^{'j}, d_{i_0}^{'j}) \longmapsto (r_{i_0}^j, d_{i_0}^j);$$

re-execute the affected subtask $\tau_{i_0}^{j_0}$ in the tolerance windows:

$$(r_{i_0}^{'j_0}, d_{i_0}^{'j_0}) \longmapsto (D'_{i_0}, T_{i_0}).$$

## 5.4 Experimental results

The experimental study, which focused on the same systems than the approach without re-execution (see section 3.3) provided promising results (see Figure 4): whatever the time of the failure, the affected core or the affected task, with the deadlines calculated by the second method, the fairness and the validity of the scheduling seems to be guaranteed. This is also true for most cases with deadlines calculated by the first method. However there are marginal invalid systems that need further extensive study.



| | Without re-execution | With re-execution | |
|---|---|---|---|
| | | Deadlines with 1st method | Deadlines with 2nd method |
| Validity and fairness | 100% | 99% | 100% |

**Figure 4: Experimental results**

## 6. CONCLUSION AND PERSPECTIVES

In this paper, we proposed the limited hardware redundancy approach to protect a real-time system against a permanent failure of one core. We proved that this approach guarantees validity and fairness of a schedule with failure without re-execution.

When re-execution is necessary, we proposed to constrain the deadlines of tasks before failure and to release them after. This method provided us promising results that we plan to demonstrate in future work. Moreover, we will study the case where re-execution of several subtasks is necessary. We will finally try to determine the maximum tolerable delay between the occurrence of a failure and its detection.

## 7. REFERENCES

[1] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli,M. Peri, and S. Pezzini, Fault-tolerant platforms for automotive safety-critical applications. *In Proceedings of the 2003 international conference on Compilers, Architecture and Synthesis for Embedded Systems* , pages 170 to 177, 2003.

[2] A. Choquet-Geniet, S. Malo, Scheduling an aperiodic flow within a real time system using Fairness properties *ARIMA Journal, vol. 18*, pp. 93 to 116, 2014.

[3] J. Anderson, A. Srinivasan, A New Look at Pfair Priorities *Rap. tech.TR00 023, University of North Carolina at Chapel Hil* , sept. 1999

[4] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15, p.600 to 625*,1996

[5] A. Bertossi, L. Mancini, A. Menapace, Fault-Tolerant Rate-Monotonic first-fit scheduling in hard real-time systems. *IEEE Transactions on parallel and distributed systems 10,9 (sept 1999) 934-935.*

[6] S. Ghosh, R. Melhem, D. Mosse Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. *IEEE Transactions on parallel and distributed systems 8,3 (march 1997) 272-283.*

[7] T. Megel, Placement, ordonnancement et mécanismes de migration de tâches temps-réel pour des architectures distribuées multicoeurs. *Thèse de doctorat de l'université de Toulouse, avril 2012.*

[8] S. Malo, Contribution à l'ordonnançabilité des applications temps-réel multiprocesseurs. *Thèse de doctorat de l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique*,décembre 2010.

[9] E. Chanthery, Y. Pencole, Modélisation et intégration du diagnostic actif dans une architecture embarquée *Dans Journal Européen des Systèmes Automatisés*, MSR 2009, pages 789 to 803, 2009.

[10] Y. Mouafo, A. Geniet-Choquet, G. Largeteau-Skapin Robustesse des applications temps-réels multicoeurs *Actes de l'école d'été temps-réel 2015, pg 143-146.*

# Discussion on the Spectral Analysis of Real-Time Multi-Path Tasks

Fabrice Guet, Luca Santinelli and Jérôme Morio
ONERA Toulouse - The French Aerospace Lab, Toulouse
{fabrice.guet|luca.santinelli|jerome.morio}@onera.fr

## ABSTRACT

The Worst-Case Execution Time (WCET) is applied to guarantee the safety critical real-time systems. The increasing complexity of todays real-time systems requires new approaches for evaluating the WCET of a task. Measurement Based Probabilistic Timing Analysis provides safe probabilistic WCET with the Extreme Value Theory (EVT). The theoretical applicability of the EVT relies on assumptions that has to be verified. The spectral analysis of multi-path task proposed in this paper aims at defining the task timing behavior and deducing its worst-case execution path. It gives also measurement guidelines to ensure the EVT applicability in the complex case of multi-path tasks.

## 1. INTRODUCTION

The safety of real-time system relies on hard timing constraints that have to be respected at every system execution [4]. System predictability is achieved by determining the Worst-Case Execution Time (WCET) of each task that depends on their implementation and the hardware architecture that runs the task. It is the schedulability analysis that guarantees timing constraints by verifying execution requirements of each task by making use of their timing evaluation.

The new functionalities that are offered by multi-core microprocessors e.g., cache memories, pipelines and so on, increase the complexity of the Critical Real-Time Systems. While they enable relevant gains in terms of average performances e.g., energy consumption and timing computation, it becomes hard to guarantee the worst-case performances of such architectures. Determining every state of the system is quite impossible, moreover for the same execution condition the task execution time could be variable due to different interferences happening [12].

Todays systems complexity makes harder to produce WCET estimates through Static Timing Analyses [15], endangering the reliability of static models and impacting estimates confidence [2]. Alternative to deterministic approaches, probabilistic ones tend to emerge from the requirements of the DO-178-B certification levels in aeronautics which corresponds to probabilistic levels. Contrary to deterministic approaches, probabilistic approaches provide probabilistic Worst-Case Execution Time (pWCET) estimates[1].

In particular, measurement based analysis applies the Extreme Value Theory (EVT) [7] for modeling extreme events such as extreme execution times in order to derive pWCETs. From system performance measurements, in this case execution times, the EVT provides pWCETs with negligible cost by reducing the analysis effort, because it does not require any system model. Limitation is the proof of the EVT theoretical applicability and so the reliability of the pWCET estimates [10, 14]. As most of statistical models, the EVT requires independent and identically distributed (i.i.d.) measurements to produce pWCETs that is quite difficult to assume in practical problems as those including real-time computing systems.

[1]pWCETs are alternative to deterministic Worst-Case Execution Times as probabilistic distributions with multiple execution times, each with a probability of happening.

Current researches focus on the independence problem of the measurements by introducing system randomness e.g., implementing random replacement policies in cache memories. On the one hand, this randomization may be regarded as harmful for real-time system safety [13] and on the other hand, strict independence is not required regarding mathematical conditions to consider the EVT [6].

Considering the timing behavior of a real-time task as a process, EVT conditions are applicable for stationary processes i.e. processes which follow the same probabilistic law. Spectral analysis in signal processing enables to stress the underlying law of the process in presence of noise and could be suitable to ensure the applicability of the EVT. The application of the spectral analysis aims at proving stationarity in the case of tasks with multiple control flow instructions defining multi-path tasks, by identifying the laws composing the task.

### *Contributions.*

In this paper we focus on the measurement based probabilistic analyses and especially on the stationarity aspect for guaranteeing the EVT applicability for the WCET problem, especially for multi-path tasks [5]. The analysis particularly stresses software control flows that drive non stationarities in the statistical analyis of real-time tasks. We face non time-randomized systems and multi-path tasks with their challenges for the EVT application.

### *Outline.*

In section 2, we introduce stationarity analysis concepts with a possible interpretation for real-time computing systems. In section 3, we introduce a framework for the spectral analysis of real-time tasks and a method to compose the spectral analysis of different tasks. In section 4, we make use of the introduced notions in a case study.

## 2. A STATIONARITY ANALYSIS FOR REAL-TIME COMPUTING SYSTEMS

Execution time measurements $C_i$ are collected according to end-to-end measurements of a task at discrete instant times $i \in [\![1; N]\!]$, with $N$ the number of execution time measurements. The collection of measurements is a trace $\mathcal{T}$ of length $N$ defined as the sequence of execution time measurements: $\mathcal{T} = (C_i)_{i\in[\![1;N]\!]}$. From $\mathcal{T}$, it exists a finite number of different possible execution times $C_{(k)}$ and the upper-ordered sequence of execution times $\mathcal{T}^{\uparrow}$ is the sequence of the $n$ different execution times such that $C_{(1)} < \ldots < C_{(n)}$.

Randomness that occurs in modern multi-core microprocessors recorded in measurements motivates the definition of a random variable for representing the task execution time.

Definition 1 (Execution Time Profile $\mathcal{C}$). *Given a trace $\mathcal{T}$, the Execution Time Profile (ETP) $\mathcal{C}$ is the discrete random variable defined on the finite number $n$ of possible execution times $C_{(k)}$ and we denote $\Omega = (C_{(k)})_{k\in[\![1;n]\!]}$ the set of the different execution times.*

*The ETP is usually depicted with the Empirical Density Function or histogram of $\mathcal{T}$ as the discrete function $p_{\mathcal{C}}$ associating an execution time $C_{(k)}$ in $\Omega$ to its probability density $p_k$:*

$$p_{\mathcal{C}}(C_{(k)}) = p_k \overset{def}{=} \frac{1}{N}\sum_{i\leq N} \mathbf{1}_{C_i = C_{(k)}}. \tag{1}$$

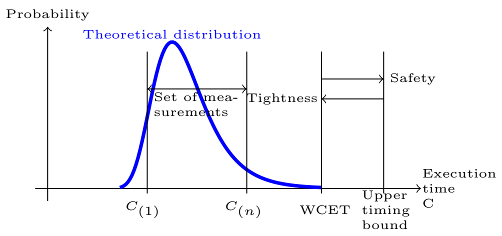


**Figure 1: Overview of the WCET problem. Example of a timing probabilistic profile of a task.**

While the true task Execution Time Profile should follow a complete theoretical distribution like in Figure 1, the Empirical Density Function is only defined on the set of execution time measurements smaller than the theoretical set of possible execution times due to the difficulty to observe extreme execution times i.e. before $C_{(1)}$ and after $C_{(n)}$.

If there is no obvious reason to assume i.i.d. measurements, one has to consider stationary measurements in order to apply the EVT [6]. In particular, for i.i.d. or stationary weakly dependent measurements whose average probabilistic law has a tail which tends slowly to zero, then the EVT applicability is ensured. Within the scope of non time-randomized hardware architectures, independence is quite hard to assume because of past states in memory units [10] whereas the hypothesis of weakly dependent stationary measurements is more realistic. Hence, we investigate the stationarity in real-time computing systems.

Definition 2 (Strictly Stationary Trace). *A trace $\mathcal{T} = (C_1, C_2, \ldots)$ is a strictly stationary trace if for all $j, k, l$, the set of execution times $C_j, \ldots, C_{j+k}$ has the same probabilistic law as the set $C_{l+j}, \ldots, C_{l+j+k}$.*

Given a trace, if Definition 2 is verified, then there is strong evidence that measurements are identically distributed (i.d.) from the same probabilistic law (e.g., Gaussian, Gumbel, Weibull etc). Since in practice the law is not known there is no chance to prove the i.d. hypothesis so that we consider stationarity instead.

Evidences of non stationarities in real-time computing systems may be

**Deterministic trends:** the execution time evolves according to a function over the time e.g., a *for* loop whose upperbound evolves deterministically over the time.

**Random walks:** this case may not happen because the current measure cannot be a sum of the preceding one and a random one.

**Seasonalities:** in multi-path programs, one path is executed (one probabilistic law) during an amount of time then another one and so on.

Stationarity is essential in statistical analyses but it is a usually assumed hypothesis in a very wide range of applications. The problem is even more diffcult because there is no practical definition and it sometimes relies on subjective analyses [11]. That is why we consider a statistical metric to evaluate the trace stationarity which are the Kwiatowski Philips Schmidt Shin (KPSS) test [9].

In the real-time computing system domain such non stationarities exposed above could be controlled with measurement rules given by the following spectral analysis.

## 3. A SPECTRAL ANALYSIS OF MULTI-PATH TASKS

In the case of the multi-path task is known, we intend to provide stationarity. Let us consider $\tau$, a multi-path task and whose paths are denoted $\pi_i$ with $i \in [\![1;\Pi]\!]$, $\Pi$ the number of paths in $\tau$. For one executed path $\pi_i$, a set of possible execution times $\overline{\Omega_i}$ cannot be observed. As a consequence, for all possible execution time $C \in \overline{\Omega_i}$, the probability to observe $C$ knowing that the path $\pi_i$ is executed is $P(C|\pi_i) = 0$. Thus, execution times in $\overline{\Omega_i}$ do not follow the same probabilistic law as those in $\Omega_i$ and so are not identically distributed.

With Definition 2, stationarity would be achieved if measurements are collected for one executed path. $\forall i \in [\![1;\Pi]\!]$, the execution time of $\tau_i$ may be seen as a natural frequency of $\tau$. Consequently, $\tau$'s timing behavior could be the sum of its natural frequencies, as in signal theory. Consequently, a complete temporal representation of multi-path tasks is required to ensure the investigation of every path for the WCET problem:

Definition 3 (Task Spectral Representation (TSR)). *We consider a task $\tau$ whose paths are denoted $\tau_i, \forall i \in [\![1;\Pi]\!]$, where $\Pi$ is the number of paths. $\forall i \in [\![1;\Pi]\!]$, we denote $\Omega_i$ the set of possible execution times that can take $\tau_i$. The Task Spectral Representation of $\tau$ is a function of the continuous time $t \in \mathbb{R}^+$:*

$$|\tau|(t) = \frac{1}{\Pi}\sum_{i=1}^{\Pi} \delta_i(t), \tag{2}$$

*with $\delta_i$ the Dirac function that equals to 1 if $t = T_i$, where $T_i$ is a natural frequency of $\tau$ and 0 otherwise.*

The natural frequencies of a task, as defined in signal theory, are either known or deduced from experiments as the execution time that each path takes the most to complete.

For instance, for $\tau$ running on a fully deterministic single-core microcontroller, if every path of $\tau$ is executed then $\forall i \in [\![1;\Pi]\!]$, $card(\Omega_i) = 1$ *and* $\Omega_i = \{T_i\}$, $T_i$ a natural frequency, so the ETP is equal to the TSR of $\tau$. In this case, the TSR is a complete timing representation of $\tau$. However, for a low deterministic tightly coupled multi-core microprocessor, $\forall i \in [\![1;\Pi]\!]$, $card(\Omega_i) > 1$ and so the ETP is different from the spectral representation of $\tau$.

The spectral analysis of software tasks is a formalism to have an overview of the timing behavior of the task and manage to collect stationary execution time measurements.

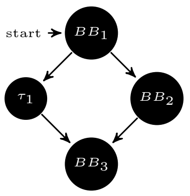


**Figure 2: Task $\tau$'s graphical representation.**

## *TSR Composability.*

To ease the worst-case path research, one would like to compose TSRs of known inner tasks included in a task that is the only scheduled. For instance, given a $(\Pi_1 + 1)$-path task $\tau$, that executes either the known $\Pi_1$-path task $\tau_1$ or a basic block (sequences of instructions that have no control flow instructions) (BB) [3] like in Figure 2, then the TSR $|\tau|(t) = |\tau_1|(t - \Delta t) + \delta_2(t)$. The dependent relationship beween $BB_1$, $\tau_1$ and $BB_3$ leads to the temporal translation of the TSR $\tau_1$ of $\Delta t$ which is the duration of BBs 1 and 3.

Property 1 (TSR Composability). *Considering a multi-path task $\tau$ composed of 2 inner tasks $\Pi_1$-path task $\tau_1$ and a $\Pi_2$-path task $\tau_2$ whose TSRs are known then the TSR $|\tau|$ is*

- $|\tau_1| + |\tau_2|$ *for independent inner tasks,*
- $|\tau_1| \circ |\tau_2| = \frac{1}{\Pi_1+\Pi_2} \sum_{i=1}^{\Pi_1} \sum_{j=1}^{\Pi_2} \delta_{1_i}(t - T_{2_j})$ *for dependent inner tasks, where $T_{2_j}$ is such that $\delta_{2_j}(T_{2_j}) = 1$, the natural frequencies of $\tau_2$.*

Remark 1. *$\circ$ is the convolution operator and is symmetric.*

Knowing all TSRs that compose the global task, we deduce analytically the worst-case path by composing all the worst-case paths of every inner tasks. Using the TSR enables to have an overview of the timing behavior of the task to deduce faster the pWCET with the EVT and also guaranteeing stationary measurements for applying the EVT.

Application of the TSR is presented next for the *ns* case study from the Mälardelen benchmarks [1].

## 4. THE *NS* CASE STUDY

The *ns* WCET benchmark task searches a key in a 4-dimensional array of 5 elements each. It is a linear search through the array so depending on the key to search there are as many paths as the number of elements in the array. We only consider the farthest element of the last dimension, which makes 4 different paths $((5 - 1) \times 5^3)$ each with significant different number of instructions which allows to better visualize the natural frequencies of *ns*. Thus, the *ns* TSR is

$$|ns|(t) = \frac{1}{4} \sum_{i=1}^{4} \delta_i(t) \tag{3}$$

where $\forall i \in [\![1;4]\!], \delta_i$ is also defined by its respective *ns* input $IN$ which is the key $KEY_i$ to search. Thus we define, $\forall i \in [\![1;4]\!], IN(\delta_i) = KEY_i$ where $KEY_i$ is the farthest key in the farthest dimension of the $i^{th}$ element of the first dimension.

Knowing that every path has the same length it is also possible to define $|ns|$ with the composability (Property 1):

$$|ns|(t) = \frac{1}{4}\left[\delta_1(t) + \delta_1(t - \Delta t) + \delta_1(t - 2\Delta t) + \delta_1(t - 3\Delta t)\right], \tag{4}$$

where $\delta_1$ is the timing function of the first path and $\Delta t$ the time spent to explore once three dimensions.

## *Hardware Platform.*

The platform running the task has two Intel®Xeon®E5620 2.4 GHz sockets, each one with four cores and three levels of cache. The first two levels (L1 and L2) are private to each core, while the last level (LLC, equivalently L3) is shared to the cores belonging to the same socket.

## *Execution Conditions.*

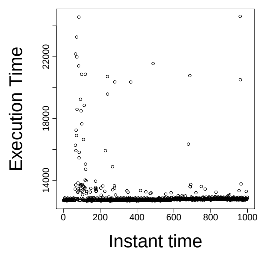
The task is running periodically in isolation on one core i.e. there is no other task running on the core and no interrupt (Irq). To guarantee the real-time task execution, we set its scheduling policy to the Linux SCHED_FIFO policy. The task is executed under different conditions:

- One input: the same input for all instant time.
- Random inputs: the key is randomly modified at each instant time.
- Periodic inputs (100 and 200 iterations case): the key is randomly modified every 100/200 instant times.

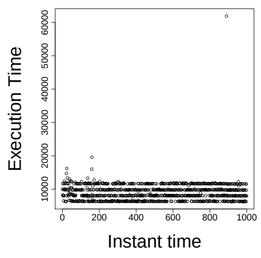
Those execution conditions may not be realistic in practice, they stress specific conditions that may be exerced for timing analysis purposes.

## *Results.*

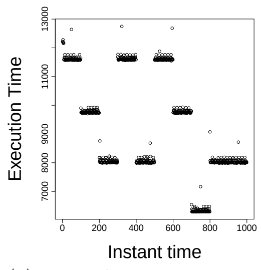
We now present the results of the experiments with remarks about stationarity.



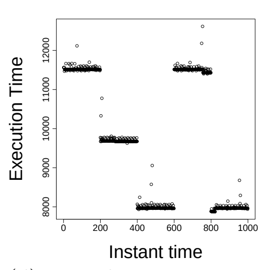
(a) One input



(b) Random inputs



(c) Periodic inputs: 100 iterations

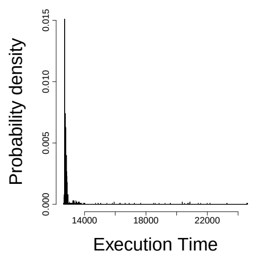


(d) Periodic inputs: 200 iterations

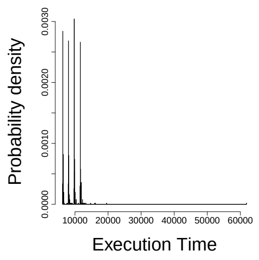
**Figure 3: Trace of execution time measurements for every execution condition.**

Plots of traces for every execution condition defined above are presented in Figure 3 where execution times are measured in number of cpu cycles. Non determinism is particularly stressed in Figure 3(a) i.e. for one input value, the task execution time takes several different values due to the hardware components. The periodic cases show that the greatest execution time measurement of each path is always lower than the lowest execution time measurement of the following path.

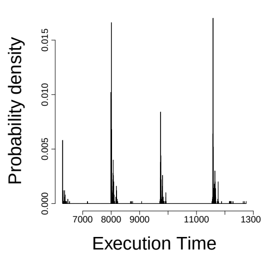
ETPs in Figure 4 support the idea of modeling task execution time with a random variable for non deterministic hardware



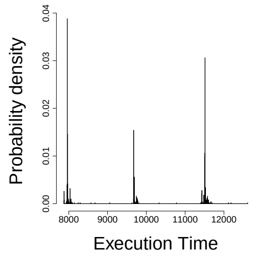
(a) One input



(b) Random inputs



(c) Periodic inputs: 100 iterations



(d) Periodic inputs: 200 iterations

**Figure 4: Execution Time Profile for every execution condition.**

architectures. They highlight the number of different executed paths according to Definition 3: one path in the first case, four paths in the second and third case, and three paths in the fourth case. Filtering the execution times except the peaks of highest probability density (Equation (1)), we deduce the experimental TSR and then the number of probabilistic laws characterizing the timing behavior of the task for the considered architecture.

In addition, we deduce from the ETP in the random case the natural frequencies of *ns* and the value of $\Delta t$ in Equation (4), results are in Table 1.

| Path $\delta_i$ | Natural Frequency $T_i$ | $\Delta t$ |
|---|---|---|
| 1 | 6300 | 0 |
| 2 | 8000 | 1700 |
| 3 | 9750 | 1750 |
| 4 | 11600 | 1850 |

**Table 1: Determination of the natural frequencies of *ns* in the random case.**

Hypothesis of a constant $\Delta t$ is confirmed with an error rate of 9%.

With regard to the spectral analysis of *ns*, stationarity would only be achieved in the first case because only one path is executed.

| Trace $\mathcal{T}$ | One path | Random | Period 100 | Period 200 |
|---|---|---|---|---|
| KPSS | 0.187 | 0.243 | 3.933 | 3.008 |

**Table 2: Results of the KPSS test for the different traces of execution time.**

Results of the stationarity analysis by the KPSS test, presented in Table 2, highlight the stationarity in the one path case, as we conjectured it in Section 3, and also in the random case. The random case is actually a mixture law that takes into account the four laws composing *ns*. Blocks of execution time measurements are considered to belong to the same mixture law and explaining why stationarity is achieved. Non stationarities in the last two cases are well detected.

## 5. CONCLUSIONS

Stationarity is often assumed in stochastic system analyses while it is required to apply statistical models, like the EVT, and derive reliable probabilistic bounds.

While there is no systematic way to prove stationarity, the spectral analysis of real-time multi-path tasks provides a new paradigm to study the stationarity of the timing behavior of a task.

The spectral analysis also intends to guarantee the task path coverage required to have safe statistical models.

## References

[1] *WCET project/ Benchmarks*, 2013.

[2] S. Altmeyer, B. Lisper, C. Maiza, J. Reineke, and C. Rochange. WCET and mixed-criticality: What does confidence in WCET estimations depend upon? In *15th International Workshop on Worst-Case Execution Time Analysis, WCET 2015, July 7, 2015, Lund, Sweden*, pages 65–74, 2015.

[3] G. Bernat, A. Colin, and S. Petters. pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems. Technical report, 2003.

[4] G. C. Buttazzo. *Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications.* The Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston, 1997. 3rd edition 2000.

[5] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *the 24th Euromicro Conference on Real-Time Systems*, Pise, Italy, July 2012.

[6] P. Embrechts, C. Klüppelberg, and T. Mikosch. *Modelling extremal events for insurance and finance.* Applications of mathematics. Springer, Berlin, Heidelberg, New York, 1997.

[7] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-Based WCET Estimation and Validation. In *9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*, pages 1–11, 2009.

[8] M. Hillary. You can't control what you can't measure, or why it's close to impossible to guarantee real-time software performance on a cpu with on-chip cache. Technical report, Applied Microsystems Corp, 2002.

[9] D. Kwiatkowski, P. C. B. Phillips, P. Schmidt, and Y. Shin. Testing the null hypothesis of stationarity against the alternative of a unit root : How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1-3):159–178, 00 1992.

[10] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A Trace-Based Statistical Worst-Case Execution Time Analysis of Component-Based Real-Time Embedded Systems. In *16th IEEE International Conference on Emerging Technology and Factory Automation (ETFA11), WiP session*, September 2011.

[11] R. Manuca and R. Savit. Stationarity and nonstationarity in time series analysis. *Phys. D*, 99(2-3):134–161, Dec. 1996.

[12] V.-A. Paun, B. Monsuez, and P. Baufreton. On the Determinism of Multi-core Processors. In C. Choppy and J. Sun, editors, *1st French Singaporean Workshop on Formal Methods and Applications (FSFMA 2013)*, volume 31 of *OpenAccess Series in Informatics (OASIcs)*, pages 32–46, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[13] J. Reineke. Randomized caches considered harmful in hard real-time systems. *LITES*, 1(1):03:1–03:13, 2014.

[14] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *14th International Workshop on Worst-Case Execution Time Analysis*, pages 21–30, 2014.

[15] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.

# Scheduling of parallel applications on many-core architectures with caches: bridging the gap between WCET analysis and schedulability analysis

Viet Anh Nguyen, Damien Hardy, and Isabelle Puaut
University of Rennes 1/IRISA, France
anh.nguyen@irisa.fr, damien.hardy@irisa.fr, isabelle.puaut@irisa.fr

## ABSTRACT

Estimating the worst-case execution time (WCET) of parallel applications running on many-core architectures is a significant challenge. Some approaches have been proposed, but they assume the mapping of parallel applications on cores already done. Unfortunately, on architectures with caches, task mapping requires a priori known WCETs for tasks, which in turn requires knowing task mapping (i.e., co-located tasks, co-running tasks) to have tight WCET bounds. Therefore, scheduling parallel applications and estimating their WCET introduce a chicken and egg situation. In this paper, we address this issue by developing an optimal integer linear programming formulation for solving the scheduling problem, whose objective is to minimize the WCET of a parallel application. Our proposed static partitioned non-preemptive mapping strategy addresses the effect of local caches to tighten the estimated WCET of the parallel application. We report preliminary results obtained on synthetic parallel applications.

## 1. INTRODUCTION

Many-core architectures have become mainstream in modern computing systems. Along with them, parallel programming frameworks have been developed to utilize the power of many-core architectures. However, employing many-core architectures in hard real-time system raises many concerns. One significant issue is to precisely estimate the worst-case execution time (WCET) of parallel applications. WCET estimation methods for many-core architecture must take into account not only program paths and architecture (as addressed in WCET estimation methods for uni-core architecture [6]) but also resource contentions (i.e., bus contention and shared memory contention [3]). Moreover, when analyzing the timing behavior of parallel applications, these methods also have to consider the application's properties (i.e., multitasking, inter-task communication and synchronization).

Some approaches reported promising results in estimating the WCET of parallel applications running on many-core architectures. Ozaktas et al. [4] combine the estimated worst-case stall times caused by inter-task synchronization with the tasks' estimated WCETs to compute their worst case response time. Then, the WCET of the parallel application is estimated as the worst case task's response time. In another way, Potop-Butucaru et al. [5] integrate code sections of tasks running on cores as well as communications between them to produce an unified control flow graph. Then, the classical implicit path enumeration technique (IPET [6]) is applied to estimate the WCET of the parallel application. These two methods assume the mapping of the parallel applications on cores a priori known. However, the mapping of the parallel application influences the worst-case response time of the tasks, and hence affects the WCET of the entire parallel application.

As an illustration, let us consider a parallel application containing three tasks *T1, T2,* and *T3* to be mapped onto on a two-core architecture with a private cache on each core. Let us assume that *T1* and *T2* access the same memory block $m$, and that *T1* and *T2* are independent from *T3*. Let us consider two mappings: (i) *T1* and *T2* are assigned to one core and *T2* runs after *T1*, while *T3* is assigned to the other core; (ii) *T1* is assigned to one core, while *T2* and *T3* are assigned to the other core and *T2* runs after *T3*. In the first case, *T2*'s access to block $m$ is a hit because $m$ was loaded by *T1*. Therefore, the WCET of the parallel application in the first case is smaller than in the second case. This small example shows that the WCET of the entire parallel application highly depends on the mapping of its tasks on the cores. This motivates the need for optimal scheduling/mapping of the parallel application to tighten its estimated WCET.

In the literature, many scheduling approaches for parallel applications running on many-core architectures have been proposed [1]. However, most of them consider tasks' WCETs as constant values. As explained above, tasks' WCETs highly depend on the mapping of the applications tasks, due to the effect of private caches. Therefore, scheduling a parallel application without considering the effect of private caches on tasks' WCETs is suboptimal. Consequently, scheduling a parallel application and estimating its WCET are interdependent problems and have to be jointly solved for getting tight estimated WCET of the parallel application.

Ding et al. [2] propose a task scheduling method that minimizes shared cache interferences to tighten estimated WCETs. Their approach is different with us since we consider the effect of private caches in the task scheduling process. Additionally, the communication cost between tasks, which varies depending on task mapping, is not taken into account in [2].

In this paper, we propose a static scheduling solution for an isolated parallel application running on a many-core architecture. Our proposed scheduler not only respects dependence constraints between tasks (i.e., communications and synchronizations) but also takes into account the effect of local caches on tasks' WCETs. We develop an optimal integer linear programming model for solving the task scheduling problem, whose objective is to minimize the WCET of the parallel application. To the best of our knowledge, we are

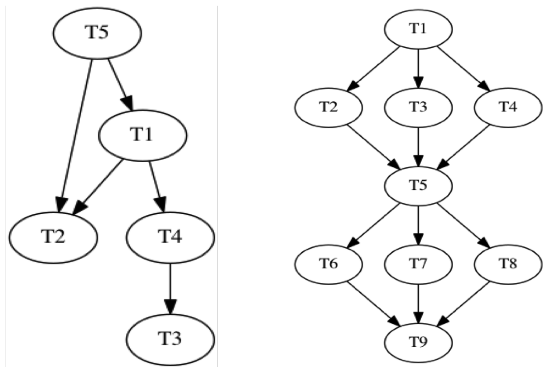


Figure 1: Arbitrary (left) and fork-join (right) task graphs

the first ones considering the effect of private caches on tasks' WCETs when scheduling parallel applications on many-core architectures. Our proposed scheduling approach is a partitioned non-preemptive scheduling approach: tasks are not allowed to be migrated and preempted, which prevents the system from suffering from hard-to predict migration and preemption costs (i.e., mainly caused by losses of working sets stored in local caches).

The paper is organized as follows. Section 2 introduces the application and architecture model, and presents the problem formulation. Section 3 presents an ILP formulation for solving the identified scheduling problem. Section 4 gives preliminary experimental results. Finally, we summarize the content of paper and give directions for future work.

## 2. MODEL AND PROBLEM FORMULATION

**Application model.** The parallel application is represented as a directed acyclic task graph (as illustrated in Fig. 1). Following the terminology used in [2], in these graphs, nodes represent tasks (i.e., pieces of code without communication or synchronization inside), and edges represent communications or synchronizations (precedence relations) between pairs of tasks. For each edge, the volume of transmitted data (zero for synchronizations) is known. For example, in the task graph illustrated in Fig. 1, the arrow from node *T5* to node *T1* means that *T1* is not authorized to execute before *T5* ends. We consider two instances of the task models: (1) arbitrary model, which does not constrain communications and synchronizations between tasks; (2) the popular fork-join model.

**Architecture model.** Our proposed scheduler applies to many-core architectures equipped with private caches, including the one depicted in Fig. 2. In the figure, cores are homogeneous and have a private cache. Our model can deal with any type of cache (instruction cache, data cache, and unified cache).

**Problem formulation.** Our scheduling method takes as input the task graph of an isolated parallel application and the following information: (a) the communication costs between tasks (when running on the same core, and when running on different cores); (b) tasks' WCETs when running alone as well as tasks' WCETs when running immediately after another task on the same core (to consider the effect of private caches). As a result the method produces a static partitioned non-preemptive schedule that determines on which cores each task is assigned, as well as a static

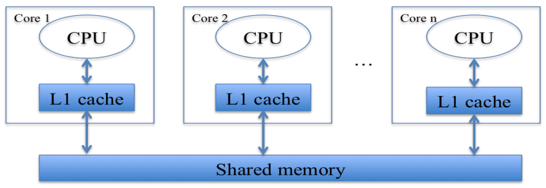


Figure 2: An example of many-core architecture with private caches

schedule on each core. The produced schedule minimizes the WCET of the parallel application.

## 3. ILP FORMULATION OF TASK SCHEDULING/MAPPING PROBLEM

Due to space limitations, only the main ILP constraints are presented hereafter. In the ILP formulation, we use uppercase letters for constants, and lowercase letters for variables to be calculated by the ILP solver. The solution is a set of variables that indicates static task mapping on cores and static task scheduling on each core.

**Base constraints for task mapping and scheduling.** We define a 0-1 variable $m_i^k$ to indicate whether task $t_i$ is assigned to core $k$ or not. Since the proposed scheduler is partitioned, each task is mapped to exactly one core, therefore:

$$\sum_{k \in K} m_i^k = 1. \tag{1}$$

In equation (1), $K$ represents the set of cores. Besides, we define a 0-1 variable $o_{j \to i}$ to determine whether task $t_i$ runs right after task $t_j$ or not, and a 0-1 variable $f_i^k$ to decide whether task $t_i$ is the first task running on core k or not. Since the produced schedule is non-preemptive, a task has at most one task running right after it, thus:

$$\sum_{i \in T - \{j\}} o_{j \to i} \leq 1. \tag{2}$$

In equation (2), $T$ represents the set of tasks. Additionally, one core has at most one first-running task, therefore, the following constraint is introduced:

$$\sum_{i \in T} f_i^k \leq 1. \tag{3}$$

**Further constraints for task mapping/scheduling.** The objective of the scheduling problem is to minimize the WCET of a parallel application. Let us represent the WCET of the parallel application by an integer variable $wcet_{pro}$, the objective function is described as:

$$\text{minimize} \quad wcet_{pro}. \tag{4}$$

The WCET of the parallel application has to be larger than or equal to the latest finish time of any of its tasks. If the latest finish time of task $t_i$ is represented by integer variable $lft_i$, the following constraint is introduced:

$$wcet_{pro} \geq lft_i, \forall t_i \in T. \tag{5}$$

In the following, we present the ILP constraints for computing the latest finish time of tasks and for computing the WCET of tasks by considering the effect of local caches.

**Constraints on tasks' latest finish times.**

The latest finish time of $t_i$ ($lft_i$) is the sum of its latest start time (denoted as $lst_i$) and its worst case execution time (denoted as $wcet_i$):

$$lft_i = lst_i + wcet_i. \tag{6}$$

In equation (6), $wcet_i$ is a variable introduced to integrate the variations of tasks' WCETs due to the effect of local caches (as explained later). The latest start time of $t_i$ ($lst_i$) is the sum of its latest ready time (denoted as $lrt_i$ which is calculated in considering its running order) as well as the worst communication delay with its predecessors (denoted as $wc_i$):

$$lst_i = lrt_i + wc_i. \tag{7}$$

In equation (7), the worst communication delay of $t_i$ with its predecessors ($wc_i$) is computed by considering the predecessors' allocations, i.e.,whether they are allocated on the same core or different cores (as explained later). The latest ready time of $t_i$ ($lrt_i$) is calculated by considering two cases: (1) $t_i$ is the first task running on a core; (2) $t_i$ runs right after another task on the same core.

In the first case, if $t_i$ has some predecessor, its latest ready time has to be equal to or larger than the latest finish time of its predecessors since $t_i$ cannot be executed before the completion of its predecessors. Otherwise, its latest ready time is greater than or equal to zero. Let's denote the set of predecessors of $t_i$ as $pred(t_i)$. The latest ready time of $t_i$ is expressed as:

$$\begin{aligned} lrt_i &\geq 0 \\ lrt_i &\geq lft_j, \forall t_j \in pred(t_i). \end{aligned} \tag{8}$$

In the second case, if $t_i$ is assigned to the same core as $t_j$ and runs immediately after $t_j$, then the latest ready time of $t_i$ is larger than or equal to the latest finish time of $t_j$, $lrt_i \geq lft_j$. Therefore, the latest ready time of $t_i$ in the second case is calculated according to the following constraint:

$$lrt_i \geq o_{j\to i} * lft_j. \tag{9}$$

Since (9) is a quadratic form, we linearize (9) as:

$$lrt_i \geq lft_j + (o_{j\to i} - 1) * M, \tag{10}$$

with $M$ the sum of all tasks' WCETs when running alone plus all communication costs between pairs of tasks when running on different cores, such that $M$ is guaranteed to be higher than the latest finish time of any tasks.

Let us denote the communication cost between $t_i$ and $t_j$ when they are placed on the same core and different cores as $C^s_{j\to i}$ and $C^d_{j\to i}$, respectively. The worst communication delay of $t_i$ ($wc_i$) with its predecessors is calculated as:

$$wc_i \geq s_{i,j} * C^s_{j\to i} + (1 - s_{i,j}) * C^d_{j\to i}, \forall j \in pred(t_i). \tag{11}$$

In equation (11), $s_{i,j}$ is a 0-1 variable to indicate whether two tasks $t_i$ and $t_j$ are assigned to the same core or not.

**Constraints on tasks' WCETs.**

To account for the variability of tasks' WCETs due to private caches, two cases have to be considered when calculating the WCET of a task $t_i$ (variable $wcet_i$): (1) $t_i$ is the first task running on a core; (2) $t_i$ runs right after another task. Let's denote by $WCET_i$ the WCET of $t_i$ when running alone, and $WCET_{j\to i}$ as the WCET of $t_i$ when running right after $t_j$ on the same core. In the first case, the WCET of $t_i$ is equal to its WCET when running alone, $wcet_i = WCET_i$. In the second case, the WCET of $t_i$ is equal to its WCET when running right after another task, $wcet_i = WCET_{j\to i}$. The WCET of $t_i$ is calculated as:

$$wcet_i = \sum_{j\in T-\{i\}} o_{j\to i} * WCET_{j\to i} + \sum_{k\in K} f^k_i * WCET_i. \tag{12}$$

## 4. EXPERIMENTAL RESULTS

Our scheduling approach was evaluated on synthetic task graphs of isolated parallel applications. The communication cost between two tasks $t_i$ and $t_j$ when running on the same core ($C^s_{j\to i}$) and different cores ($C^d_{j\to i}$) is generated randomly with the constraint $C^s_{j\to i} < C^d_{j\to i}$. The WCET of task $t_i$ when running right after task $t_j$ is calculated according to the following equation:

$$WCET_{j\to i} = WCET_i - r_{i,j} * WCET_i. \tag{13}$$

In equation (13), in order to address the effect of local caches on tasks' WCETs, $r_{i,j}$ ($0 \leq r_{i,j} < 1$) is randomly chosen according to the relation between $t_i$ and $t_j$; the range of $r_{i,j}$ in case $t_j$ is direct predecessor of $t_i$ is higher than that in case $t_j$ is indirect predecessor of $t_i$ and that in case $t_j$ and $t_i$ are independent.

In order to evaluate the performance of the proposed scheduler, we compare the WCET values obtained by our proposed scheduling method (S_CACHE), a random scheduling method (S_RAND) and scheduling method without taking into account the effect of private caches (S_NOCACHE). The smaller the WCET, the better the scheduling method. For S_RAND, we first randomly allocate tasks to cores, then tasks scheduling on each core is calculated such that communication/synchronization constraints are respected. We generate 10 schedules using S_RAND and report the best, average and the worst of the estimated WCETs. For S_NOCACHE, we apply the proposed ILP formulas for getting the schedule, but the WCET of a task when running right after another task on the same core is considered to be equal to its WCET when running alone ($WCET_{j\to i} = WCET_i$); when estimating the WCET of the entire parallel application, tasks' WCETs are re-evaluated by considering the effect of private caches. We use CPLEX version 12.5 as ILP solver.

For space considerations, we provide results for two examples of task graphs only (see Fig. 3). In the example of fork-join graph (illustrated in Fig.3), the communication cost between two tasks $t_1$ and $t_3$ when running on the same core is 247785 cycles, and when running on different cores is 376633 cycles. In our example, the range of $r_{i,j}$ in case $t_j$ is direct predecessor of $t_i$ is set to [0.6;0.9], the range of $r_{i,j}$ in case $t_j$ is indirect predecessor of $t_i$ is set to [0.2;0.5], and the range of $r_{i,j}$ in case $t_i$ and $t_j$ are independent is set to [0;0.1].

Fig. 4(b) gives the static schedule obtained by our method for our example of fork-join graph on a two-core architecture equipped with a private cache per core. In the schedule, up arrows denote ready time of tasks ($lrt_i$), while down arrows denote the finish time of tasks ($lft_i$). Colored boxes represent communications (in this specific case, there is no overlap between communications and computations, but overlaps may happen in the general case).

Fig. 4(a) compares estimated WCETs obtained when using the different scheduling methods for our example of arbitrary graph and fork-join graph. We normalize all results with respect to the WCET value obtained by S_CACHE. Our scheduling method generates schedules that lead to the smallest estimated WCET. Moreover, the WCET obtained

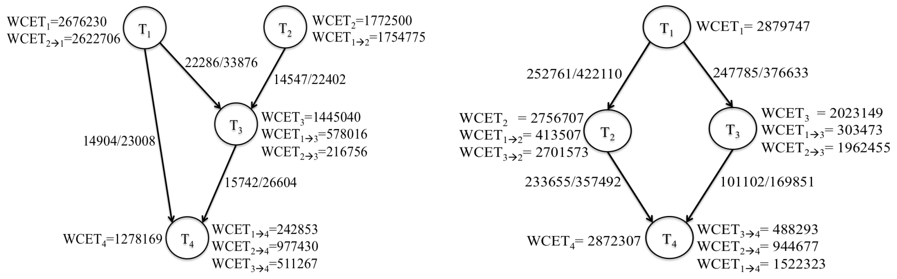
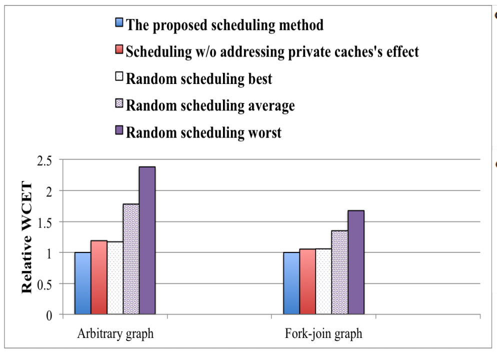
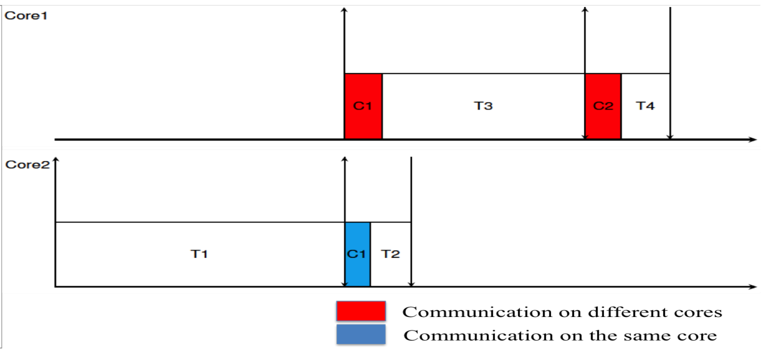


Figure 3: Our example of arbitrary graph and fork-join graph.



(a)


Core1
C1
T3
C2
T4
Core2
T1
C1
T2
Communication on different cores
Communication on the same core

(b)

Figure 4: (a) WCET comparison between different scheduling methods and (b) scheduling graph of tested fork-join graph.

by S_CACHE is less than 43% in our example of arbitrary graph and 26% in our example of fork-join graph when compared with the average results of S_RAND. The sizes of the test graphs is small, leading to a small solution space, explaining why S_RAND finds a schedule as good as S_CACHE (i.e., in our example of fork-join graph, the best WCET obtained by S_RAND is more than 5% when compared to the WCET obtained by S_CACHE). Additionally, compared to S_NOCACHE, we achieve 16% reduction in WCET in our example of arbitrary graph and 5% reduction in WCET in our example of fork-join graph, which shows the interest of considering the effect of private caches on tasks' WCETs in task scheduling. Furthermore, the runtime of our scheduling approach for these two graphs is very small (10 milliseconds on a 3GHz Intel Core i7 CPU with 16GB of RAM).

## 5. CONCLUSION

In this paper, we have developed an ILP formulation for finding an optimal schedule for a parallel application on a many-core architecture. Experimental results show the advantage of the proposed scheduler when considering the effect of private caches on tasks's WCETs. In the future, we will investigate the scalability of the proposed scheduling strategy by applying it to synthetic graphs with larger size, as well as to real applications. Additionally, we will address the effect of shared resource interferences (i.e., shared bus) in task scheduling.

## 6. ACKNOWLEDGMENTS

This work was supported by PIA project CAPACITES (Calcul Parallèlle pour Applications Critiques en Temps et Sûreté), reference P3425-146781). The authors would like to thank Benjamin Rouxel for comments on earlier versions of this paper.

## 7. REFERENCES

[1] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys*, 2011.

[2] H. Ding, Y. Liang, and T. Mitra. Shared cache aware task mapping for wcrt minimization. *Asia and south pacific - Design automation conference (ASP-DAC)*, 2013.

[3] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla. Contention in multicore hardware shared resources: Understanding of the state of the art. *International workshop on worst-case execution time analysis (WCET)*, 2014.

[4] H. Ozaktas, C. Rochange, and P. Sainrat. Minimizing the cost of synchronisations in the wcet of real-time parallel programs. *International workshop on software and compiler for embedded systems (SCOPES)*, 2014.

[5] D. Potop-Butucaru and I. Puaut. Integrated worst-case execution time estimation of multicore applications. *International workshop on worst-case execution time analysis (WCET)*, 2013.

[6] R. Whilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution time problem: overview of methods and surveys of tools. *ACM transactions on embedded computing systems*, 2008.

# A Comparative Study of the Precision of Stack Cache Occupancy Analyses

Amine Naji
U2IS
ENSTA ParisTech
Université Paris-Saclay
amine.naji@ensta-paristech.fr

Florian Brandner
LTCI, CNRS
Telecom ParisTech
Université Paris-Saclay
florian.brandner@telecom-paristech.fr

## ABSTRACT

Utilizing a stack cache in a real-time system can aid predictability by avoiding interference between accesses to regular data and stack data. While loads and stores are guaranteed cache hits, explicit operations are required to manage the stack cache. The (timing) behavior of these operations depends on the cache occupancy, which has to be bounded during timing analysis. The precision of the computed occupancy bounds naturally impacts the precision of the timing analysis. In this work, we compare the precision of stack cache occupancy bounds computed by two different approaches: (1) classical inter-procedural data-flow analysis and (2) a specialized stack cache analysis (SCA). Our evaluation, using MiBench benchmarks, shows that the SCA technique usually provides more precise occupancy bounds.

## Categories and Subject Descriptors

F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program analysis*; C.3 [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*

## Keywords

Program Analysis, Stack Cache, Real-Time Systems

## 1. INTRODUCTION

To meet the timing constraints in systems with hard deadlines, the worst-case execution time (WCET) of software needs to be bounded. Many features of modern processor architectures, such as caches, improve the average performance, but have an adverse effect on WCET analysis. Time-predictable computer architectures [7] thus propose alternative designs that are easier to analyze, particularly focusing on the memory hierarchy [5, 6]. One such design is the stack cache [1, 8], i.e., a cache for stack data complementing a regular data cache. This promises improved analysis precision, since unknown access addresses can no longer interfere with stack accesses (and vice versa). Secondly, the stack cache design is simple and thus easy to analyze [4].

The cache can be implemented using a circular buffer using two pointers: the memory top pointer `MT` and the stack top pointer `ST`. The `ST` points to the top element of the stack and data between `ST` and `MT` is present only in the cache. The remaining data above[1] `MT` is available only in main memory. In contrast to traditional caches, memory accesses are guaranteed hits and the compiler (programmer) is responsible to enforce that all stack data is present in the cache when needed using three stack control instructions: reserve (`sres`), free (`sfree`), and ensure (`sens`). The worst-case (timing) behavior of these instructions only depends on the worst-case spilling and filling of `sres` and `sens` respectively, which can be bounded by computing the maximum and minimum cache occupancy [4], i.e., the value of $\texttt{MT} - \texttt{ST}$.

Stack cache occupancy bounds, and the associated spill/fill costs can be computed using the recently proposed Stack Cache Analysis (SCA) [4]. The approach splits the analysis problem into several smaller steps, using context-insensitive data-flow analyses to capture function-local properties and longest/shortest path searches on the call graph to model calling contexts. An alternative solution would be to simply model the problem as a traditional inter-procedural data-flow analysis (iDFA) [2]. This appears simpler to implement, as the various steps of SCA are modeled in a single concise analysis. However, the impact on analysis precision has not been investigated so far. Indeed, overestimating the occupancy can increase the spill costs associated with `sres` instructions, while underestimating the occupancy can increase the fill costs of `sens` instructions. This work thus compares the precision of the two analysis approaches with respect to the attained max./min. occupancy bounds.

The paper is structured as follows: Section 2 provides some background related to the stack cache as well as static program analysis. We then present the two approaches to analyze the occupancy bounds for the stack cache. The analyses are evaluated in Section 4 before concluding.

## 2. BACKGROUND

The stack cache is implemented as a ring buffer with two pointers [1]: *stack top* (`ST`) and *memory top* (`MT`). The top of the stack is represented by `ST`, which points to the address of all stack data either stored in the cache or in main memory. `MT` points to the top element that is stored only in main memory. The stack grows towards lower addresses.

The difference $\texttt{MT} - \texttt{ST}$ (*occupancy*) represents the amount of occupied space in the stack cache, which cannot exceed the size of the cache's memory $|SC|$, thus $0 \leq \texttt{MT} - \texttt{ST} \leq |SC|$.

[1]We assume that the stack grows towards lower addresses.

The stack control instructions manipulate the two stack pointers and initiate memory transfers to/from the cache to main memory, while preserving the equation from above. A brief summary is given below, details are available in [1]:

| | |
|---|---|
| `sres` $k$: | Subtract $k \leq \|SC\|$ from `ST`. If the cache size is exceeded, a memory *spill* is initiated to decrement `MT` until $\texttt{MT} - \texttt{ST} \leq \|SC\|$. |
| `sfree` $k$: | Add $k \leq \|SC\|$ to `ST`. If this would result in $\texttt{MT} < \texttt{ST}$, `MT` is set to `ST`. Memory is not accessed. |
| `sens` $k$: | Ensure that the occupancy is larger than $k \leq \|SC\|$. If this is not the case, a memory *fill* is initiated to increment `MT` until $\texttt{MT} - \texttt{ST} \geq k$. |

The compiler manages the stack frames of functions quite similar to other architectures with exception of the ensure instructions. For brevity, we assume a simplified placement of these instructions. Stack frames are allocated upon entering a function (`sres`) and freed immediately before returning (`sfree`). A function's stack frame might be (partially) evicted from the cache during calls. Ensure instructions (`sens`) are thus placed immediately after each call. We also restrict functions to only access their own stack frames.[2]

## 2.1 Data-Flow Analysis

Data-flow analysis (DFA) is used to gather information about a program without executing it. A DFA is a tuple $A = (\mathcal{D}, T, \sqcap)$, where $\mathcal{D}$ is an abstract domain (e.g., values of stack pointers), transfer functions $T_i : \mathcal{D} \to \mathcal{D}$ in $T$ model the impact of individual instructions $i$ on the domain, and $\sqcap : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ is a join operator. Together with a CFG an instance of an (*intra-procedural*) DFA can be formed, yielding a set of data-flow equations. The join operator ($\sqcap$) and transfer function ($T$) are instantiated to form $\text{IN}(i)$ and $\text{OUT}(i)$ functions, which are associated with an instruction $i$ and represent values over $\mathcal{D}$. The resulting (recursive) equations are finally solved by iteratively evaluating these functions until a fixed-point is reached [2].

*Inter-procedural* analyses additionally consider the call relations between functions. In this case, additional data-flow equations are constructed modeling function calls and returns [2]. Often these analyses are *context-sensitive*, i.e., the analyses distinguish between (bounded) chains of functions calls. Such a chain of nested function calls is then called a *call string*, which defines a calling context that can be distinguished from other parts of the program calling the same function. Call strings typically have a length limit. The longer the call strings, the higher the ability to distinguish different contexts. Consequently, the analysis results are more precise. Increasing the call string length may also increase the computation complexity and the required memory footprint since additional data-flow equations are created for each context. A call string length of zero corresponds to a context insensitive data-flow analysis.

## 3. CACHE OCCUPANCY ANALYSES

We present how to compute the cache's occupancy, which ca be used to bound timing, using an inter-procedural data-flow analysis (iDFA) and a tailored stack cache analysis.

## 3.1 Inter-procedural Data-flow Analysis

The domain of the iDFA approach are positive integer values in $\mathcal{D} = \{0, \ldots, |SC|\}$, where $|SC|$ represents the stack cache's size. Since both, the min. and the max. occupancy are needed, two analysis problems have to be defined. We will start with the max. occupancy. The analysis starts at the program entry, where the occupancy is assumed to be 0. It then propagates occupancy values along all execution paths, while considering the effect of the instructions along the path. Only the stack control instructions (see Section 2) can have an impact: (1) `sres` instructions increase occupancy by their argument $k$, (2) `sens` instructions make sure that the occupancy is larger than $k$, and (3) `sfree` instructions reduce the occupancy by $k$. The resulting data-flow equation for an instruction $i$ are given below:

$$\text{OUT}_{Occ}(i) = \begin{cases} \min(\text{IN}_{Occ}(i) + k, |SC|) & \text{if } i = \texttt{sres } k \\ \max(\text{IN}_{Occ}(i), k) & \text{if } i = \texttt{sens } k \\ \max(0, \text{IN}_{Occ}(i) - k) & \text{if } i = \texttt{sfree } k \\ \text{IN}_{Occ}(i) & \text{otherwise} \end{cases}$$

The occupancy right before an instruction (due to control-flow joins) is derived by taking the maximum occupancy from any of its predecessors ($Preds$), except for the program's entry. In the case of inter-procedurual analysis, predecessors can also be calls or returns from other functions:

$$\text{IN}_{Occ}(i) = \begin{cases} 0 & \text{if } i = \texttt{entry}, \\ \max_{s \in Preds(i)}(\text{OUT}_{Occ}(s)) & \text{otherwise} \end{cases}$$

The data-flow equations to compute the min. occupancy are very similar. Only the max operator of the $\text{IN}_{Occ}(i)$ equation needs to be replaced by the min operator. Context sensitivity can easily be ensured by adding context information to the data-flow equations of the respective instructions.

This model is also implemented and validated in Absint's aiT timing analyzer tool [10].

## 3.2 Stack Cache Analysis

The stack cache analysis (SCA) [4] relies on similar DFA analyses. However, instead of a single, large inter-procedural DFA, several smaller function-local analyses are used. The impact of other functions at function calls in these DFAs are modeled through minimum and maximum *displacement* values, which represent the min./max. amount of data potentially evicted from the the stack cache during a function call. Displacement values are computed by performing shortest/longest path search on program's call graph whose weights represent the reserved stack space $k$. Complex context-sensitive analysis thus can be avoided.

The analysis is based on the observation that the occupancy at any instruction within a function can be computed from the occupancy at the function's entry and the displacement of all the potential function calls on any path leading to the particular instruction. The min. occupancy thus can be computed by considering the initial min. occupancy and the max. displacement. Likewise, the min. displacement allows to derive the max. occupancy.

The program is thus analyzed in several steps. First, the min. and max. displacement of each function is computed using longest/shortest path searches on a weighted call graph. Next, local DFAs are performed to compute local lower and local upper bounds on the min. and max. occupancy within each function assuming a stack cache that is full at function entry. Finally, the concrete occupancy bounds are computed for each function considering the occupancy bounds at its respective callers and the previously

[2]Data that is larger than the stack cache or that is shared can be allocated on a *shadow stack* outside the stack cache.

computed local occupancy bounds. The final phase can deliver fully context-sensitive information, if so desired.

This analysis was implemented and validated against runtime measurements in previous work [8].

## 4. EXPERIMENTS

We evaluated both approaches using the LLVM-based compiler framework of the Patmos processor [9], which comes with a stack cache and its associated control instructions. Benchmarks of the MiBench benchmark suite [3] were compiled using optimizations (`-O2`) and subsequently analyzed using both techniques, assuming a stack cache size of 256 byte, 4 byte cache blocks, and a contexts string length of 0. Figure 1 shows the percentage of functions where the occupancy bound at function entry of SCA was either greater, equal, or smaller than that computed by iDFA.

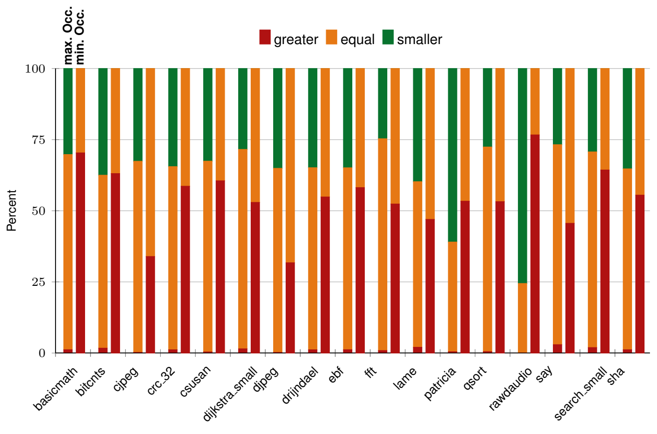


Figure 1: Percentage of occupancy bounds (max./min) by SCA being (1) greater, (2) equal, or (3) smaller than iDFA.

When considering max. occupancy, SCA is less precise when the delivered bound is greater, i.e., the lower portion of the first bar of each benchmark should be as small as possible. Indeed, these cases are rare (< 3% over all benchmarks), while SCA is often more precise (34% on average).

The situation is inverse when considering min. occupancy. Here, SCA is less precise when the delivered bounds is smaller. This would be represented by the upper portion of the second bar. However, this appears for one function of the three benchmarks `tiff2bw`, `tiffdither`, and `tiffmedian` respectively. SCA is usually even more precise (52% on average).

We repeated these experiments for iDFA with other call string lengths (1, 2, 3, 10 and 20). However, we only observed minor improvements for max. occupancy and almost no change for min. occupancy. The `bitcnts` benchmark, for instance, has a maximum call depth of 20, ignoring recursive functions, and still does not show relevant improvements with call strings of length 20 due to the impact of recursion elsewhere as explained later.

Overall, SCA is almost always as precise or even more precise than iDFA. The results are similar, albeit less pronounced, with longer context string lengths.

### 4.1 Discussion

A closer look reveals that the imprecision of iDFA is mostly due to chains of function calls, whose lengths exceed the analysis' context string length (e.g., due to recursion). Let us first examine such situations for max. occupancy.

The problem of iDFA with long call chains is that calling contexts are no longer distinguished, i.e., all information is merged in a single calling context. The occupancy information computed for these regions is, as expected, rather pessimistic, leading to considerable overestimation of the max. occupancy. Even worse, the overly conservative occupancy level is propagated out of these merged calling contexts along control-flow edges of function returns. Recall that the meet operator for this analysis is the max operator. This means that the conservative max. occupancy bounds are even further propagated, way beyond the merged calling contexts that initially caused the imprecision. This particularly applies to recursive functions.

*Example* 1. Figure 2 shows an example illustrating this situation. Assume that function `A` consists of one basic block and that function `B` is called before function `D`. Since `B` and `C` recursively call each other, their respective max. occupancy grows until they reach the stack cache size during the fixed-point computation of iDFA (unless unbounded call strings are used). The transfer functions for the return instructions then propagate the maximum to their respective callers, which leads to a max. occupancy that is close to the stack cache size right after the function call to `C` within `B` (and vice verse). A similarly high occupancy is propagated out of the recursion to the instruction succeeding the call from `A` to `B`. The high occupancy might actually occur within the recursion. However, the actual occupancy at this point is much lower. The overestimation is further propagated to function `D`. Resulting in overly conservative analysis results there, even when the context string length is not exceeded.

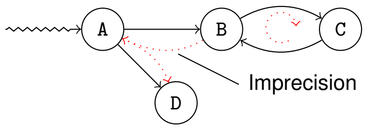


Figure 2: Imprecision propagated out of recursive functions when computing max. occupancy with iDFA.

Patmos' newlib C library contains (potentially) recursive functions in the start-up code of each program. iDFA thus assumes that the stack cache is filled up entirely before even reaching the program's `main` function. Since the computed max. occupancy at `main` is considerably overestimated, imprecision is propagated throughout large portions of the considered benchmarks. An important observation here is that increasing the call string length will not help fixing this problem, as the precision limit will be reached before the end of the recursion (unless infinite call strings are used). The SCA approach does not face this problem. Instead of relying on the occupancy propagated outwards by the recursive functions, it simply relies on their displacement values. A possible fix for this problem for iDFA would be to memorize the occupancy level before each call. The occupancy propagated backwards from a return then always has to be smaller than the memorized value. However, the potentially large displacement of the called function is ignored, which may still lead to considerable overestimation.

A similar problem arises for non-recursive programs with deep call chains containing two subsequent function calls that eventually invoke the same function. iDFA then behaves similar to recursive programs as shown in Figure 3.

*Example* 2. Assume that, in this example, the function call from `B` to `C` appears before the call from `B` to `D`. Then, iDFA initially propagates an accurate occupancy level through the calls from A to B and finally to C. At first, even the occupancy at D is computed correctly. However, due to the deep call chain leading up to C, both calling contexts for D (originating from B or D) are merged. Due to the intermittent execution of D the occupancy is higher for this call chain. This increases the max. occupancy of C. The increase is subsequently propagated out of C to both of its callers. This incidentally increases the occupancy after the call to C within B. Which then again increases the occupancy at the following call to D. This leads to a feedback loop similar to that seen for recursive functions in the previous example.

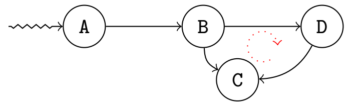


Figure 3: Feedback loop enforcing imprecision of non-recursive functions for iDFA computing max. occupancy.

Still, iDFA can be more precise than SCA (as shown by or results). This is explained by an underestimation of the min. displacement. As mentioned before, the min. displacement is obtained by performing a shortest path search on the program's call graph. The path here represents nested function calls and its length the minimal amount of stack space required in the stack cache by the functions stack frames respectively. Now, consider a case where two leaf functions[3] are called within a single basic block, i.e., when one function is called the other function is called too. In this case, the minimal path search will chose the function with the smaller stack frame to compute the min. displacement. However, since both functions are called, the actual min. displacement is determined by the larger stack frame. This situation can, of course, also appear in more general forms. The imprecise min. displacement ultimately leads to an underestimation of the max. occupancy observed in our experiments. However, this appears to be of minor importance in practice.

For min. occupancy iDFA appears to be even more imprecise. For one, this is explained by the fact that the max. displacement (in contrast to the min. displacement) can be computed precisely. SCA's min. occupancy thus does not suffer from inherent imprecision. In addition, iDFA spreads imprecision as before in the presence of deep call chains. This may even lead to feedback loops in non-recursive programs as described before. Two observations are particularly interesting at this point. While the iDFA approach is amenable to improvements by memorizing the max. occupancy before calls, such a fix appears to be impossible here. The problem is that a lower bound cannot be established as easily for function calls when the min. occupancy is computed. Secondly, it appears that the precision could be improved using very long call strings (ignoring cases incurring recursion). This, however, leads to a paradox situation: the precise computation of the min. occupancy would then require high levels of context sensitivity in order to compute the worst-case filling at sens instructions. The filling, however, only depends on the nesting of functions called right before the ensure and thus is by its nature context-insensitive. The SCA approach exploits precisely this property, and evidently achieves excellent results.

[3]Leaf functions do not call any other function.

## 5. CONCLUSION

We compared the precision of stack cache occupancy bounds computed by two different approaches. On the one hand, the iDFA approach, which models the problem as a traditional inter-procedural data-flow analysis. On the other hand, the SCA approach that splits the analysis problem into several smaller steps, using context-insensitive data-flow analyses along with longest/shortest path searches on the call graph. Our experiments revealed that iDFA suffers from imprecision in nearly all benchmarks of the MiBench benchmark suite. The lack of precision is due to chains of function calls, whose lengths exceed the analysis' context string length (e.g., due to recursion). As a future work, we plan to compare the efficiency (i.e computation complexity and required memory footprint) of iDFA and SCA.

## Acknowledgments

This work was supported by a grant (2014-0741D) from Digiteo France: "Profiling Metrics and Techniques for the Optimization of Real-Time Programs" (PM-TOP).

## 6. REFERENCES

[1] S. Abbaspour, F. Brandner, and M. Schoeberl. A time-predictable stack cache. In *Proc. of the Workshop on Software Technologies for Embedded and Ubiquitous Systems*. 2013.

[2] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2nd edition, 2006.

[3] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proc. of the Workshop on Workload Characterization*, WWC '01, 2001.

[4] A. Jordan, F. Brandner, and M. Schoeberl. Static analysis of worst-case stack cache behavior. In *Proc. of the Conf. on Real-Time Networks and Systems*, pages 55–64. ACM, 2013.

[5] S. Metzlaff, I. Guliashvili, S. Uhrig, and T. Ungerer. A dynamic instruction scratchpad memory for embedded processors managed by hardware. In *Proc. of the Architecture of Computing Systems Conference*, pages 122–134. Springer, 2011.

[6] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee. PRET DRAM controller: Bank privatization for predictability and temporal isolation. In *Proc. of the Conference on Hardware/Software Codesign and System Synthesis*, pages 99–108. ACM, 2011.

[7] C. Rochange, S. Uhrig, and P. Sainrat. *Time-Predictable Architectures*. ISTE Wiley, 2014.

[8] S.Abbaspour, A. Jordan, and F. Brandner. Lazy spilling for a time-predictable stack cache: Implementation and analysis. In *Proc. of the International Workshop on Worst-Case Execution Time Analysis*, pages 83–92. OASICS, 2014.

[9] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. Probst, S. Karlsson, and T. Thorn. *Towards a Time-predictable Dual-Issue Microprocessor: The Patmos Approach*, volume 18, pages 11–21. OASICS, 2011.

[10] T-CREST. Report on architecture evaluation and WCET analysis. Technical report, 2013.

# ASLA: Adaptive System Level in AUTOSAR

Amel BELAGGOUN
CEA LIST-Paris
LISE Labs,Point courrier 174
Gif-sur-Yvette,91191 France
amel.belaggoun@cea.fr

Ansgar RADERMACHER
CEA LIST-Paris
LISE Labs,Point courrier 174
Gif-sur-Yvette,91191 France
ansgar.radermacher@cea.fr

Valerie ISSARNY
Inria Paris-Rocquencourt
78153 Le Chesnay,France
valerie.issarny@inria.fr

## ABSTRACT

This paper presents an initial approach towards making AUTOSAR dynamic starting from the application layer down to the operating system level (task model and RTE)(i.e. extending AUTOSAR ECU Software architecture) and describes ASLA, which is the framework that provides tasks-level adaptation techniques in AUTOSAR.

## 1. INTRODUCTION AND MOTIVATION

Nowadays, the complexity of the next generation of automotive embedded systems such as Fully Electric Vehicles (FEVs) is increasing due to the growing number of functionality (more than 2.500 functions such as power trains, steering or braking systems, “X-by-wire” systems etc.). These systems are by nature, real-time. Moreover, most of them work under several resource constraints, due to cost, space and energy limitations. In addition, they are running in highly dynamic environments. Combining real-time features in tasks with dynamic behavior, together with cost and resource constraints may create new problems to be addressed in the design of such systems. Using the classical design approaches adopted in hard real-time systems, such as WCET analysis to guarantee timeliness, for example, is no longer acceptable in highly dynamic environments because it would waste resources and increase costs [5]. Instead of allocating resources for the worst case, there is a need for smarter techniques to sense the current state of environment and react accordingly, which means, to cope with dynamic environments, automotive systems need to be *adaptive*; that is, they must be capable of changing their structure and/or their behavior to better reflect their current situation or in order to keep the system requirements at a desired level; if this is not possible, degrade it in a controlled way. Besides dealing with problems generated by adaptation in terms of meeting real-time constraints despite the system evolution; system reliability remains an important factor to be investigated within embedded systems due to their interactions with the physical word. Although, there has been much research on building reliable distributed real-time systems [3][7], a trend towards more complex features in a system with cost and resource constraints poses a major challenges in developing such a system. In order to address the above challenges we need to study the feasibility of applying adaptation techniques in real-time embedded systems. Specifically, in automotive systems it would be very useful to support adaptation in order to increase the availability and reliability of software-based applications without additional hardware costs. Currently AUTOSAR[1]– The standard architecture for automotive systems– has no support for runtime adaptation. Making it adaptive requires specific support at different levels of the software architecture. The most important component affecting adaptivity is the Operating System (OS), but some flexibility can also be introduced in the runtime environment(RTE). Therefore, we propose a layer called ASLA, which is used for task mapping, bandwidth allocation and adaptation for mixed-criticality distributed systems. The novelty of our approach is the capability to support the adaptation of applications with soft and hard real-time requirements (mixed) while respecting timing and safety requirements in AUTOSAR.

**Overcoming AUTOSAR’s limitations with ASLA:** ASLA provides the ability to dynamically adapt the system, such as adding a new application or moving an existing application to a different ECU. In AUTOSAR, the system configuration is by design static: the AUTOSAR RTE is configured at design-time for specific ECUs and partly generated based on the requirements of the Software Components(SWC). A reconfiguration of the system, such as adding an application or moving an application from one ECU to another, cannot be done dynamically at runtime. ASLA extends AUTOSAR in two ways. It changes the scheduling policy from a fixed-priority (assigned to tasks at design time) to a dynamic preemptive policy based on EDF and CBS schedulers to guarantee mixed critical requirements. It also contains RTE extension that supports the runtime application migration between ECUs in response to both anticipated changes caused by the environment, such as network connectivity, as well as unexpected failures in both software and hardware. The deployment and re-configuration of an application onto ASLA is based on the work of [15], which uses a Tabu Search based meta-heuristic to search design space exploration to give the best task allocations and bandwidth allocation. Unlike the existing approaches [10, 18], ASLA explicitly introduces the concept of runtime adaptation in mixed-criticality applications in automotive systems.

**Contribution.** This paper presents the design of a real-time adaptive distributed architecture, ASLA, to provide task-level adaptation techniques in AUTOSAR.

The remainder of this paper is organized as follows. Section 2 summarizes the background, the scope and requirements needed to understand the proposed approach. Section 3 presents our approach and the architecture of ASLA. Section 4 presents the related work. We state our conclusion and future work in Section 5.

## 2. BACKGROUND AND DEFINITIONS

Before delving into the approach, it is important to clarify the scope and the requirements that need to be addressed with our solution. We use Kiviat diagrams [16] to show visually the characteristics of our solution (Fig.1) and the requirements we consider for adaptive automotive systems (Fig.2). These diagrams provide developers of automotive software an easy way of viewing the characteristics of their applications. The dimensions represent axes of the Kiviat diagrams and characteristics of the dimensions represent the set of properties to be met by our solution (i.e. the red bullets).

## 2.1 The scope of our Research

**The adaptation Model.** The adaptation may typically be *synchronous* and/or *asynchronous* with respect to the execution of applications. In the *synchronous* case, the applications synchronize their execution, and new tasks (actions) are introduced only after all applications have finished performing the actions specified in the initial configuration. The schedulability analysis of the adaptation is thus not required, because there is no interference between tasks before and after the adaptation (i.e. in the new system configuration). On the other hand, in the case of *asynchronous* adaptation, all the applications start changing their configuration as soon as they receive an adaptation trigger without considering the behavior of the other applications. As a result, actions of the initial system configuration run concurrently with the new ones during the transition, which calls for schedulability analysis.

**Promptness.** Adaptation is well suited for systems that require reactive behavior, there is no need to wait until an idle period or slack before performing such a change as in Tindell's model [17] or at the end of cycles like in some approaches based on cyclic executive scheduling [14]. The adaptation may be classified in three different categories in the time domain: (i) *time-based adaptation:* These are adaptations where we know the arrival time of the adaptation request in advance. (ii) *event-driven adaptation:* These are adaptations triggered by events rather than time. We don't know exactly when they happen.(iii) *irregular event-driven adaptation*: These are adaptations when no prediction can be made about the arrival time of the adaptation request. An example of this type is a system fault. The system may change its configuration and migrate to a degraded one where not all the functionalities are provided. In our work, we consider all three types of adaptation.

**Scheduling policy.** The use of dynamic priorities scheme suits better with systems running in highly dynamic environments [14][9].

**Scheduling algorithm.** Combining two scheduling mechanisms CBS and EDF has proven its efficiency to solve the problem of temporal isolation due to the integration of soft and hard real-time applications on the same platform [9][15].

**Architecture.** Distributed architecture. In our work a real-time distributed system is defined to be a system with multiple autonomous processing units (ECUs) cooperating together to achieve a common goal. We use the term distributed architecture to refer to loosely coupled architectures where message passing is required (full connectivity is assumed between ECUs i.e. each of the ECUs is connected to each other).

**Timing requirement criticality.** In our solution we are dealing with mixed-criticality systems, to the best of our knowledge no one has applied runtime adaptation considering both soft and hard real-time constraints in AUTOSAR.

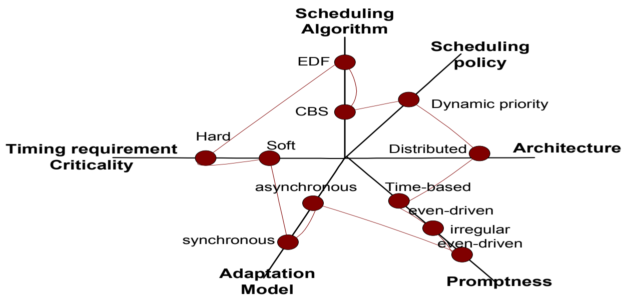


**Figure 1: The scope of our Research.**

## 2.2 Definition of our Adaptation Requirements

**(R1) Timeliness**: the timeliness requirements of an adaptation characterize the time constraints under which the adaptation is executed. Hard real-time constraints require the execution of adaptation within a firm deadline. Adaptations executed under soft real-time constraints minimize the adaptation execution and blackout time (which is the time the application is unavailable due to state transfer and reconfiguration). Unbounded adaptations are executed without any time bound[6].

**(R2) Consistency**: preserving the system consistency and leaving the system under change in a correct state after adaptation are two major requirements that must be ensured when performing adaptation of the running system. Many adaptation approaches freeze the entities to be reconfigured into an adaptation safe state called quiescent state [8].

**(R3) Flexibility**: The dynamic behavior of real-time systems requires executing applications with certain flexibility requirements in which the temporal properties and the number of applications vary during runtime. That means, the tasks of flexible application provide implementations that can adapt their execution to the available processing resources. These tasks have variable period and/or may demand variable WCET (i.e. stochastic [9]). So executing flexible applications, prevents the use of an efficient static temporal partitioning of the processing time. A static temporal partitioning that lasts over the entire lifetime of a system would result in an oversized system. Hence, in order to efficiently use the processing time, the flexibility of applications and possible demand changes during runtime have to be considered during runtime analysis.

**(R4) Adaptation trigger**: adaptation can be triggered either *internally* due to the monitoring infrastructure or *externally*; requested from an outside entity, for example the user. Furthermore , adaptation triggers may arrive synchronously (i.e. at specific time) or asynchronously (i.e. with unknown arrival pattern).

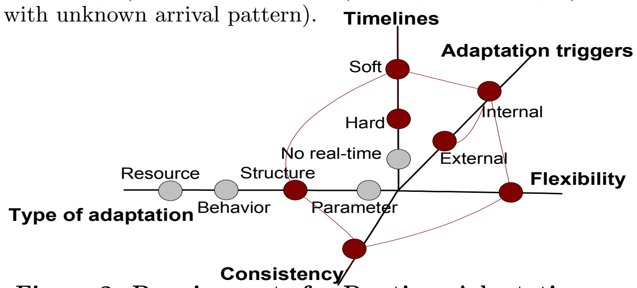


**Figure 2: Requirements for Runtime Adaptation.**

**(R5) Type of adaptation**: the type of adaptation defines what is being reconfigured. This type can be either, *Resource adaptation*(i.e.allocation of resources dynamically based on observed conditions), or *Software adaptation* which includes three categories: (i) structural, (ii) behavioral and (iii) parameter. The first one (i) structural adaptation, changes the actual architectural parts of an application, e.g., by removing a SWC, introducing a new one or replacing /updating an existing SWC with another newer version. The second category (ii) behavioral adaptation allows changes of the behavior of the application and the last category (iii) parameter adaptation involves modifying variable values that determine program behavior.

## 3. TOWARDS ADAPTIVE AUTOSAR

We present an approach that provides an adaptive solution for AUTOSAR[1]. The approach is based on monitoring various applications distributed on different ECUs to detect the need for adaptation at the application and the system-level ( i.e. the Basic SoftWare(BSW)) while maintaining the system consistency [8], which means the system under adaptation must be left in a correct state after adaptation.

### 3.1 The ASLA Architecture

Fig.3 provides an overview of ASLA's overall design. Every ECU that supports adaptation through ASLA layer consists of a real-time OS with EDF and CBS scheduling policies, an Adaptive SWC that is responsible for reconfiguring applications running on the system, RTE, and application layers. The real-time OS is responsible for HW abstraction, communication, scheduling and executing tasks in real-time. We assume that the underlying HW is a fail-silent system and the communication network is fault-tolerant. Our application layer consists of a set of SWCs (similar to AUTOSAR's) and the new Adaptive SWC which can be distributed over several ECUs. The RTE provides a communication abstraction to SWCs. Unlike AUTOSAR, our RTE extension contains functions to support adaptation. These functions are managed by the Adaptive SWC (more precisely by the Reconfiguration Manager (RM)) which also communicates with the others Adaptive SWCs running on the different ECUs to make one of the adaptation actions such as: adding, deleting or updating application. When a new application is being added, the mapping between the application's SWCs and the ECUs is given to the RM, then each RM analyzes the mapping and renews the RTE's functions.

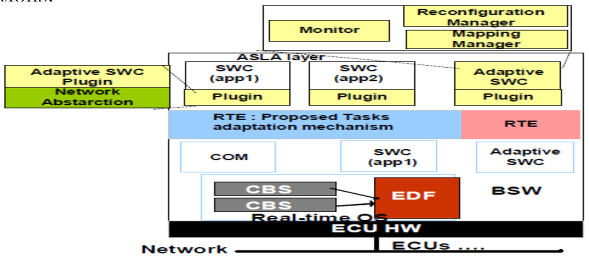


**Figure 3: The ASLA architecture.**

The ASLA layer is composed of an Adaptive SWC (one on each ECU) and plugin offering a task execution container. This plugin enables any task launched on ASLA layer to be periodically executed. The adaptive component has a coordination-based architecture. One Adaptive SWC acts as a coordinator of the other Adaptive SWCs which are responsible for handling tasks on each ECU and monitoring a health vector. The latter contains all Non-Functional Requirements(NFRs) needed for the adaptation such as the ECU's processor utilization, resources, QoS, HW NFRs..etc. All operational ECUs compute their resources and processor utilization in form of a health vector at a fixed time period and share their health vector with each other. This provides each ECU a consistent view of the available resources and utilization on the other nodes. Since our Adaptive SWC has a coordination based-architecture, we define a management protocol between the different Adaptive SWCs inspired by [11]. However, we differ from them in the sense that our protocol is much more simpler and it is used for managing the process of an adaptation in distributed real-time systems. In our protocol, all the Adaptive SWCs including the coordinator broadcast messages to each other. The coordinator can detect the failure of the others Adaptive SWCs by a lack of heartbeat messages. The major components of ASLA are described below.

**A. The Adaptive SWC.** As illustrated in Fig 3, an Adaptive SWC is composed of a monitor,a Mapping Manager (MM) and a Reconfiguration Manager(RM). The monitor is responsible for monitoring events that trigger the adaptation. The MM offers a dynamic deployment of tasks on the ECUs and the RM can automatically reconfigure tasks inside/or between the different ECUs :

- **The Monitor.** The monitor periodically sends messages to other ECUs in the system via the network[1]. The monitor allows ASLA to agree on the availability of each ECU. Any adaptation trigger received by the application during its execution may invoke the monitor which sends a message to the RM in order to adapt the application. The loss of a message for two consecutive cycles means that the ECU is no longer alive and the adaptation needs to be triggered to accommodate the desired changes.

- **The Mapping Manager.** The MM offers an automatic deployment of tasks on ECUs. We use TSMBA (Tabu search Mapping and Bandwidth Allocation)[15] as a base line algorithm for our work to do the allocation which provides a comprehensive solution that allocates mixed critical application to a distributed heterogeneous architecture and reserves processor bandwidth for guaranteeing timing requirements. We propose the extension O-TSMBA (Operational chains-TSMBA) a variant of TSMBA that supports task dependencies (i.e. pipeline tasks model). The MM takes as input the application description (an initial system configuration file) and changes the current mapping when it's necessary to do so. Changes of the allocation can occur due to the adaptation or in case of one or several ECUs failures.

- **The Reconfiguration Manager.** The RM is a sporadic task that gets triggered upon the reception of an adaptation trigger (requests for adding new tasks, requests for migrating failed tasks/and or failed ECUs, replacement of tasks with an improved version and removing tasks).

**B. ASLA Plugins.** All applications will run on the top of ASLA plugins. ASLA plugins support the mechanisms for task reconfiguration and bandwidth allocation (i.e.TSeRBA algorithm "Tabu Search Reconfiguration and Bandwidth Allocation") and also enables tasks to have guaranteed and protected access to required processing resources during reconfiguration in a timely manner.

[1]The Network is beyond the scope of this paper. We assume a synchronous communication network

## 3.2 ASLA Development Process

Our process for developing automotive software in compliance with AUTOSAR standard is shown in Fig.4. It starts with an application description, in terms of a SWC architecture, dependencies between SWCs, real-time constraints, HW resource requirements and other information needed in the vehicle. In our approach we are interested in mixed critical applications with soft and hard real-time requirements. We consider that each SWC contains one runnable and is represented by one AUTOSAR task. We construct an operational chain OP which is composed of periodically executing runnables generating data and events regularly that flow through multiple runnables (i.e. they are connected by data flow or/and control flow). OP correspond to the AUTOSAR execution model (see Fig4-❶). We distinguish chains with soft and hard timing requirements ($OP_{Soft}$ and $OP_{Hard}$). We assume that the initial mapping of runnables to AUTOSAR tasks is given at design time similarly to AUTOSAR and all runnables are executed periodically within the context of an AUTOSAR Task (see Fig4-❷). At this step we will have an initial solution which is not necessary schedulable and it is given as input to the task mapping algorithm O-TSMBA which we designed for the operational chain model. The output of O-TSBMA algorithm will be used as input for TSeRBA (i.e. a configuration schedulable and tagged optimized (see Fig4- ❸). TSeRBA algorithm is used for task mapping, bandwidth allocation and reconfiguration for mixed-criticality distributed systems. The runtime support to AUTOSAR will be realized by TSeRBA as it allows the dynamic allocation of SWCs with both hard and soft real-time constraints as well as supports the insertion, the deletion and the migration of SWCs at runtime (see Fig4-❹).

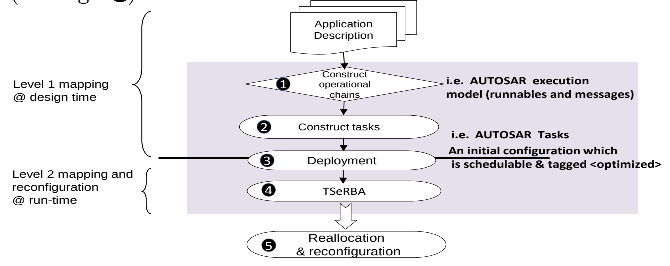

Application Description
Level 1 mapping @ design time
❶ Construct operational chains
i.e. AUTOSAR execution model (runnables and messages)
❷ Construct tasks
i.e. AUTOSAR Tasks
❸ Deployment
An initial configuration which is schedulable & tagged <optimized>
Level 2 mapping and reconfiguration @ run-time
❹ TSeRBA
❺ Reallocation & reconfiguration

**Figure 4: ASLA Development Process**

## 4. RELATED-WORK

Runtime adaptation in real-time systems have been extensively studied in[4, 18, 7]. Unlike existing works, ASLA provides a framework to support runtime adaptation with taking into account schedulability analysis and task allocation for mixed-criticality applications in AUTOSAR. Except SAFER[7] which can be used as a complementary solution for our proposal for the dependability point of view. Another well researched topic in real-time systems–mode change concept, which has a close relationship with the adaptation. In order to guarantee timing requirements in the presence of changes in the system several approaches have been proposed and focused on mode change protocols (see the survey of [14] for more details),[12], which can be potentially used as an extension to ASLA. The authors of [13] proposed SIRAP, a protocol for synchronization in a hierarchical real-time scheduling framework. Their approach is relevant to our case because we are using a server-based technology to schedule AUTOSAR SWCs. However, we have a different focus; we are tackling the challenge of making runtime adaptation in AUTOSAR.

## 5. CONCLUSION AND FUTURE WORKS

Runtime adaptation in embedded real-time systems is a topic expected to grow in the coming years, gaining particular moment in the context of designing FEVs. Yet there is a lack of techniques and tools for performing such adaptation. We presented ASLA, a novel framework that supports task-level reconfiguration features in AUTOSAR. We have built an experimental platform using three ARM-based STM32F4Discovery boards, an Open-source AUTOSAR implementation, ERIKA-OS[2] served as the BSW. A CAN bus communication was established between the three ECUs and currently we are focusing on the implementation of the algorithms behind ASLA framework to demonstrate the theoretical ideas. As future work, ASLA will be validated by means of a real case study from the SafeAdapt project. We will measure the overhead of the adaptation mechanisms, in particular the impact on timeliness.

**Acknowledgments:** We would like to thank Robert Davis for the interesting discussions about this work.

## 6. REFERENCES

[1] Autosar,http://www.autosar.org/.
[2] Erika entreprise,http://erika.tuxfamily.org/drupal/.
[3] J. Balasubramanian, A. Gokhale, A. Dubey, F. Wolf, C. Lu, C. Gill, and D. Schmidt. Middleware for resource-aware deployment and configuration of fault-tolerant real-time systems. In *RTAS*, pages 69–78, Los Alamitos, CA, USA, 2010.
[4] B. e. a. Becker. Model-based extension of autosar for architectural online reconfiguration. MODELS'10, pages 83–97, Berlin, Heidelberg, 2010.
[5] G. Buttazzo and L. Santinelli. Adaptive mechanisms for component-based real-time systems". In *NASA/ESA Conference(ASH 2015), June 15-18, Montreal, QC, Canada.*, pages 1–8, 2015.
[6] S. Fritsch, A. Senart, and D. C. Schmidt. Time-bounded adaptation for automotive system software. ICSE '08, pages 571–580, New York, NY, USA, 2008.
[7] J. Kim, G. Bhatia, R. Rajkumar, and M. Jochim. SAFER: system-level architecture for failure evasion in real-time applications. In *RTSS '12, San Juan, PR, USA, December 4-7, 2012*, pages 227–236, 2012.
[8] J. Kramer and J. Magee. Self-managed systems: An architectural challenge. FOSE '07, pages 259–268, Washington, DC, USA, 2007.
[9] L.Abeni and G. Buttazzo. Stochastic analysis of a reservation based system. In *IPDPS'01, pp.946-952*, 2001.
[10] H. Martorell, J.-C. Fabre, M. Roy, and R. Valentin. Improving adaptiveness of autosar embedded applications. SAC '14, pages 384–390, New York, NY, USA, 2014.
[11] M. Mitzlaff and Kapitza. Enabling mode changes in a distributed automotive system. CARS '10, pages 75–78, New York, NY, USA, 2010.
[12] V. Nelis and B. e. a. Andersson. Global-edf scheduling of multimode real-time systems considering mode independent tasks. ECRTS'11, pages 205–214, Washington, DC, USA, 2011.
[13] T. Nolte, I. Shin, M. Behnam, and M. Sjödin. A synchronization protocol for temporal isolation of software components in vehicular systems. 5(4):375–387, November 2009.
[14] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, 26(2):161–197, Mar. 2004.
[15] P. K. Saraswat, P. Pop, and J. Madsen. Task mapping and bandwidth reservation for mixed hard/soft fault-tolerant embedded systems. RTAS '10, pages 89–98, 2010.
[16] K. W. K. T. software empiricist. *SIGMETRICS Perform. Eval. Rev.*, 2(2), 1973.
[17] K. Tindell, A. Burns, and A. Wellings. Mode changes in priority preemptively scheduled systems. In *RTSS'92, USA*, pages 100–109, 1992.
[18] C. Zeller, Marc; Prehofer. Timing constraints for runtime adaptation in real-time, networked embedded systems. SEAMS '12, pages 73–82, 2012.

# Towards Utilizing Reconfigurable Shared Resources in Multi-Core Hard Real-Time Systems

Luca Pezzarossa
lpez@dtu.dk

Martin Schoeberl
masca@dtu.dk

Jens Sparsø
jspa@dtu.dk

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Kongens Lyngby, Denmark

## ABSTRACT

Dynamic partial reconfiguration (DPR) of FPGAs allows the reconfiguration of selected areas of an FPGA after its initial configuration, while the remaining part of the system continues to operate without interruption. Hard real-time systems are a class of systems whose temporal behavior has to be completely predictable. Our research explores the use of DPR of FPGAs in the context of hard real-time multi-processor systems for embedded applications, targeting the T-CREST multi-core platform. This paper provides an overview and discusses the challenges related to the use of DPR to share reconfigurable resource in a time-predictable manner. It presents an approach to the application of DPR in multi-core hard real-time systems and proposes two models to describe the effect of DPR on the software tasks execution and scheduling. Related works, plans for future evaluation and a preliminary test are also presented.

## 1. INTRODUCTION

Dynamic partial reconfiguration (DPR) is an emerging concept in the FPGA industry that allows the reconfiguration of portions of the FPGA, while the rest of the device continues to operate without interruption [1, 2, 3].

Hard real-time embedded systems are a class of systems characterized by strict timing constraints on the execution time of the tasks. These systems are used for safety-critical applications where a failure to respond in time may lead to catastrophic consequences (e.g., flight electronics, wind turbine control systems, medical devices, factory automation systems, etc.). The design process of such systems, addressing multi-core architectures instead of single processor architectures, is more complicated and challenging due to the fact that the temporal behavior has to be completely predictable and analyzable.

For multi-core hard real-time embedded systems, which are mainly used in professional or high-end applications, the development and the fabrication cost of an ASIC is typically prohibitive, since it is not possible to amortize the development costs over the production volume. Therefore, FPGAs usage is preferable for this class of applications. FPGAs are less efficient than ASICs, but this can be compensated for by shorter development time and increased flexibility. DPR brings this flexibility even further by enabling run-time changes in the hardware.

Our research investigates the usage of FPGAs' DPR in the context of multi-core hard real-time systems. It targets the existing platform T-CREST [4] and aims to supplement it with the DPR feature. T-CREST is a homogeneous multi-core platform for embedded hard real-time applications especially optimized to simplify static worst-case execution time (WCET) analysis. Our hypothesis is that DPR can provide substantial benefits by allowing dynamic hardware modifications. More specifically, we hypothesize that a system which uses a DPR approach can be more efficient in terms of size, power consumption and cost than an equivalent static version, while maintaining comparable computational performance.

This paper provides an overview and discusses the challenges related to the use of DPR to share reconfigurable resources between software tasks in a time-predictable manner. More specifically, it contributes (i) by presenting an approach to the application of DPR in multi-core hard real-time systems and (ii) by proposing two models to describe the effect of DPR on the tasks execution and scheduling. Related works, plans for future evaluation and a preliminary test are also presented.

This paper is organized as follows: Section 2 provides general background about DPR by presenting the state-of-the-art, the potential benefits and the limitations of the current FPGA technology. Section 3 presents a classification of DPR with the relative configuration latencies and proposes two formulated models for DPR feature in hard real-time systems. Section 4 describes our plans for evaluation and a preliminary test utilizing the T-CREST platform. Section 5 briefly presents related works and finally Section 6 concludes the paper.

## 2. DYNAMIC PARTIAL RECONFIGURATION OF FPGAS

Dynamic partial reconfiguration (DPR) allows the modification of an operating FPGA design by loading a partial configuration file (bit-file), while the remaining part of the system continues to operate without interruption. After the initial configuration of the FPGA, partial bit-files can be loaded into the FPGA to modify selected regions, without compromising the integrity and the functionality of those parts of the device that are not being affected by the reconfiguration.

Therefore, a system that uses DPR can be conceptually considered as divided in two main parts: a static part and a dynamic part. The static part is configured only once at boot-time with a full bit-file. The dynamic part, which may consist of several independent reconfigurable regions, can be reconfigured multiple times during run-time with different

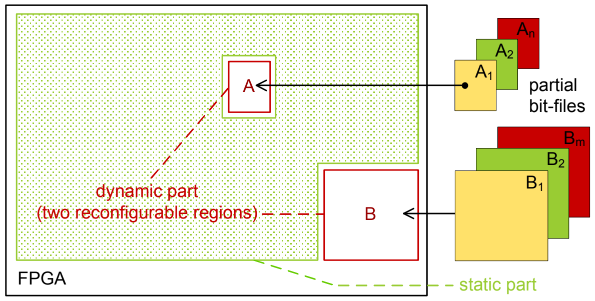


**Figure 1: A FPGA divided into a static and a dynamic part (two reconfigurable regions).**

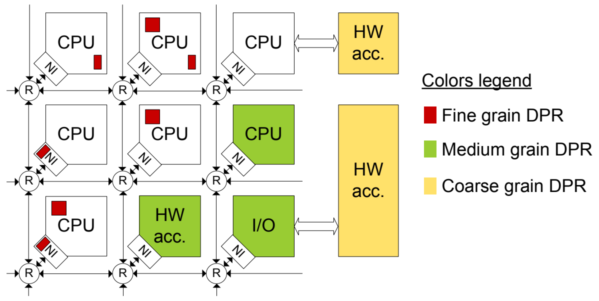


**Figure 2: Example of the three DPR classes in a network-on-chip-based multi-core platform.**

partial bit-files. Figure 1 shows a FPGA divided into a static part and a dynamic part, where the dynamic part consists of two non-topologically-connected reconfigurable regions (A and B). For every region, a partial bit-file can be loaded from a set (e.g., $A_1$, $A_2$, ..., $A_n$) without interfering with the functionality of the static part.

The use of DPR allows the creation of systems with a very high level of flexibility since reconfigurable regions can be dynamically reused by realizing the functionality that is needed at any point in time. This makes more efficient usage of the FPGA hardware resources, leading to a reduction of the FPGA size, with consequent reduction of power consumption and cost. Moreover, a blank (empty) partial bit-file can be loaded into a temporarily not used reconfigurable region in order to reduce the power consumption to the minimum.

Using DPR to obtain these benefits increases the complexity of the design process and of the hardware design itself. The design flow performed with the commercial tools is more articulated and it requires additional steps to generate the partial bit-files. For instance, the definition of the physical areas on the FPGA chip where the reconfigurable regions must be placed and the specification of dedicated timing constrains for the reconfigurable regions. Something that is not required for standard non-reconfigurable design.

From a hardware point of view, additional logic is needed in the borders between the static part and the dynamic part in order to insulate the reconfigurable regions during reconfiguration. Moreover, a hardware controller is also required to manage the reconfiguration process and to move the partial bit-file from the memory where they are stored (on-chip block-RAM or off-chip memory) to the FPGA's configuration memory.

The next section addresses this design challenges keeping in mind that, since we target multi-core platform for hard real-time systems, the time behavior has to be completely predictable. Therefore, also the DPR approach must be performed in a time-predictable manner and be completely analyzable.

## 3. APPROACH AND MODELS

The main idea is to share the dynamic part of the FPGA between the resources used by one or more processors of the platform for a limited period of time (e.g., hardware accelerators, co-processors, I/O units, etc.) or to reconfigure part of the platform (e.g., processors or sections of them, networks-on-chip, etc.) to dynamically adapt the hardware to the actual needs of the software tasks running on it.

In this section we address how to perform this. First we identify 3 classes of DPR, then we present the interfaces available for invoking DPR, we discuss latency aspects of these and finally we propose two models for the DPR feature in hard real-time system.

### 3.1 Classes of DPR

In a multi-core platform, such as T-CREST, we have identified three possible classes of DPR depending on the size of the reconfigured area: fine grain, medium grain, and coarse grain DPR. An example of the DPR classes is shown in Figure 2, where the three granularities are represented in different colors.

We define the DPR as fine grain when the area to be reconfigured is very small, in the order of hundreds of FPGA logic cells (LCs). An example of fine grain reconfiguration is the application of minor changes to a CPU architecture in order to modify, during run-time, its instruction set (shown in red in Figure 2).

A medium grain DPR involves an area in the order of thousands of LCs. An example is a reconfiguration of an entire intellectual property of the platform, such as CPUs, small stateless hardware accelerators, etc. (shown in green in Figure 2).

Finally, a coarse grain DPR involves a large area of the FPGA, in the order of tenths of thousands of LCs. An example of this class is a set of stateful hardware accelerators for compute-intensive operations (fast Fourier transform, encryption/decryption, etc.) to be swapped into large reconfigurable regions and connected to a subset of CPUs with a dedicated network-on-chip or a shared bus (shown in yellow in Figure 2).

Since the reconfiguration time depends on the amount of LCs to be reconfigured, it is immediately clear that a fine grain DPR is very fast. On the contrary, a coarse grain DPR is a relatively slow process.

### 3.2 Interfaces and Reconfiguration Latencies

For XILINX FPGAs, DPR can be performed through different interfaces. In this paper we mention only the two interfaces that we have considered to use: the internal configuration access port (ICAP) and the SelectMap interface.

The ICAP is a primitive found in XILINX FPGAs. It is an internal interface on the FPGA fabric that can be accessed by the hardware implemented on the FPGA itself and it provides direct access to the configuration memory.

**Table 1: Calculated reconfiguration latencies for the three classes of DPR for a XILINX Virtex-6 FPGA.**

| Class of DPR | # of CLBs | Bit-file size | Reconfig. latency |
|---|---|---|---|
| Fine grain | 150 | 45 kB | 110 $\mu$s |
| Medium grain | 2 000 | 680 kB | 1.5 ms |
| Coarse grain | 10 000 | 2.9 MB | 7.3 ms |

It requires the instantiation of an ICAP controller and the logic to drive the interface itself.

The SelectMap interface is a fast external reconfiguration interface that also provides direct access to the configuration memory. It dedicates I/O pins for a bi-directional data bus and control signals.

For a XILINX Virtex-6 FPGA, both interfaces have a maximum data width of 32 bits and a maximum frequency of 100 MHz, hence the maximum transfer speed that can be reached to load a partial bit-file is 3.2 Gb/s. Table 1 shows some calculated values of reconfiguration latencies for the three classes of DPR, based on the aforementioned transfer speed. CLB stands for configurable logic block of XILINX FPGAs. The reconfiguration latency is the time interval needed to load a partial bit-file in a reconfigurable region.

## 3.3 DPR Models

Considering the granularity of the DPR and the associated configuration latencies we have formulated two different models to describe the effect of DPR on the tasks execution and schedule: a task-level DPR model and a mode-level DPR model.

### *Task-Level DPR Model*

Observing the reconfiguration latencies shown in Table 1 we can safely assume that the latency for fine grain DPR is smaller than (or comparable to) the WCET of a software task (maximum possible duration of a task). Therefore, the effect of DPR can be modelled by associating the reconfiguration latency to the tasks to which the reconfiguration is related. In other words, the tasks that uses reconfigurable resources need to include the reconfiguration latency in their WCET before the task set is scheduled.

As an example, Figure 4(a) shows a static schedule period with three tasks $T_1$, $T_2$ and $T_3$ sharing a reconfigurable region. Every time a task is started or resumed, it needs to reconfigure the dynamic part in order to have available the hardware resources to perform its operation. The solid color at the beginning of each task represents the reconfiguration delay added to each task WCET. The last row of the diagram shows how the reconfigurable region is shared between the tasks.

### *Mode-Level DPR Model*

Assuming the reconfiguration latency for coarse grain DPR larger than the WCET of a software task, the reconfiguration of this DPR class must be associated to an operation mode change, where the system, during normal operation, changes a subset of the executing tasks to adapt its behavior to new environment conditions.

The graph in Figure 3 shows the operational mode changes between three modes: $M_1$, $M_2$, and $M_3$. Every mode consists of a set of tasks and a set of resources (e.g., hardware

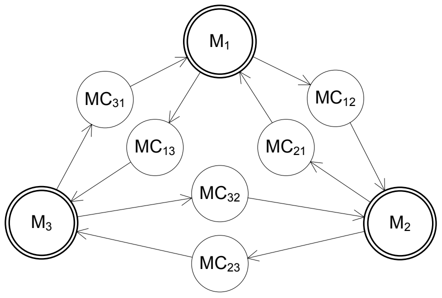


**Figure 3: Graph showing the operational mode changes for three modes.**

accelerators) implemented on the dynamic part of the design that corresponds to a different operational scenario.

A reconfiguration associated with a mode change can be modelled as a task that belongs to a mode change scenario (e.g., $MC_{12}$, $MC_{21}$, etc. in Figure 3). A mode change scenario consists of a set of task that need to be maintained active in the transition between the old and the new mode and a set of tasks that models the reconfiguration process.

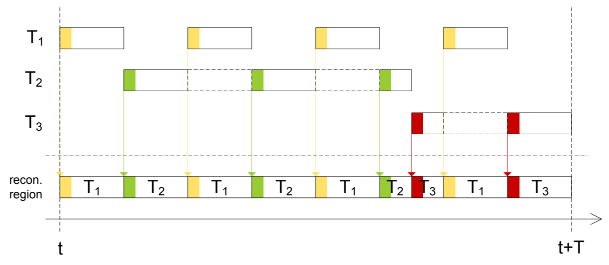
As an example, Figure 4(b) shows a change between two statically scheduled modes $M_1$ and $M_2$. $M_1$ consists of the tasks $T_1$ and $T_2$, while $M_2$ consists of the tasks $T_1$ and $T_3$. We assume that $T_2$ and $T_3$ need different resources to be implemented on the shared reconfigurable region and that the periodic execution of $T_1$ cannot be suspended during the mode change. We also assume that a task that continues its execution through the mode change cannot reconfigure its own resources. We can observe that during the mode change scenario $MC_{12}$ the task $T_1$ continues to run and the task $T_{rec}$, which models the reconfiguration process, is executed. The last row of the diagram shows how the reconfigurable region usage changes between different modes. Also in this case, the solid color represents the reconfiguration delay.

## 3.4 Final Remarks

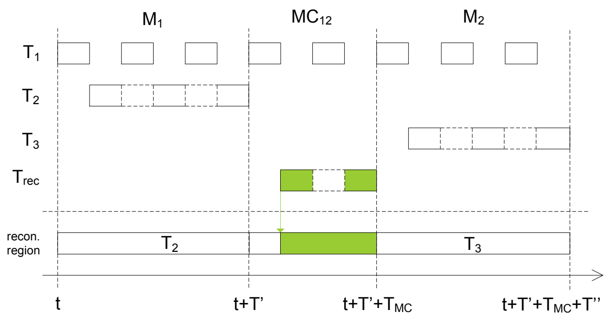
The two models presented above, which are applicable to different classes of DPR, are not exclusive. Fine, medium, and coarse grain DPR can be present in the same architecture and be modelled independently using the proper model. However, since the current FPGA technology allows to reconfigure only one region at a time, interference between tasks can occur and this must be taken into account during the task scheduling. Utilizing the presented models to describe the effects of the DPR makes still possible to apply the traditional shared resources scheduling protocol (e.g., PIP, PCP, SRP, etc.) to properly share the reconfigurable regions between the tasks. Finally, we mention that medium grain DPR can be modelled with both methods depending on the actual relation between the task WCET and the reconfiguration latency. If the reconfiguration latency is smaller or comparable than the task WCET, the task-level model can be applied. Otherwise, the mode-level DPR model needs to be used.

## 4. EVALUATION

We plan to evaluate our approach, models and design utilizing the T-CREST platform. T-CREST is a homogeneous multi-core platform for embedded hard real-time applica-



(a) Task-level DPR model.



(b) Mode-level DPR model.

**Figure 4: Example of time diagrams for the task-level and the mode-level DPR models.**

tions [4]. All features are optimized to simplify static WCET analysis. The T-CREST platform contains several processing cores, called Patmos [5], and it is supported by the WCET analysis tool aiT [6] from AbsInt that allows to statically derive tight WCET bounds. T-CREST contains two time-predictable networks-on-chip (NOCs): a time-division multiplexing message passing NOC between the cores, called Argo [7], and a memory tree NOC towards the shared external main memory.

We are currently developing an ICAP/SelectMap controller that allows to load partial bit-files in a time-predictable manner. Preliminary experiments have been carried out targeting the XILINX Virtex-6 FPGA on the ML605 development board. The tested architecture consists of a single Patmos processor that manage a fine grain reconfigurable region, using the ICAP/SelectMap controller to swap a simple I/O led driver between different configurations. The test allowed to prove the functionality of the controller and to collect some preliminary measurements regarding the reconfiguration latency. With a size of 40 configurable logic blocks, and a bit-file size of 12.7 kB stored in block-RAM, the measured reconfiguration latency is 35 $\mu$s.

Further evaluation is expected in the near future when the ICAP/SelectMap controller and the software tools currently under development will be completed and stable.

## 5. RELATED WORKS

The use of DPR in the context of hard real-time systems and of multi-processor platforms is largely unexplored. However, we briefly mention two works addressing general purpose non-real-time platforms that have been taken into account in our research.

The ReCoBus-builder [8] is a FPGA design oriented framework for component-based reconfigurable systems. It uses DPR to generate systems with one or more reconfigurable areas to be used by different hardware modules.

The LogiCORE IP XPS HWICAP [9] is an IP from XILINX that enables an embedded microprocessor to read and write the FPGA configuration memory through the ICAP interface. This controller, although being widely used, is not designed to be time-predictable. Hence, it is not suitable to be directly used in our design.

## 6. CONCLUSION

This paper presented a preliminary exploration of the use of FPGAs' DPR in the context of multi-core hard real-time systems. It discussed the challenges related to the use of DPR and it presented an approach on how to use the DPR feature to share reconfigurable resource in a time predictable manner. The paper also proposed two models to describe the effect of DPR on the tasks execution and scheduling.

## 7. REFERENCES

[1] L. Wang and F. Y. Wu. Dynamic partial reconfiguration in FPGAs. In *Proc. of IEEE Third International Symposium on Intelligent Information Technology Application*, volume 2, pages 445–448, 2009.

[2] XILINX. UG702: Partial reconfiguration user guide. Technical report, 2012. Online.

[3] ALTERA Corporation. QII51026: Design planning for partial reconfiguration. Technical report, 2013. Online.

[4] M. Schoeberl et al. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 2015. Accepted for pubblication. Online at *http://www.jopdesign.com/doc/t-crest-jnl.pdf*.

[5] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S.Karlsson, and T. Thorn. Towards a time-predictable dual-issue microprocessor: the Patmos approach. In *Proc. of First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems*, pages 11–20, 2011.

[6] R. Heckmann and C. Ferdinand. Worst-case execution time prediction by static program analysis. Technical report. Online at *http://www.absint.de/aiT_WCET.pdf*.

[7] E. Kasapaki et al. Argo: A real-time network-on-chip architecture with an efficient GALS implementation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2015. Accepted for pubblication. Online at *http://www.jopdesign.com/doc/argo-jnl.pdf*.

[8] D. Koch, C. Beckhoff, and J. Teich. ReCoBus-Builder - a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs. In *Proc. of International Conference on Field Programmable Logic and Applications*, pages 4629918, 119–124. Inst. of Elec. and Elec. Eng. Computer Society, 2008.

[9] XILINX. DS586: LogiCORE IP XPS HWICAP (v5.00a) product specifications. Technical report, 2010. Online.

# Regulation versus Flow Control in NoC for Hard Real-time Systems: a Preliminary Case Study

Hamdi Ayed, Jérôme Ermont, Jean-luc Scharbarg, Christian Fraboul
Toulouse University - IRIT - ENSEEIHT
Toulouse, France
{hamdi.ayed2, jerome.ermont, Jean-Luc.Scharbarg, christian.fraboul}@enseeiht.fr

## ABSTRACT

Many-core architectures are promising candidates for the design of hard real-time systems. Inter-core and core to external memory or peripheral communications use the Network-on-Chip (NoC). Such a NoC is typically composed of a set of routers. Internal organization of routers (mainly buffers) as well as flow control aspects impact NoC performances and thus those of the many-core, including the Worst-Case Traversal Time (WCTT) which has to be guaranteed for hard real-time systems.

In this paper we study the impact of flow control aspects on this WCTT. We consider two classes of NoC architectures, representative of the trend in the many-core market: Tilera Tile64-like NoCs where flow control is implemented at the router level and KalRay MPPA 256-like NoCs where flows are regulated at the source node level.

We compute flow WCTT for different configurations and we show that there is no clear winner, since NoC performances highly depend on flow features.

## Keywords

Network-on-Chip, Flow control, Worst-case traversal time

## 1. INTRODUCTION

Many-core architectures are promising candidates to support the design of hard real-time systems. They are based on simple cores interconnected by a Network-on-Chip (NoC). Timing constraints, such as bounded delays, have to be guaranteed for hard real-time systems. Thus worst-case behavior of the NoC is a key feature for such systems.

However, the initial motivation when designing NoCs was to increase the average case throughput. NoCs can thus be used in hard real-time systems using one of the following approaches:

1. analysis of the Worst-Case Traversal Time (WCTT) of flows on existing many-cores,
2. modification of the hardware so that no contentions can occur by design, leading to straightforward WCTT for flows.

Several NoC have been proposed based on the second approach [3, 4, 9]. However, none of these NoCs targeting hard real-time constraints are available in commercially existing many-core architectures, such as for instance the Tilera Tile CPUs [10], the STMicroelectronics P2012/STHORM fabric [8] or the KalRay MPPA [1]. In this work, we focus on these commercially existing architectures, where NoC relies on wormhole switching [7] and Round-Robin Arbitration (RRA) within routers. Using wormhole switching, a packet is divided in flow control digits (flits) of fixed size which are transmitted one by one by routers. The header flit (i.e. the first flit) contains the routing information that defines the path for all the flits of the packet.

NoC implement flow control in order to control buffer occupancy. Two main strategies are considered in commercial many-cores. The first one implements flow control in each router: a packet cannot be forwarded if the next output port is busy. The Tilera Tile64 [10] uses this strategy. The second strategy implements flow regulation in source nodes, in order to bound the traffic. The KalRay MPPA 256 [1] uses this strategy.

The contribution of this paper is to evaluate the impact of these two strategies on flow WCTT. This preliminary evaluation is based on two small case studies.

The rest of the paper is organized as follows. Sections 2 summarizes considered NoC features. Section 3 presents the evaluation. Section 4 concludes and gives some direction for future work.

## 2. DESCRIPTION OF TWO NOC ARCHITECTURES

In this section, we describe two different NoC architectures: Tilera Tile64 and KalRay MPPA 256. The first one uses the classical credit based flow control. The second consists in a source regulation of flows.

### 2.1 Overview of the Tilera Tile64

The Tilera Tile64 is composed of a grid of 64 tiles. Each of them contains a processor engine (core), a private cache and a crossbar switch. The tiles communicate by exchanging packets through the embedded switch. To minimize the interference and to maximize the performance, inter-tile communication uses six independent networks. The traffic related to memory, caches, I/O and processors is transmitted upon distinct networks.

The Tilera Tile64 uses classical wormhole switching: packets are split into several flits and are transmitted flit by flit from the source to the destination tile. The first flit is called the header flit and contains the destination address. When the packet is granted to access to an output port, this output port is locked until the last flit has successfully traversed the switch. The flits follow the same path as the header flit. When the output port is locked, the flits are stored into a small sized (three flits) buffer of the input port, as shown in

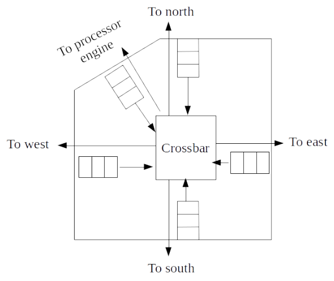


**Figure 1: Tilera Tile64 router [10]**

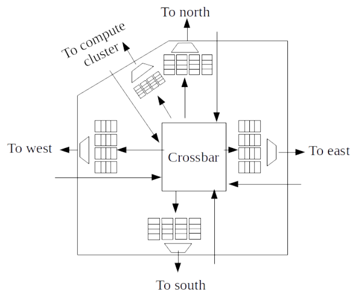


**Figure 2: KalRay MPPA 256 router [1]**

Figure 1. When the output port is freed, the switch fabric uses round-robin arbitration to ensure fairness. Thus, when the transmission of a packet from an input port is terminated, the transmission of a packet from another input port can start.

Control flow uses a credit scheme: the output ports contain a credit count corresponding to how many flits can be stored into the input ports of the next switch. Each time a flit is routed to the output port, the credit is decremented. When the credit count is zero, the flits are blocked into the input buffers. When an input buffer place becomes empty, the credit of the corresponding output buffer (from the previous switch) is incremented.

## 2.2 Overview of the KalRay MPPA 256

The architecture of the KalRay MPPA-256 is different from the one of the Tilera Tile64. It is composed of 16 processing elements (PE) which contain 16 cores each, and two parallel networks-on-chip, one for the data (D-NoC) and one for the control (C-NoC). The network topology is a 2D torus.

D-NoC is dedicated to high bandwidth data transfers. KalRay MPPA-256 uses flow regulation [6] at the source node. This regulation is parametrized by a window length ($\tau$) and a bandwidth quota ($\beta$). At each cycle, the regulator compares the length of the packet to send plus the number of flits already sent during the previous $\tau$ cycles to $\beta$. If not greater, the packet can be sent, a flit each cycle. Using the network calculus theory, these parameters allow to determine the capacity constraints of the links and the router buffer sizes [1]. Consequently the NoC does not need control flow mechanism.

**Table 1: Case study 1: flows set description**

| Flow | Period (cycles) | Length (flits) | T-bound (cycles) | K-bound (cycles) |
|---|---|---|---|---|
| $f_1$ | 1000 | 50 | 257 | 295 |
| $f_2$ | 500 | 100 | 256 | 295 |
| $f_3$ | 1000 | 50 | 278 | 304 |
| $f_4$ | 1000 | 20 | 278 | 317 |

On NoC routers, flows can arrive from different directions. As shown in Figure 2, each direction has its own FIFO buffer at the output port. In that way, flows can be blocked only if they share the same output link. Round-robin is used to determine which packet in the FIFO queues is granted to be transmitted.

## 3. CASE STUDIES AND END-TO-END DELAY ANALYSIS

The goal of this section is to compare the WCTT on Tilera TILE64-like NoCs and KalRay MPPA 256-like NoCs. This comparison is based on two case studies.

## 3.1 Case study 1

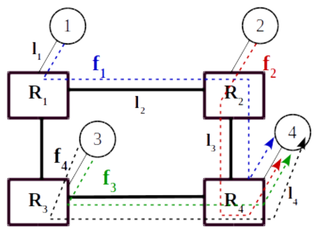


**Figure 3: Case study 1: description**

Figure 3 describes the first case study: a 2x2 Mesh NoC with 4 periodic flows (Table 1) transmitted to core 4. To compute WCTT for each flow, we use the Recursive Calculus [2] for Tilera NoC (T-bound) and Network Calculus [5] for KalRay NoC (K-bound). In the rest of this section, we illustrate these approaches for flow $f_1$.

### *3.1.1 Tilera NoC network*

Several approaches have been proposed for WCTT computation for Tilera-like NoC architecture [2]. For this paper, we use the Recursive Calculus (RC), which gives tighter results than Network Calculus (NC) [2]. Figure 4 illustrates Recursive Calculus principle. For ease of presentation, it shows a simplified version of the worst-case scenario determined by RC approach for flow $f_1$. In this simplified version, the size of each input buffer is equal to one flit and packets size is 3 flits. This scenario can be easily generalized with packets of arbitrary size and larger buffers.

In this scenario, $f_2$ delays $f_1$ on link $l_3$. Thus, $f_1$ is blocked at $R_2$ till the end of transmission of $f_2$. $f_2$ is delayed on link $l_4$ by the packet with the maximum size coming from $R_3$ ($l_3$ in Figure 4). Due to round robin scheduling, $f_1$ is also delayed by one packet coming from $R_3$ ($f_4$ in Figure 4). This leads to the WCTT at the bottom of the the figure. Using actual packet sizes of Table 1, we obtain $d_{e2e}^1 = 7 * d_{sw} + 2 * L_3/C + L_2/C + L_1/C = 257\ cycles$.

It should be noted that $f_3$ is considered twice at $l_4$, since its packet size is larger than one $f_4$. Obviously, such a scenario is not feasible, due to $f_3$ period. Thus RC approach introduces some pessimism.

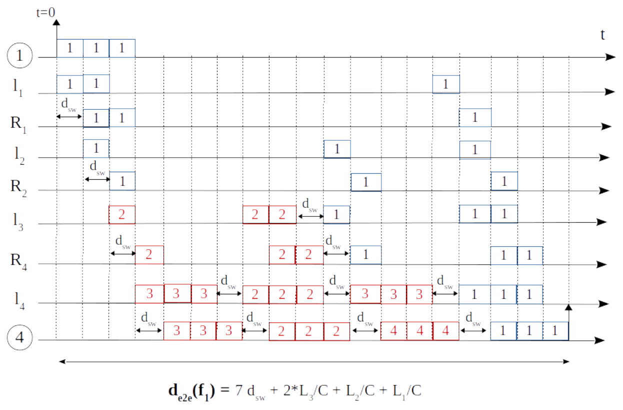


**Figure 4: Case study 1: recursive calculus application for flow $f_1$ with Tilera NoC**

### 3.1.2 KalRay NoC network

As presented in Section 2, flows are regulated at source node level and no flow control is applied at router level. Thus flits arriving in a router are stored in corresponding output port buffers (allocated to their input link). A packet-by-packet round robin scheduling is applied for each output port.

In this paper we consider a classical network calculus approach [5] for the WCTT analysis of flows. In such an approach, each flow $f_i$ is modeled by an arrival curve $\alpha_i$ which overestimate its traffic. Each source node $s_i$ or router output port $R_{i_j}$ is modeled by a minimum service curve $\beta_{s_i}$ or $\beta_{R_{i_j}}$. In the case study in Figure 3, each flow $f_i$ is defined by a period $T_i$ and a packet length $L_i$. Thus the arrival curve of a flow $f_i$ is $\alpha_i(t) = \sigma_i + \rho_i * t$. $\sigma_i$ is the maximum burst $L_i$ and $\rho_i$ is the maximum long term rate $L_i/T_i$. Arrival curves of the flows in Figure 3 are:

$$\begin{aligned}\alpha_1(t) &= \alpha_3(t) = 50 + 0.05 * t\\ \alpha_2(t) &= 100 + 0.2 * t\\ \alpha_4(t) &= 20 + 0.02 * t\end{aligned}$$

Concerning service curves, we assume that every link in the NoC has a transmission rate of $C = \frac{1\ flit}{cycle}$, the technical latency of source nodes is negligible and the technical latency of a router is equal to $d_{sw} = 1\ cycle$. Thus the overall service curves for a source node $s_i$ and a router output port $R_{i_j}$ are:

$$\begin{aligned}\beta_{s_i}(t) &= 1 * t\\ \beta_{R_{i_j}}(t) &= max(0, 1 * (t - d_{sw})) \quad i \in \{1, 2, 3, 4\}\end{aligned}$$

where $d_{sw}$ represents the overall router latency.

Source nodes implement a First Come First Served policy while a round-robin scheduling is applied by router output

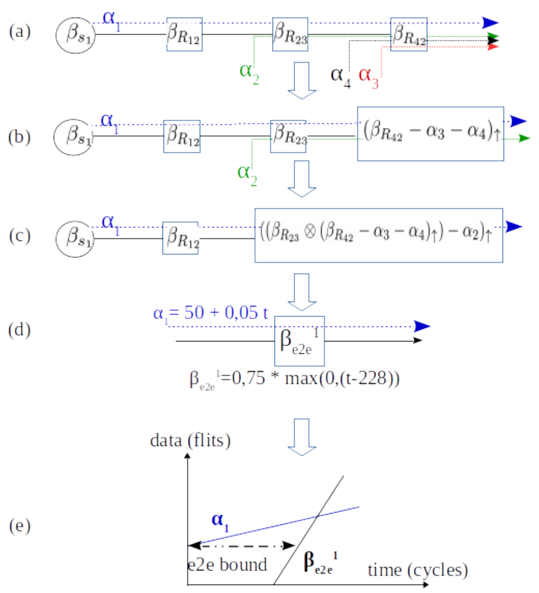

(a)
(b)
(c)
(d)
(e)
$\beta_{s_1}$
$\beta_{R_{12}}$
$\beta_{R_{23}}$
$\beta_{R_{42}}$
α1
α2
α4
α3
$(\beta_{R_{42}} - \alpha_3 - \alpha_4)_\uparrow$
$((\beta_{R_{23}} \otimes (\beta_{R_{42}} - \alpha_3 - \alpha_4)_\uparrow) - \alpha_2)_\uparrow$
α1 = 50 + 0,05 t
β e2e 1
β e2e 1 =0,75 * max(0,(t-228))
data (flits)
α1
e2e bound
β e2e 1
time (cycles)

**Figure 5: Case study 1: WCTT for flow $f_1$ with KalRay NoC**

ports. In order to deal with both policies we apply the (pessimistic) blind multiplexing model [5]: when two flows $f_1$ and $f_2$ share an output port $R_{i_j}$ implementing any scheduling algorithm, the minimum service curve for flow $f_1$ is:

$$\beta^1_{R_{i_j}}(t) = (\beta_{R_{i_j}}(t) - \alpha_2(t))_\uparrow$$

where $\beta_{R_{i_j}}$ is the overall service curve offered by $R_{i_j}$ and $\beta_\uparrow$ is the positive and non-decreasing upper closure defined as $(\beta)_\uparrow(t) = max(0, sup_{0 \le s \le t}\beta(s))$.

The WCTT computation for a flow $f$ is based on the following result. When a flow $f$ traverses two nodes $R_{i_j}$ and $R_{k_l}$ in sequence, offering service curves $\beta_{R_{i_j}}$ and $\beta_{R_{k_l}}$, respectively, network calculus theory [5] establishes that the concatenation of these two nodes offers the service curve $\beta_1 \otimes \beta_2$. $\beta_1 \otimes \beta_2(t) = inf_{0 \le s \le t}\beta_1(t - s) + \beta_2(s)$ is the Min-Plus convolution.

Figure 5 illustrate the WCTT computation for flow $f_1$:

- (a) first, service curves for source node $s_1$ and traversed output ports at crossed routers are computed as described previously in this section;
- (b) a service curve for aggregate flow $\{f_1, f_2\}$ at output port $R_{43}$: $(\beta_{R_{43}} - \alpha_3 - \alpha_4)_\uparrow = max(0, 1 * (t - d_{sw})) = max(0, 0.93 * (t - 77))$;
- (c) a service curve for flow $f_1$ offered by the system composed of the sequence $\{R_{23}, R_{42}\}$: $(\beta_{R_{12}} \otimes (\beta_{R_{43}} - \alpha_3 - \alpha_4)_\uparrow - \alpha_2)_\uparrow = max(0, 1*(t - d_{sw})) = max(0, 0.75 * (t - 227))$;
- (d) an end-to-end individual service curve for flow $f_1$: $\beta^1_{e2e} = max(0, 0.75 * (t - 228))$;
- (e) as explained in [5], we can compute a bound on the WCTT of flow $f_1$ as the maximum horizontal deviation between $\alpha_1$ and $\beta^1_{e2e}$: $h(\alpha_1, \beta^1_{e2e}) = 295\ cycles$

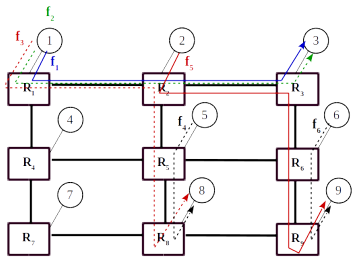


**Figure 6: Case study 2: description**

**Table 2: Case study 2: flows set description**

| Flow | Period (cycles) | Length (flits) | T-bound (cycles) | K-bound (cycles) |
|---|---|---|---|---|
| $f_1$ | 1000 | 50 | 523 | 337 |
| $f_2$ | 500 | 100 | 523 | 282 |
| $f_3$ | 1000 | 50 | 774 | 276 |
| $f_4$ | 500 | 100 | 154 | 190 |
| $f_5$ | 1000 | 50 | 774 | 276 |
| $f_6$ | 500 | 100 | 154 | 190 |

As we can see in Table 1, for case study 1 using Tilera Tile64-like NoC leads to lower WCTT bounds than KalRay MPPA 256-like NOC. All flows in this case study suffer only from direct blocking in routers, due to round robin arbitration. Using source regulation or flow control does not impact WCTT. The main reason for the differences in results of Table 1 is the worst-case delay computation, which is pessimistic for KalRay MPPA, because we overestimate round-robin impact. Removing this pessimism is an open problem, but it should lead to comparable results for KalRay MPPA and Tilera Tile. In such a situation Tilera Tile is probably the best choice, since the overall buffer size is smaller.

### 3.2 Case study 2

Figure 6 and Table 2 describe the second case study: a 3x3 mesh network with 6 periodic flows. Using the same approaches as described for case study 1, we computed WCTT for Tilera (T-bound) and KalRay NoCs (K-bound) and we reported them in Table 2.

As reported in Table 2, for case study 2, using KalRay-like NoC implies lower WCTT bounds for flows $f_1$, $f_2$, $f_3$ and $f_5$. The source node regulation strategy implemented by KalRay MPPA is clearly better for these flows. This is due to the fact that the considered configuration leads to indirect blockings when router flow control is used. These indirect blockings significantly increase flow WCTT. Conversely they do not impact WCTT when source node regulation is used. For flows $f_4$ and $f_6$, which do not suffer from indirect blocking, using Tilera Tile64-like NoC leads to lower WCTT bounds than KalRay MPPA 256-like NoC.

## 4. CONCLUSIONS AND FUTURE WORK

In this paper, we show that flow control implemented in NoC has a significant impact on flow WCTT. We compare source node regulation as implemented in KalRay MPPA and router flow control as implemented in Tilera Tile. We consider two small case studies. These two case studies show that flow WCTT highly depends on flow control strategy. We show that source node regulation leads to smaller WCTT for one case study while router flow control is better for the other one. It means that there is no clear winner and deeper studies are needed in order to be able to determine the most suitable flow control strategy from flow features. This should be based on a much larger evaluation of these strategies, based on representative case studies.

Up to now, we have considered very basic WCTT approaches. For instance, network calculus approach for KalRay MPPA doesn't make any assumption on flow scheduling, which might be very pessimistic. More elaborate approaches have to be considered.

Another question concerns the impact of buffer size on WCTT. To what extend can we decrease the flow WCTT by increasing buffer size?

## 5. REFERENCES

[1] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager. Time-critical computing on a single-chip massively parallel processor. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, March 24-28, 2014*, pages 1–6, 2014.

[2] T. Ferrandiz, F. Frances, and C. Fraboul. A Sensitivity Analysis of Two Worst-Case Delay Computation Methods for SpaceWire Networks. In *Proc. of the 24th Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 47–56, Pisa, Italy, July 2012.

[3] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design & Test of Computers*, 22(5):414–421, 2005.

[4] A. Hansson, M. Subburaman, and K. Goossens. Aelite: A flit-synchronous network on chip with composable and predictable services. In *Proc. of the Conf. on Design, Automation and Test in Europe (DATE'09)*, pages 250–255, Nice, France, 2009.

[5] J. Leboudec and P. Thiran. *Network Calculus.* Springer Verlag LNCS volume 2050, 2001.

[6] Z. Lu, M. Millberg, A. Jantsch, A. Bruce, P. van der Wolf, and T. Henriksson. Flow regulation for on-chip communication. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 578–581, April 2009.

[7] L. Ni and P. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Transactions on Computers*, 26(2):62–76, Feb 1993.

[8] D. Rahmati, S. Murali, L. Benini, F. Angiolini, G. De Micheli, and H. Sarbazi-Azad. Computing accurate performance bounds for best effort networks-on-chip. *IEEE Transactions on Computers*, 62(3):452–467, March 2013.

[9] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proc. of the Intl. Symp. on Networks-on-Chip (NOCS)*, pages 152–160, Copenhagen, Denmark, May 2012.

[10] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal. On-chip interconnection architecture of the tile processor. 2007.