ECRTS 2019

Proceedings of the Work-in-Progress Session

July 10th, 2019
Stuttgart, Germany

Editor:
Andrea Bastoni

Message from the Chair

It is my pleasure to welcome you to the Work-in-Progress (WiP) session of the 31st Euromicro Conference on Real-Time Systems (ECRTS 2019). The WiP session is dedicated to new and on-going research in the field of real-time and embedded systems. The WiP session provides an opportunity for researchers to communicate their evolving ideas, to discuss them with the real-time community, and to receive feedback which will help them identify new trends and gather new insights on their on-going research.

This year, the Work-in-Progress session features five excellent contributions in multiple fields of real-time systems. Each paper summarizes in three pages the motivation for the analyzed problem, provides insights on the relevance of the topics, and discusses possible solutions. Each paper is also accompanied by a poster that will help the authors to discuss further details with the community during the reception following the presentations.

I am confident that many of the contributions presented today will appear as full conference or journal papers in the near future, and I wish the authors to find inspiration and fruitful collaborations in this session.

The WiP proceedings are available online on the ECRTS 2019 website:

`https://www.ecrts.org/work-in-progress/`

I would like to express my gratitude to the members of the Program Committee who have done a tremendous job in providing constructive feedback for each contribution in a very limited time. Additionally, I would like to thank the organizers of ECRTS 2019 for their support in organizing the WiP session.

On behalf of the Program Committee, I wish you a pleasant session. I hope that you will enjoy the presentation and that you take the opportunity to discuss the papers with the authors during the poster session.

Andrea Bastoni
SYSGO GmbH
ECRTS 2019 Work-in-Progress Chair

Organizers

Andrea Bastoni, *SYSGO GmbH, Germany*

Program Committee

Martina Maggio, *Lund University, Sweden*
Ahlem Mifdaoui, *DISC, Univ. Toulouse, France*
Geoffrey Nelissen, *CISTER, ISEP, Portugal*
Bryan Ward, *MIT Lincoln Laboratory, USA*

ECRTS 2019 Work-in-progress Papers

Design and implementation of an FPGA-based NoC for Real Time Systems
Yilian Ribot González and Geoffrey Nelissen 1

Towards a generic platform for the distribution of avionics applications on manycores
Ghina Abdallah, Jérôme Ermont, Sandrine Mouysset and Jean-Luc Scharbarg 4

ResilienceP Analysis: Bounding Cache Persistence Reload Overhead for Set-Associative Caches
Syed Aftab Rashid, Geoffrey Nelissen and Eduardo Tovar 7

Towards Real-time Self-adaptation Using a Verification Mechanism
Hiroyuki Nakagawa, Hiroki Tsuda and Tatsuhiro Tsuchiya 10

Towards Robust and Cost-Effective Critical Real-Time Systems under Thermal-Aware Design
Javier Pérez Rodríguez and Patrick Meumeu Yomsi 13

Design and implementation of an FPGA-based NoC for Real Time Systems

Yilian Ribot González and Geoffrey Nelissen
CISTER Research Centre, ISEP, Polytechnic Institute of Porto, Portugal

***Abstract*—In order to communicate, cores of a multi-core platform traditionally relied on shared busses. However, with the increasing number of computation nodes integrated in multi- and many-core platforms, Network-on-Chips (NoCs) emerged as a new communication medium in Systems-on-Chips (SoCs). Hoplite-RT is a new NoC design that was recently proposed in [1]. Hoplite-RT has a compact design, is easy to analyze, and was specifically tailored to target FPGAs. In this work, we introduce priority-based routing to Hoplite-RT and change the network topology so as to improve packets' Worst-Case Traversal Time (WCTT).**

1. Introduction

SoCs are usually composed of several, possibly heterogeneous, processing elements. In order to communicate, the processing elements used to rely on shared busses. However, due to the large increase of on-chip elements during the last decade, communication through shared busses is not an appropriate solution for such platforms anymore. Indeed, at most one node can take control of a bus and transmit data at each cycle. This causes a bottleneck for the overall system. NoCs have been identified as a good alternative to palliate this issue [2]. NoCs are router-based packet switching networks and hence allow several processing elements to transmit messages in parallel [3]. As discussed in [4], NoCs have remarkable scalability, parallelism, and re-usability properties, and help meet system-wide power and timing constraints.

FPGAs are chips made of reconfigurable elements that can be programmed to implement virtually any digital functionality. Therefore, they are well suited to the development of custom-made SoCs. FPGAs allow to design systems with a high degree of parallelism and high data processing rate at a relatively low cost. However, FPGAs expose a limited number of reconfigurable elements and most FPGAs do not supply enough resources to embed complex NoC designs together with a large number of processing elements.

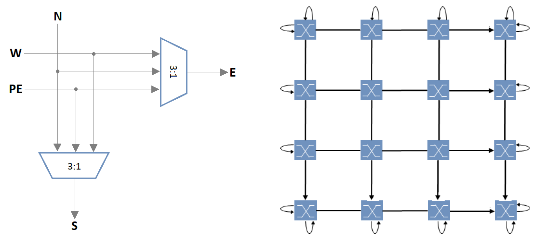
The literature on NoCs is extensive. Most of the proposed solutions that present suitable properties for real-time systems (i.e., bounded worst-case timing behaviors) rely on wormhole switching with virtual channels, buffering, and often some sort of priority-driven routing arbitration [5], [6]. These concepts allow to develop powerful NoC infrastructures with bounded WCTT but suffer from two main drawbacks: (1) they are expensive to implement in FPGA platforms; and (2) their complexity renders their analysis extremely complex as shown by the number of flaws that were recently discovered in existing works [7].

In complete opposition to the solutions mentioned above, Hoplite is a newly proposed NoC infrastructure [8] that reduces the NoC features to their bare minimum and hence decreases considerably the network analysis complexity and implementation cost (in terms of FPGA resources utilization). Introduced by Wasly et al in [1], Hoplite-RT is a modified version of Hoplite that provides an upper bound on the NoC WCTT. Each Hoplite-RT router has three input ports (North (N), West (W) and Programming Element (PE)) and two output ports (South (S) and East (E)) (see Figure 1(a)). Hoplite-RT assumes that packets transmitted through the NoC are composed of a single flit with two fields: the destination address and the transmitted data. Hoplite-RT connects routers in a torus topology (Figure 1(b)) and employs a modified version of X-Y routing (packets first travel horizontally on the X axis, and then vertically on the Y axis). Specifically, the Hoplite-RT routing policy is built upon the concept of deflection to avoid the cost of packet buffering. When two packets coming from the W and N port of a router conflict for the S port, Hoplite-RT gives the highest priority to the packet originating from the W port. The packet originating from the N port is then deflected toward to the E port inconsiderately of its final destination. However, as the deflected packet is now traveling along the X axis, it will have the highest priority when it will require to go south again. Therefore, the maximum number of deflections suffered by a packet can be upper-bounded (see Sec. 2).

Contribution. In Hoplite-RT, a packet may be deflected after each and every hop on the Y-axis, thereby leading to possibly large WCTTs. Furthermore, Hoplite-RT treats all packets identically. It does not allow to associate different priorities, and hence quality of services to different packets. Nonetheless, Hoplite-RT is a compact, easy to analyze, easy to implement, and inexpensive design. For those reasons, we are interested in developing a NoC that keeps the advantages of Hoplite-RT while improving its real-time capabilities. This paper represents the first stages of our research: (1) we propose a solution to limit the number of deflections and hence the WCTT of a packet; and (2) we introduce a notion of quality of service in the routing policy.

2. Background

The architecture of an Hoplite-RT router is shown in Figure 1(a). It is implemented using two multiplexers of three inputs. Hoplite-RT takes advantage of the possibility of *"fracturing"* the Look Up Tables (LUTs) of modern FPGAs to reduce the implementation cost of the expensive crossbar multiplexers. The modern families of Xilinx



(a) Hoplite-RT router (b) Hoplite-RT torus topoligy

Figure 1. Hoplite-RT design.

TABLE 1. RESOURCES UTILIZATION IN A VIRTEX-7 485T FPGA

Router	LUTs	FFs
Hoplite-RT router	85	139
Hoplite-RT router + Priorities	86	139
Hoplite-RT router + Priorities + New Topology	88	139

FPGAs present 6-inputs LUTs that can be fractured in two 5-inputs LUTs sharing the same five input signals. Since each 3:1 multiplexer can be implemented with a 5-inputs LUT, the two multiplexers of the router can be implemented with a single 6-inputs LUT. The first row of Table 1 shows the cost in terms of LUTs and flip-flops (FFs) of a 64bits Hoplite-RT router implemented in a Xilinx Virtex-7 485T FPGA after fracturation.

The WCTT wc_{TT} of a packet transmitted over Hoplite-RT between two nodes with coordinates (x_o, y_o) and (x_d, y_d) in a mesh of size $S_x \times S_y$ is given by Equation (1) (in clock cycles)

$$wc_{TT} = h_x + h_y + (h_y \times S_x) + 2, \tag{1}$$

where h_x and h_y are the distances travelled by the packet on the X and Y-axis, respectively, when it does not contend with any other packet (i.e., without any deflection). That is,

$$h_x = (x_d - x_o + S_x) \mod S_x \tag{2}$$
$$h_y = (y_d - y_o + S_y) \mod S_y \tag{3}$$

The term $(h_y \times S_x)$ accounts for potential deflections costs.

3. Preliminary results

In this section, we describe the modifications we made to the Hoplite-RT design in order to: (1) introduce a notion of priority in the routing policy; and (2) decrease the worst-case traversal time of a packet.

3.1. Priority-based routing

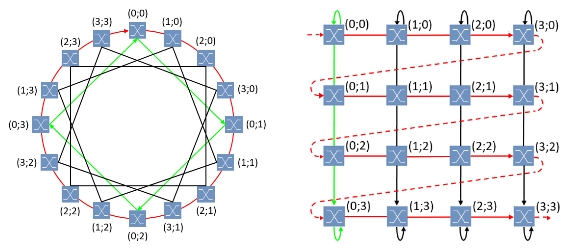
We first modify Hoplite-RT to introduce a two priority levels (Low and High) scheduling scheme. This approach allows us to provide different levels of quality of service to different packets and hence decrease the *average* traversal time of the high priority packets. The WCTT of high priority packets may also be improved depending on the application mapping and the network configuration. However, we leave that analysis for future work.

In Hoplite-RT, the packets coming from the W port always have the highest priority. Instead, in our new design, low priority packets coming from the W port will *never* be permitted to deflect high priority packets coming from the N port. That is, if a high priority packet coming from the N port and a low priority packet coming from the W port conflict for the S port, then the N packet wins the right to use the S port, and the W packet is deflected towards the E port. To support this new routing policy, the packet priority is encoded in its most significant bit.

We observe from Table 1 that this simple modification consumes only one additional 6-inputs LUT in comparison to a normal Hoplite-RT router.

3.2. New topology

Even though it looks beneficial, the new priority-based routing policy described in Section 3.1 is in fact extremely inefficient; the WCTT of high priority packets remains unchanged (only their average-case traversal time is reduced), but more importantly, the WCTT of low priority



(a) Ring representation (b) Equiv. mesh representation

Figure 2. Directional ring with bypasses topology

packets is not bounded anymore. Indeed, a low priority packet may always contend with high priority ones and therefore never be able to use the S port of a router.

In this section, we propose a new network topology that, in most cases, reduces the WCTT of high priority packets by a factor of two, and allows to reinstate the same WCTT bound for the low priority packets as in the original design of Hoplite-RT.

In the standard torus topology (see Fig. 1(b)), a deflected packet will hop through S_x routers (where S_x is the number of routers on the X axis), before entering again in the same router where it was initially deflected. That is, the packet did not progress toward its destination after those S_x additional hops. We prevent this issue to happen by changing the network topology to a directional ring with bypasses (see Fig. 2(a)). In that new topology, all routers are connected by a single unidirectional ring (red links). Then, every pair of routers that are S_x positions apart on the ring are connected by a bypass (green and black links). Equivalently, if we look at the network as a mesh (see Fig. 2(b)), it connects the E port of the last router in row number y to the W port of the first router in row number $(y+1) \mod S_y$. Then, the bypasses correspond to the links on the columns of the mesh.

Thanks to this new topology, when a packet is deflected, it reaches the same router as it would have if it was not deflected, after S_x hops. That is, the packet always progresses toward its destination even when deflected. Consequently, the WCTT decreases.

However, this new topology requires to modify the router design. Indeed, in the particular case where two packets arrive at the same instant in the same router (via the W and N ports) and that router is their destination, Hoplite-RT would solve the conflict by deflecting one of the two packets to the E port. This situation causes a remarkable increase in the WCTT of the deflected packet. We aim at solving this issue by allowing the programming element connected to the router to read both packets simultaneously. This necessarily increases the design cost of the router and the programming element. The cost of implementing a router with priority routing in the new topology is shown in Table 1. It requires only three additional 6-inputs LUTs in comparison to the original Hoplite-RT design.

3.3. Bound on the WCTT

The WCTT of a packet is defined as follow:

$$wc_{TT} = n_{hops} + n_{def} \times c_{def}, \tag{4}$$

where n_{hops} is the number of hops in a network with zero load (i.e., when the packet does not suffer any deflection), n_{def} is the maximum number of deflections suffered by the packet on its route, and c_{def} is the cost of a deflection.

The term n_{hops} is defined as:

$$n_{hops} = h_r + h_b + 2, \tag{5}$$

where h_r and h_b are the number of hops on the ring and bypasses, respectively, and the additional two hops account for the injection (at the source node) and exit (at the destination node) of the packet into and from the network. We prove bounds for each of those terms.

Lemma 1. The number of hops on the ring in a zero-load network is given by $h_r = (x_d - x_o + S_x) \mod S_x$.

Proof: According to our routing policy, each packet travels first through the ring from the origin router at coordinate (x_o, y_o) until it reaches a router with the same X coordinate x_d as the destination. According to the topology presented in Fig. 2(b), the number of hops on the ring is:

$$h_r = \begin{cases} x_d - x_o & \text{when } x_d \geq x_o \\ x_d - x_o + S_x & \text{when } x_d < x_o \end{cases}$$
$$= (x_d - x_o + S_x) \mod S_x$$

□

Lemma 2. The number of hops on a bypass in a zero-load network is given by $h_b = (y_d - y_o' + S_y) \mod S_y$ where

$$y_o' = \begin{cases} y_o & \text{when } x_d \geq x_o \\ y_o + 1 & \text{when } x_d < x_o \end{cases} \tag{6}$$

Proof: Remember that a bypass in Fig. 2(a) corresponds to a column in Fig. 2(b). Let S_y be the number of routers in a column, and y_o' be the Y coordinate of the router at which the packet stops travelling on the ring and starts using bypasses (i.e., the first router with the same X coordinate x_d as the destination). Then, according to the router numbering shown in Fig. 2(b),

$$y_o' = \begin{cases} y_o & \text{when } x_d \geq x_o \\ y_o + 1 & \text{when } x_d < x_o \end{cases}$$

and the number of hops on the Y axis of the mesh is:

$$h_b = \begin{cases} y_d - y_o' & \text{when } y_d \geq y_o' \\ y_d - y_o' + S_y & \text{when } y_d < y_o' \end{cases}$$
$$= (y_d - y_o' + S_y) \mod S_y$$

□

The maximum number of deflections n_{def} that a packet may suffer is different for high and low priority packets. We analyze both cases in Lemmas 3 and 4.

Lemma 3. The maximum number of deflections suffered by a high priority packet is given by $n_{def} = \left\lfloor \frac{h_b}{2} \right\rfloor$.

Proof: Let P denote a high priority packet. When P enters a router from the W port and requests the S port, it cannot be deflected. Then, in the following router of the Y axis, P will be a packet coming from the N port. In this new router, P may be deflected toward the E port by another high priority packet conflicting for the S port. Subsequently, P will travel on the ring until it reaches a router with the same X coordinate as its destination. The whole process will then repeat until P arrives to its destination router. That is, P may be deflected at half the routers it traverses that share the same X coordinate as its destination, i.e., it may be deflected $\left\lfloor \frac{h_b}{2} \right\rfloor$ times. □

Lemma 4. The maximum number of deflections suffered by a low priority packet is given by $n_{def} = h_b$.

Proof: A low priority packet entering from the W port may always be deflected to the E port. Therefore, a low-priority packet may be deflected as many times as it may try to use a bypass, i.e., h_b times. □

The additional cost in terms of hops introduced by each deflection is analyzed in Lemma 5.

Lemma 5. The cost of a deflection is $c_{def} = S_x - 1$.

Proof: When a packet is deflected, it must hop through S_x routers on the ring to reach the same router as it would have if it could have used the bypass instead, i.e., though S_x routers instead of 1, thereby leading to an additional cost of $S_x - 1$. □

4. Conclusion and Future works

In this paper, we presented solutions to improve the timing performance of Hoplite-RT with a marginal increase of the FPGA resource utilization. We first introduced a notion of priority in the routing policy. Then, by changing the network topology to a directional ring with bypasses, we reduced the number of deflections and therefore the WCTT of high priority packets, and maintained the WCTT of low priority packets.

Yet, because each packet may suffer a different number of deflections, Hoplite-RT does not guarantee that packets will be received at the destination router in the same order as they were emitted at the origin router. Therefore, with the current design, long messages can difficulty be split in several flits sequentially injected in the network. As future work, we plan on developing solutions to ensure that the packets arrive in an orderly fashion at the destination router while keeping the advantages of Hoplite-RT and trying to increase its cost as little as possible. We are also working on solutions to map applications on nodes and configure packet injection rates at each node to fully take advantage of the new network design discussed in this paper, e.g., to optimize the link bandwidth usage, or to improve the WCTT of high priority packets at the detriment of low priority ones.

Acknowledgements. This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234), by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement through the European Regional Development Fund (ERDF), and by national funds through FCT within project POCI-01-0145-FEDER-029119 (PReFECT).

References

[1] S. Wasly, R. Pellizzoni, and N. Kapre, "HopliteRT: An efficient FPGA NoC for real-time applications," in *ICFPT 2017*, 2017.

[2] L. Benini and G. De Micheli, "Networks on chip: a new paradigm for systems on chip design," in *Proc. of DATE 2002*, 2002.

[3] A. Agarwal and R. Shankar, "Survey of network on chip (noc) architectures and contributions," *JECA 2009*, vol. 3, 2009.

[4] C. A. Zeferino and A. A. Susin, "Socin: a parametric and scalable network-on-chip," in *Proc. of 16th SBCCI*, 2003.

[5] Y. Huan and A. DeHon, "FPGA optimized packet-switched NoC using split and merge primitives," in *Proc. of FPT 2012*, 2012.

[6] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. Wilson, M. Wrighton, and A. DeHon, "Packet switched vs. time multiplexed fpga overlay networks," in *Proc. of 14th IEEE FCCM*, 2006.

[7] N. Borislav, S. Tobuschat, L.Soares, R. Ernst, and A. Burns, "Real-time analysis of priority-preemptive nocs with arbitrary buffer sizes and router delays," *RTS 2019*, vol. 55, no. 1, 2019.

[8] N. Kapre and J. Gray, "Hoplite: Building austere overlay nocs for fpgas," in *Proc. of 25th FPL*, 2015.

Towards a generic platform for the distribution of avionics applications on manycores

Ghina Abdallah, Jérôme Ermont, Sandrine Mouysset, Jean-Luc Scharbarg
IRIT - Université de Toulouse
2 rue Charles Camichel
31000 Toulouse, France
{firstname.lastname}@irit.fr

***Abstract*—The interconnection of many-cores by an avionics full duplex switched Ethernet network (AFDX) is envisioned for future avionics architecture. The principle is to distribute avionics functions on these many-cores. Many-cores are based on simple cores interconnected by a Network-on-Chip (NoC). The allocation of functions on the available cores as well as the transmission of flows on the NoC has to be performed in such a way that avionics timing constraints are never violated. Several theoretical solutions have been proposed for this distribution. However they have not been evaluated on real architectures. In this paper we introduce a framework for the prototyping of such implementations. This framework is based on the existing ProNoC tool which allows the configuration of an FPGA as a NoC. The goal is to be able to compare distribution solutions with different NoC features in terms of scheduling or routing.**

***Index Terms*—Many-cores, NoC, task distribution, avionics**

I. Application Domain and Challenge

Aircrafts include numerous electronic equipments. Some of them, like flight control and guidance systems, provide flight critical functions, while others may provide assistance services that are not critical to maintain airworthiness. Current avionics architecture is based on the integration of numerous functions with different criticality levels into single computing systems (mono-core processors) [1]. These computing systems are interconnected by an AFDX (Avionics Full Duplex Switched Ethernet) [2]. As depicted in the upper part in Figure 1, the End System (ES) provides an interface between a processing unit and the network.

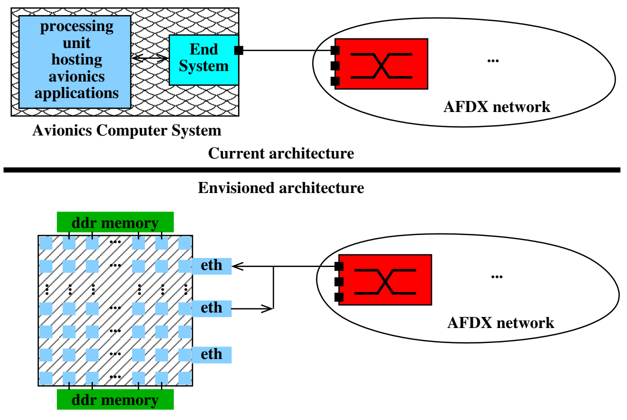


Fig. 1. An AFDX network.

Mono-core architectures are being replaced by multi- or many-core ones in many contexts. This move is also envisioned in aircrafts. However, multi-core architectures are based on complex hardware mechanisms whose temporal behavior is difficult to master. Conversely many-core architectures are based on simpler cores interconnected by a Network-on-Chip (NoC). These cores are more predictable [3]. Thus, many-cores are promising candidates for avionics architecture. Such an architecture integrating many-cores is illustrated in the lower part in Figure 1. A typical many-cores architecture provides Ethernet interfaces which are used for the connection with the AFDX network. Additionaly, memory controllers manage access to DDR. An an example, Tilera Tile64 has 3 Ethernet interfaces and 4 memory controllers [4].

The envisioned avionics architecture depicted in the lower part in Figure 1 is a mixed NoC/AFDX architecture. Avionics functions are distributed on the available many-cores. Communications between two functions allocated on the same many-cores (local functions) use the NoC, while the communications between two functions allocated on different many-cores (remote functions) use both the NoC and the AFDX. Main constraints on this communication are the following:

1) end-to-end transmission delay has to be upper-bounded by an application defined value,
2) frame jitter at the ingress of the AFDX network has to be smaller than a given value (typically 500 μs).

The first constraint concerns local and remote functions, while the second one only concerns remote functions. Transmission delays on the NoC have an impact on both constraints. These delays can vary for different reasons. First, a frame can be delayed by other frames crossing the same routers (router contentions). Second, in the case of a transmission between remote functions, the Ethernet controller can be busy, transmitting another frame (controller contention).

The mapping of functions on the many-cores has a major impact on this NoC delay variation. [5] proposes a mapping strategy that minimizes router contention. In this strategy, each core is allocated at most one function and each avionics flow is managed by its source function. [6] proposes a different strategy, based on a static scheduling of Ethernet transmissions: each transmission is assigned a periodic slot in a table. Thus there are no more controller contentions and router contentions are reduced, thanks to the mapping of functions on cores.

II. Motivation

Both [5] and [6] consider Tilera Tile64 many-cores [4]. However delay computation are based on a model of the many-core and no implementation is provided.

Therefore the first motivation of this study is to map avionics functions on a hardware platform. One goal is to validate results in [5] and [6].

The second motivation of this study is to be able to tune many-core features, mainly NoC ones. Different many-cores implement different topologies, different buffer sizes in router ports, different scheduling algorithms in routers or different routing strategies. For instance, Kalray MPPA [7] has larger buffers than Tilera Tile64.

Therefore our goal is to map avionics functions on a generic hardware platform that can be configured, based on existing NoC features. As a first step, we consider a single many-core.

III. Problem statement

The problem is to distribute a set of n applications $A_0, \ldots, A_{n-1}$ on a many-core. Each application A_i is composed of n_i communicating tasks $t_{i,0}, \ldots, t_{i,n_i-1}$ Communication between tasks are modelled by a graph. The number of cores as well as the NoC topology, buffer size and scheduling algorithm in routers, routing are configurable. The distribution assumes a mapping strategy, e.g. SHiC [8], Map$_{\text{IO}}$ [9], strategies proposed in [5] and [6] or an ad hoc one.

The resulting mapping is implemented on a hardware platform, typically an FPGA, e.g. a Nexis4 card and transmission delays are measured. These measured delays are then compared with theoretically computed values.

IV. Proposed approach and preliminary results

The proposed solution is based on a prototyping tool. In the next paragraphs we present its main features and a preliminary case study.

A. *The prototyping tool ProNoC*

ProNoC (Prototype NoC) has been defined in [10] and is a prototyping tool which allows the design of many-core system on chips (MCSoC). It proposes an interface to generate the hardware code of a complete MCSoC. As shown in Figure 2, this MCSoC is composed of processing tiles (PT) interconnected using a NoC.

A processing tile (Figure 2b) is composed of different IP (Intellectual Property) cores interconnected by a Wishbone bus, an internal shared bus defined by OpenCores [11]. These IP cores include memory (RAM), processor, GPIO, timer, UART jtag and NI (Network Interface). The NI core allows the transmission of the data from (to) the tile to (from) the NoC.

ProNoC allows to generate a NoC. A NoC router is shown in Figure 2c. It is composed of Input and Output ports and a crossbar switch. The route of the packet is computed in LRC. In order to support quality of service for different messages, the NoC can use different virtual channels (VC). The management of these virtual channels is done by the SSA and VSA parts of the router.

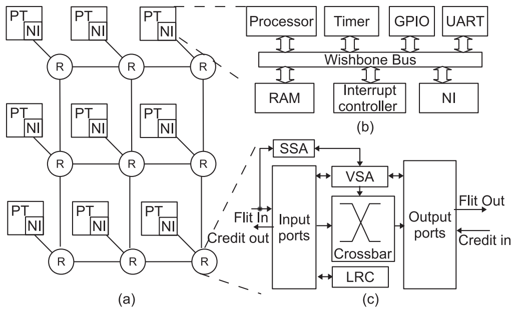


Fig. 2. Overview of an MCSoC obtained using ProNoC [10]

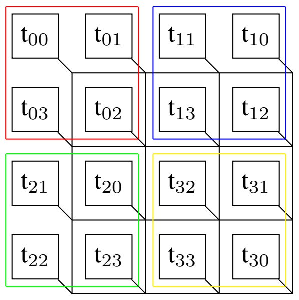


Fig. 3. Application mapping in the many-core (A0: upper left, A1: upper right, A2: bottom left, A3: bottom right)

ProNoC proposes a graphical interface in order to customize the MCSoC. It is possible to configure the definition of the PTs, *i.e.* what are the IP cores used for each PT. The parameterizitation of the NoC includes the number of virtual channels, the size of the buffers, the routing algorithm (XY, adaptative routing, ...), the switch arbitration (RRA, WRRA) and the topology (2D Mesh, Torus, ...).

The main goal of ProNoC is to provide the FPGA implementation of a fully functionnal MCSoC. Once all the MCSoC parts have been constructed, the tool generates the Verilog files that can be compiled for the FPGA. ProNoC tool provides also a NoC simulator to evaluate the performance of the NoC. Finally, ProNoC contains a NoC emulator. It provides a behavioural execution model of the MCSoc and the programming interface for processors cores.

B. *A preliminary case study*

We illustrate our solution based on ProNoC on a small case study. This case study is composed of 4 applications, named A_0 to A_3. Each application A_i is composed of 4 tasks $t_{i,0}$ to $t_{i,3}$. The communication graph for each application is given in Figure 4. The size of all the packets is 3 flits. One flit (flow digit) is 4 bytes.

As represented in Figure 3, the NoC is a 2D-mesh network. It uses XY routing algorithm. The flits are stored in input queues of the NoC routers. The size of these queues is 4 flits.

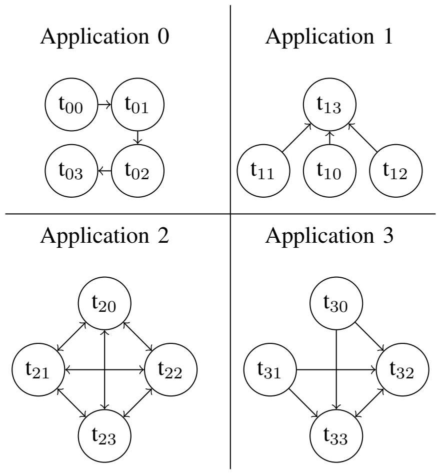


Fig. 4. Communication task graph for each application

The scheduling policy is round-robin. No virtual channel is used.

Each core of the tiles executes at most one task.

The applications are allocated on the many-core using SHiC strategy [8]. Figure 3 shows the resulting mapping.

Using the ProNoC emulator, the tasks are executed by the processor core of the tiles. Our methodology is as follow. (1) The task send the packet using `ni_transfert` function. (2) This function asks the transmission to the network interface of the tile. (3) A timestamp function is started. This function gets the global clock value. (4) When the data are received by the destination tile, the NI sends an interruption to the processing core executing the reception function. (5) The handler of this interruption reads the packet and gets the clock value.

Finally, the difference between the sending time and the receiving time is the global transmission delay obtained using the ProNoC tool. The results are given in Table I. We compare the results with theoretical ones obtained using the recursive calculus method described in [12]. The results show that the delays obtained using the implementation are much larger than the ones computed using a theoretical tool. It is due the overhead in the source and destination tiles.

V. Envisioned solution

The theoretical computation of delays takes into account the transmission between source and destination tiles. However it ignores the delays within these tiles. Preliminary results on the small use case show that these tile delays cannot be neglected. Thus they have to be precisely characterized. Therefore a precise analysis of the delays induced by tile architecture has to be conducted.

In the case study, the ProNoC emulator is used. The case study has to be extended with an implementation on an FPGA which can then be connected to an AFDX network, in order to obtain the architecture in Figure 1. We also have to consider more complex (realistic) case studies and different NoC features.

TABLE I
Transmission delays of each flow (in μs)

Flows	Practical delays	Theoretical delays
t_{00} to t_{01}	27.28	0.12
t_{01} to t_{02}	24.22	0.12
t_{02} to t_{03}	24.7	0.12
t_{10} to t_{13}	54.06	0.12
t_{11} to t_{13}	25.04	0.12
t_{12} to t_{13}	38.02	0.12
t_{20} to t_{21}	16.9	1
t_{21} to t_{20}	8.49	1
t_{21} to t_{22}	24.78	1
t_{22} to t_{21}	24.36	1.2
t_{20} to t_{22}	23.02	1.2
t_{22} to t_{20}	38.22	1.2
t_{20} to t_{23}	15.76	1.2
t_{21} to t_{23}	17.5	1.2
t_{22} to t_{23}	24.68	1.2
t_{31} to t_{32}	24.24	0.6
t_{31} to t_{33}	52.02	0.6
t_{32} to t_{33}	23.46	0.4
t_{30} to t_{33}	37.68	0.4
t_{30} to t_{32}	38.66	0.24

References

[1] DO-RTCA, "178c," *Software considerations in airborne systems and equipment certification*, 2011.

[2] Aeronautical Radio Inc. ARINC 664, *Aircraft Data Network, Part 7: Avionic Full Duplex Switched Ethernet (AFDX) Network*, 2005.

[3] V. Nélis, P. M. Yomsi, L. M. Pinho, J. C. Fonseca, M. Bertogna, E. Quiñones, R. Vargas, and A. Marongiu, "The Challenge of Time-Predictability in Modern Many-Core Architectures," in *14th Intl. Workshop on Worst-Case Execution Time Analysis*, Madrid, Spain, 2014, pp. 63–72.

[4] 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," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[5] L. Abdallah, J. Ermont, J. Scharbarg, and C. Fraboul, "Towards a mixed NoC/AFDX architecture for avionics applications," in *IEEE 13th International Workshop on Factory Communication Systems, WFCS*, 2017, pp. 1–10.

[6] J. Ermont, S. Mouysset, J. Scharbarg, and C. Fraboul, "Message scheduling to reduce AFDX jitter in a mixed NoC/AFDX architecture," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS*, 2018, pp. 234–242.

[7] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Proc. of the Conf. on Design, Automation & Test in Europe (DATE'14)*, 2014, pp. 97:1–97:6.

[8] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proc. of the 50th Annual Design Automation Conference*, 2013, p. 39.

[9] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "Reducing the contention experienced by real-time core-to-i/o flows over a tilera-like network on chip," in *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*. IEEE, 2016, pp. 86–96.

[10] A. Monemi, J. Wei Tang, M. Palesi, and M. N. Marsono, "Pronoc: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsystems*, vol. 54, pp. 60–74, October 2017.

[11] OpenCores, "WISHBONE System-on-Chip (SoC) interconnection architecture for portable ip cores." [Online]. Available: https://opencores.org/howto/wishbone

[12] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "Wormhole networks properties and their use for optimizing worst case delay analysis of many-cores," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Siegen, Germany, June 2015, pp. 59–68.

ResilienceP Analysis: Bounding Cache Persistence Reload Overhead for Set-Associative Caches

Syed Aftab Rashid, Geoffrey Nelissen, Eduardo Tovar
CISTER, ISEP, Polytechnic Institute of Porto, Portugal

I. Motivation and Introduction

In modern systems, the latency of an access to the main memory is much higher than the latency of an individual computation on the processor. Cache memory bridge this performance gap between the main memory and processor by holding frequently required data and instructions. Intuitively, caches are used to decrease average-case memory access latency; however, due to their limited capacity in comparison to main memory the use of caches can also cause large variations in the execution times of tasks. Due to limited space, not all data/instructions of all tasks can simultaneously reside in the cache. Hence, tasks may compete for cache space, with the execution of one task potentially evicting memory blocks previously loaded into the cache by other tasks. This may result in increasing the worse-case execution/response time (WCET/WCRT) of tasks depending on whether the instructions/data needed by the tasks are already present in the cache (i.e. cache hit) or not (i.e. cache miss).

The impact of caches on the WCET/WCRT of tasks is more evident under preemptive scheduling. In preemptive scheduling, tasks may suffer additional execution delays depending on the state of the cache, namely, *Cache Related Preemption Delays* (CRPDs) and *Cache Persistence Reload Overheads* (CPROs). CRPDs are delays suffered by the preempted tasks in reloading *useful cache blocks* (UCBs) (blocks cached before the preemption and potentially reused after) that were evicted from the cache during the execution of preempting tasks. On the other hand, CPROs result from the eviction of *persistent cache blocks* (PCBs) (memory blocks that, once loaded into cache by the task, will never be invalidated or evicted by the task itself and hence always available for fast access) due to the interleave or preemptive execution with other tasks. Many different approaches have been presented in the state-of-the-art (SoA) to bound CRPDs [2] and CPROs [3], [4]. However, most of these approaches focus on CRPD/CPRO calculation assuming a *direct-mapped* cache. In a direct-mapped cache, each cache set can hold at most one memory block and in case of a cache conflict between two tasks τ_i and τ_j, each cache block used by τ_j during its execution (i.e., called an evicting cache block (ECB)) can evict at most one UCB/PCB of τ_i and vice versa. However, today most processor architectures rely on *set-associative* caches. In set-associative caches, each cache set may hold more than one memory block depending on the number of available cache *ways* (also called cache *associativity*). Hence, one cache access by a task τ_j to the same cache set used by another task τ_i, may lead to multiple cache misses for τ_i (i.e., known as the cascading effect).

The few analyses in the literature that consider set-associative caches only focus on CRPD computation. However, it has been shown in recent works [3], [4] that only considering CRPDs for tasks scheduled under fixed-priority preemptive scheduling may result in largely pessimistic WCRT bounds and that the analyses that considers both CRPD and CPRO [3], [4] dominate the WCRT analyses that only consider CRPD [1]. Considering that the existing approaches for CPRO calculation only consider direct-mapped caches, in this paper we present different approaches to bound CPRO for set-associative caches. First, we present the PCB-ECB approach that considers PCBs of the task under analysis and ECBs of all other tasks in the system to calculate CPROs. We then introduce the resilienceP analysis that removes some of the pessimism in the PCB-ECB approach by considering the *resilience* of PCBs when calculating CPRO. Finally, we present a multi-set alike resilienceP analysis that considers variation in the resilience of PCBs over different job executions of a task in order to have an even tighter CPRO bound.

II. Notations and Background

We focus on set-associative caches using the Least-Recently-Used (LRU) replacement policy, i.e., on a cache miss the least recently used memory block within a cache set is evicted. The number of memory blocks each cache set can store is known as the number of ways or the associativity of the cache and is denoted by k. The total number of sets in the cache is denoted by cs. We use d_{mem} to denote the time needed to load one cache block from the main memory into the cache. As we consider fixed priority preemptive scheduling (FPPS), we use $\mathrm{hep}(i)$ to denote the set of tasks with priorities higher than or equal to that of τ_i (hence including τ_i).

Evicting and Useful Cache Blocks (ECBs and UCBs). All cache blocks used by the task during its execution are called ECBs [5] and an ECB m is also a UCB at a program point P, if m is cached at P and may be reused at program point Q that may be reached from P without eviction of m [6].

Cache Related Preemption Delay (CRPD). when a task τ_i is preempted by a higher priority task τ_j, ECBs of τ_j may evict UCBs of τ_i that are to be reloaded from the main memory after τ_i resumes. The additional execution time incurred by τ_i due to these extra cache reloads is termed as CRPD.

For set-associative caches, the resilience analysis [7] dominates all other method in the SoA to compute CRPD. It uses the notion of *resilience* to bound the CRPD of task τ_i due to preemptions by a higher priority task τ_j.

Resilience [7]. The Resilience of a memory block m at program point P is the largest l such that all possible next accesses to m (i) would be cache hits if there is no preemption, and (ii) would still be cache hits if there is a preemption at program point P with l cache accesses to the same cache set as m. The Resilience of a cache block m at a program point P is given by

$$res_P(m) = (k-1) - max\text{-}age_P(m) \tag{1}$$

where $max\text{-}age_P(m)$ is the *maximum* LRU-age of m at program point P, i.e., the maximum number of accesses to the same cache set as m from the last use of m before or at program point P to the next access to m after P [7]. In resilience analyses, the CPRD of task τ_i due to a single preemption by a higher priority task τ_j in a cache set s is given by $\gamma_{i,j}^{res,s}$;

$$\gamma_{i,j}^{res,s} = d_{mem} \times |UCB_i^s \setminus \{m_i | res(m_i) \geq |ECB_j^s|\}| \tag{2}$$

where $|UCB_i^s|$ and $|ECB_j^s|$ denote the number of UCBs/ECBs of τ_i and τ_j in cache set s. Effectively, the total CRPD over all cache sets $s \in cs$ is given by $\gamma_{i,j}^{res}$, where

$$\gamma_{i,j}^{res} = \sum_{s=0}^{cs} \gamma_{i,j}^{res,s} \tag{3}$$

Note that since the number of UCBs and the resilience is calculated for each program point, $\gamma_{i,j}^{res}$ is given by the program point that maximize Eq. (3) over all program points. Details on the formulation of Eq. (1)-(2) can be found in [7].

Persistent Cache Block (PCB) [3]. A memory block m_i of task τ_i is a PCB if, once loaded by τ_i, m_i will never be invalidated or evicted from the cache when τ_i executes in isolation.

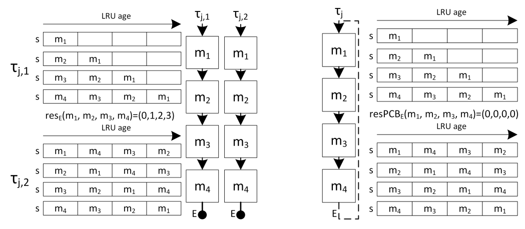
Cache Persistence Reload Overhead (CPRO) [3]. The CPRO of a task τ_j executing during the response time of a task τ_i is denoted by $\rho_{j,i}$ and is formally defined as the maximum memory reload overhead suffered by task τ_j due to evictions of its PCBs by tasks in $\text{hep}(i) \setminus \tau_j$.

III. PCB-ECB Approach for CPRO calculation

Existing approaches for CPRO calculation [3], [4] cannot be used as is for set-associative caches. This is due to the cascading effect in set-associative LRU caches which may result in evicting several PCBs of task τ_j due to a single ECB of tasks in $\text{hep}(i) \setminus \tau_j$. This effect does not happen in a direct-mapped cache where each ECB of tasks $\in \text{hep}(i) \setminus \tau_j$ can evict at most one PCB of τ_j. Before presenting our solution for set-associative caches, we first recall the CPRO-union approach [3] that calculates the CPRO $\rho_{j,i}^{dir}$ of a task τ_j executing during the response time of another task τ_i considering a direct-mapped cache,

$$\rho_{j,i}^{dir} = d_{mem} \times \left| PCB_j \cap \Big(\bigcup_{\forall \tau_k \in \text{hep}(i) \setminus \tau_j} ECB_k \Big) \right| \tag{4}$$

where PCB_j is the set of PCBs of τ_j and $\bigcup_{\forall \tau_k \in \text{hep}(i) \setminus \tau_j} ECB_k$ is the set of ECBs of all tasks $\in \text{hep}(i) \setminus \tau_j$. For a formal proof of Eq. (4) see [3]. It is proved in [3] that the set of PCBs of τ_j, i.e., PCB_j, upper bound the CPRO τ_j may suffer. We can easily extend that concept to set-associative caches by observing that the number of PCBs of τ_j in a cache set s, i.e., $|PCB_j^s|$, upper bounds the CPRO τ_j may suffer due to s. Let $CPRO_j^s$ denote the CPRO τ_j may suffer in cache set s, then $CPRO_j^s = |PCB_j^s|$, i.e., the total number of PCBs of τ_j in cache set s.


LRU age
$\tau_{j,1}$
$\tau_{j,2}$
$res_E(m_1, m_2, m_3, m_4)=(0,1,2,3)$
τ_j
$resPCB_E(m_1, m_2, m_3, m_4)=(0,0,0,0)$
E

(a) SoA resilience Analysis (b) ResilienceP analysis

Fig. 1: Overestimation in the SoA resilience analysis

From [3], we also know that the worst-case impact of all tasks in $\text{hep}(i) \setminus \tau_j$ on PCBs of τ_j is bounded by the set of all ECBs of all tasks in $\text{hep}(i) \setminus \tau_j$ (See Eq. 4). Hence, the worst-case impact of all task in $\text{hep}(i) \setminus \tau_j$ on PCBs of τ_j in a cache set s can be upper-bounded by $CPRO_{\text{hep}(i)\setminus\tau_j}^s$, where

$$CPRO_{\text{hep}(i)\setminus\tau_j}^s = \begin{cases} k & \text{if } \bigcup_{\forall \tau_k \in \text{hep}(i) \setminus \tau_j} ECB_k^s \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Consequently, the CPRO of task τ_j in cache set s is bounded by $\rho_{j,i}^{set,s}$, where

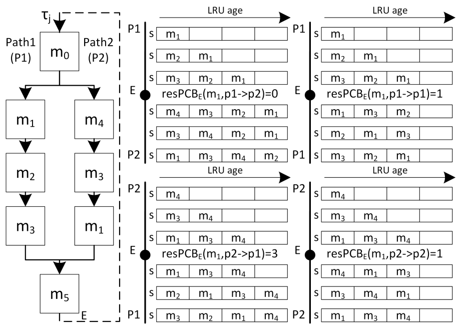
$$\rho_{j,i}^{set,s} = d_{mem} \times \min\left(CPRO_j^s, CPRO_{\text{hep}(i)\setminus\tau_j}^s \right) \tag{6}$$

and the total CPRO one job of τ_j may suffer during the response time of τ_i is thus given by $\rho_{j,i}^{set} = \sum_{s=0}^{cs} \rho_{j,i}^{set,s}$.

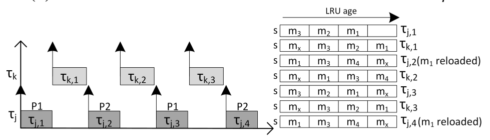
IV. ResilienceP Analysis

The PCB-ECB approach presented in Section III assumes that if one ECB of any task $\tau_k \in \text{hep}(i) \setminus \tau_j$ is mapped to a cache set s then all the PCBs of τ_j in s will be evicted. This assumption is safe but very pessimistic. Therefore, to have a tighter bound on the CPRO, in this section we determine the set of PCBs of task τ_j that may remain cached even after preemptions/executions of tasks $\in \text{hep}(i) \setminus \tau_j$ thanks to the resilience of τ_j's PCBs. However, we first note that the SoA resilience analysis [7] cannot be used as is to calculate the resilience of PCBs. To illustrate, see Fig. 1a showing the control-flow graph (CFG) and mapping of memory blocks of two jobs of task τ_j, i.e., $\tau_{j,1}, \tau_{j,2}$, in a 4-way set associative cache. We assume that $\{m_1, m_2, m_3, m_4\}$ are all PCBs of τ_j. Using the SoA resilience analysis that only considers the execution of one job of τ_j, i.e., $\tau_{j,1}$, it results that the resilience of PCB m_1, m_2, m_3 and m_4 is 0, 1, 2, and 3 respectively (see Fig. 1a). However, these resilience bounds are not sound considering that PCB m_1, m_2, m_3 and m_4 are reused only during the execution of the next job of τ_j, i.e., $\tau_{j,2}$. In fact, the maximum-age of all these PCBs across two jobs of τ_j is 3 which leads to a resilience of 0 for all the PCBs (See Eq. (1)).

The ResilienceP analysis accounts for the overestimated resilience of PCBs in the existing resilience analysis by calculating the maximum-age of PCBs over all job executions of τ_j. This is done by assuming that τ_j is cyclic, i.e., a loop between the end point E and start point S of τ_j (e.g., see Fig. 1b). The cyclic assumption ensures that the maximal number of different cache accesses between the last use of m_j in one job of τ_j and the first access of m_j in the next job of τ_j are considered when determining the maximum-age of m_j. Moreover, knowing that PCBs are calculated at task



(a) Variation in the resilience of PCBs of task τ_j



(b) Different job executions of τ_j and τ_k

Fig. 2: Highlighting the pessimism in ResilienceP analysis

level [3] in contrast to UCBs (calculated per program point) and the evictions of cache blocks in $UCB \cap PCB$ are already accounted for in the CRPD cost, the resilienceP analysis only calculates the maximum-age of PCBs at the end point E of a task τ_j using the same approach as proposed in [7]. Formally, under the resilienceP analysis the maximum-age of a PCB m_j is given by $max\text{-}age(m_j) = max\text{-}age_E(m_j)$, and the resilience of PCB m_j is given by $res_{PCB}(m_j) = (k-1) - max\text{-}age(m_j)$. Consequently, the total CPRO of one job of task τ_j executing during the response time of τ_i is bounded by $\rho^{res}_{j,i} = \sum_{s=1}^{cs} \rho^{res,s}_{i,j}$, where

$$\rho^{res,s}_{j,i} = d_{mem} \times \left| PCB^s_j \setminus \left\{ m_j | res_{PCB}(m_j) \geq \sum_{\forall \tau_k \in \text{hep}(i) \setminus \tau_j} |ECB^s_k| \right\} \right| \tag{7}$$

V. Multiset alike ResilienceP Analysis

The resilienceP analysis always considers the worst-case (i.e., minimum) resilience of PCBs for all jobs of τ_j that may execute in a time interval of length t. This is true if τ_j only has a single execution path as in Fig. 1b. However, if τ_j has multiple execution paths, the resilience of PCBs may vary depending on the actual execution paths taken by two successive jobs of τ_j. Therefore, always considering the minimum resilience of PCBs over all job executions of τ_j may overestimate the total CPRO τ_j may suffer. To illustrate this, see Fig. 2a that shows the CFG of a task τ_j with two execution paths and four possible execution flows between two jobs of τ_j, i.e., $p1 \rightarrow p2$, $p2 \rightarrow p1$, $p1 \rightarrow p1$ and $p2 \rightarrow p2$. The cache contents along each execution flow are also shown in Fig. 2a. We assume that all memory blocks of τ_j except m_0 and m_5 map to the same cache set s of a 4-way set-associative cache. For clarity, we only focus on PCB m_1.

We can see in Fig. 2a that the resilience of m_1 is minimum, i.e., $res_{PCB}(m_1) = 0$, if first job of τ_j follows path $p1$ and the next job follow path $p2$. Now consider the example schedule shown in Fig. 2b showing four jobs of τ_j along with three jobs of a task $\tau_k \in \text{hep}(i) \setminus \tau_j$ such that $ECB^s_k = \{m_x\}$. Fig. 2b also shows the contents of cache set s after the execution of every job of τ_j and τ_k.

TABLE I: CPRO-table for every PCB m_j of task τ_j

		Number of jobs of τ_j (J)			
		2	3	...	$\left\lceil \frac{t}{T_j} \right\rceil$
Disturbance (D)	1	$\min(1, x)$	$\min(2, x)$	...	$\min(\left\lceil \frac{t}{T_j} \right\rceil - 1, x)$
	2	$\min(1, x)$	$\min(2, x)$	...	$\min(\left\lceil \frac{t}{T_j} \right\rceil - 1, x)$
	...	...	...	...	...
	$\geq k$	1	2	...	$\left\lceil \frac{t}{T_j} \right\rceil - 1$

As the minimum resilience of m_1 is 0 and $|ECB^s_k| > res_{PCB}(m_1)$, the resilienceP analysis (i.e., Eq. (7)) implies that every time τ_k preempts τ_j or executes between two subsequent jobs of τ_j, m_1 will be evicted. This results in a CPRO of $d_{mem} \times 3$. However, we can see in Fig. 2b that this is not true. In fact even in the worst-case when we maximize jobs of τ_j following the execution flow with the minimum resilience (i.e., $p_1 \rightarrow p_2$), m_2 is evicted and reloaded only two times resulting in a CPRO of $d_{mem} \times 2$.

The multi-set alike ResilienceP analysis reduces the pessimism in the ResilienceP analysis by considering the variation in the resilience of PCBs across different job execution of a task τ_j. For each PCB m_j of τ_j we create a CPRO-table (See Table I) to determine how many times m_j can be evicted in an interval of length t considering a given disturbance D, i.e., the total number of ECBs of tasks in $\text{hep}(i) \setminus \tau_j$. Given the value of D and J, one entry in Table I (i.e., x) for a PCB m_j tells us how many times m_j may be evicted and must therefore be reloaded.

In future, we will investigate how to efficiently build Table I and evaluate our solutions.

Acknowledgments. This work was partially supported under PhD grant SFRH/BD/119150/2016, by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project POCI-01-0145-FEDER-029119 (PREFECT); by the European Union through the Clean Sky 2 Joint Undertaking, under the H2020 Framework Programme (H2020-CS2-CFP08-2018-01), grant agreement nr. 832011 (THERMAC).

References

[1] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

[2] M. Lv, N. Guan, J. Reineke, R. Wilhelm, and W. Yi, "A survey on static cache analysis for real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 3, no. 1, pp. 05–1, 2016.

[3] S. A. Rashid, G. Nelissen, D. Hardy, B. Akesson, I. Puaut, and E. Tovar, "Cache-persistence-aware response-time analysis for fixed-priority pre-emptive systems," in *ECRTS*, 2016, pp. 262–272.

[4] S. A. Rashid, G. Nelissen, S. Altmeyer, R. I. Davis, and E. Tovar, "Integrated analysis of cache related preemption delays and cache persistence reload overheads," in *RTSS*. IEEE, 2017, pp. 188–198.

[5] H. Tomiyama and N. D. Dutt, "Program path analysis to bound cache-related preemption delay in preemptive real-time systems," in *CODES*, 2000, pp. 67–71.

[6] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *Computers, IEEE Transactions on*, vol. 47, no. 6, pp. 700–713, 1998.

[7] S. Altmeyer, C. Maiza, and J. Reineke, "Resilience analysis: Tightening the crpd bound for set-associative caches," in *LCTES*. ACM, 2010, pp. 153–162.

Towards Real-time Self-adaptation Using a Verification Mechanism

Hiroyuki Nakagawa, Hiroki Tsuda, Tatsuhiro Tsuchiya
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, 565-0871 Japan
Email: {nakagawa, h-tsuda, t-tutiya}@ist.osaka-u.ac.jp

***Abstract*—Self-adaptation is desirable as an essential feature for dealing with environmental changes. Several studies are conducted on the development of self-adaptive systems; however, most of the studies do not pay attention to time constraints that the systems should satisfy. Even though some studies verify the time constraints at design time, to our knowledge, no middleware for self-adaptive systems religiously deal with time constraints at runtime. This paper describes the vision and the current state of our middleware that provides a modeling method, scheduling mechanism, and programming framework for self-adaptive systems that should behave under time constraints.**

I. Introduction

Systems in the real world have to interact with their environments. Because of an increase in the diversity of the environments and the life-span of systems, many of these systems also have to adapt to their environmental changes that are not considered at the design time. Self-adaptation is regarded as an essential feature for dealing with such environmental changes. This feature is studied in the software engineering field [1].

Many of systems in the real world have time constraints, that is, they are a kind of real-time systems [2]. Since an increase of processing time caused by an unexpected environmental change may result in a delay of behavior and a system failure, it is desirable to establish a design methodology to implement real-time self-adaptive systems. While most of the real-time systems work in highly predictable environments as hard real-time systems, our study aims to construct a self-adaptation middleware that behaves as a soft real-time system. Some studies, such as [3] [4] [5], have focused on adaptive real-time systems. As in these studies, we assume that tasks are sporadic. A sporadic task has a worst-case execution time. However, since the worst-case execution time, which is sometimes extended by unexpected environmental changes that are not part of the considered operational domain, is not determined at design time, we acquire the time from the monitoring results at run time. While existing algorithms schedule fixed tasks that have multiple service levels by determining suitable service levels at runtime, our study deals with the variability of multiple candidate tasks, that is, the system has to determine which tasks should be executed or not. In self-adaptive systems, configurations define which tasks can be executed on the system.

Studies on self-adaptive systems such as [1] and [6] pointed out the importance of dynamic verification; however, there is no middleware that provides dynamic verification mechanisms of time constraints. This paper presents our approach to designing self-adaptation middleware that can deal with time constraints. This paper also describes the current state of our study.

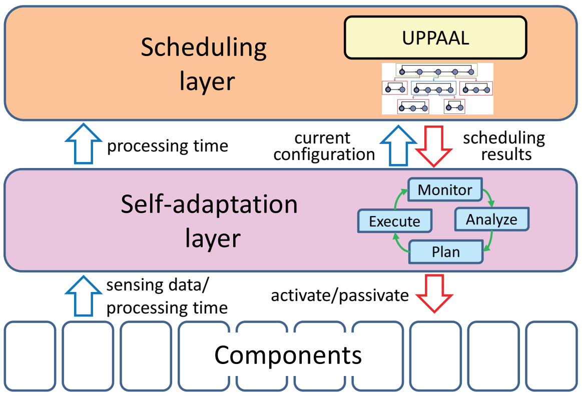


Fig. 1. The overview of our self-adaptation middleware that can handle time constraints.

II. Approach

Figure 1 illustrates the overview of our self-adaptation middleware that handles time constraints. We assume that the target system is constructed as a component-based system [7], that is, the system is placed into separate components which provide their functions and have their interfaces. Our middleware adds two layers on the components that provide system functions. The lower layer, called *self-adaptation layer*, realizes self-adaptation. Self-adaptation mechanisms such as those based on the MAPE (Monitor-Analyze-Plan-Execute) loop [8] mechanism can be located in this layer. This layer monitors components and collects sensing data and processing time from the components. If the layer determines it must adapt to environmental changes from the collected data, the layer changes the configuration by controlling (activating/deactivating) the components. A configuration defines the elements that a system is composed of. Therefore, the configuration defines which tasks can be executed on the current system.

The upper layer, called *the scheduling layer*, on the other hand, dynamically checks whether the current configuration satisfies time constraints. The layer verifies the adequacy of the current configuration using input processing time and the model that represents the configuration. The layer sends the verification results to the lower layer.

In such a system, we have to consider the following three parameters related to time constraints:

- t_{exec}: the time necessary to complete given tasks.
- t_{calc}: the time necessary to calculate t_{exec}.
- t_{find}: the time to find a suitable configuration for the current environment.

When the scheduling layer predicates a time constraint violation, that is, when the layer concludes that the scheduled tasks, which take the time t_{exec}, will miss the deadline, the system decides to start an adaptation. The scheduling layer searches for a new configuration that satisfies all of time constraints. When the verification passes all the time constraints, the layer sends the suitable configuration as a verification result to the self-adaptation layer, and then the self-adaptation layer changes the states of the components to implement adaptation.

Here, we define our system model. A system has one or more sporadic functions. We denote the *i*-th function as F_i. To provide a function, set of tasks should be executed. Some functions have alternative task sets to deal with environmental changes. Generally, a function has one or more task sets. We denote the *j*-th task set of the function F_i as Ts_{ij}. The *k*-th task in Ts_{ij} is denoted as T_{ijk}. Every task has a estimated worst-case execution time, denoted as $\mathbf{e}(T_{ijk})$. Like the assumptions presented by Block et al. [3], the worst-case execution times are not assumed. Each task set Ts_{ij} has a service level.

III. Scheduling Layer

The input of the scheduling layer is a current system configuration, which we can model as a set of task sets for all functions $\Sigma_i Ts_{ij}$. The scheduling layer returns *yes* if the current system configuration satisfies all of the time constraints. If the current system configuration cannot satisfy the time constraints, the scheduling layer returns *no* and generates a new systems configuration that satisfies the time constraints, i.e., a new set of task sets $\Sigma_i Ts_{ij'}$. If two or more task sets for the same functions can be the candidates of the new configuration, the one that has a higher service level should be selected.

To implement the scheduling layer, we use UPPAAL. UPPAAL is a tool box for modeling, simulation, and verification of real-time systems [9]. The verifier of UPPAAL checks whether the timed automata which represent system behaviors satisfy time constraints. Since our verification mechanism has to verify time constraints at run time, the verification mechanism dynamically generates an intermediate file that represents a network of timed automata and executes the UPPAAL verifier at runtime.

Although model checking techniques may suffer from the state explosion problem, our approach uses UPPAAL not to verify the violation of time constraints under all of the possible situations but to find a suitable configuration under the current environment, like one for using a model checking mechanism for the planning [10]. We have observed that our scheduling engine can output scheduling results well ahead of time in our examples.

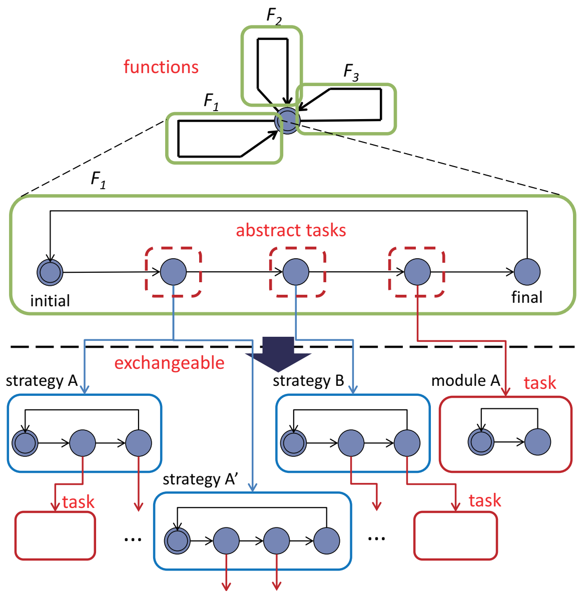


Fig. 2. Hierarchy of automata. The root automaton represents state transitions of functions. States in functions represent abstract tasks, which are modeled in automata of strategies or modules. A module corresponds to a task.

Figure 2 shows the automata that the engine accepts as a model that represents system behaviors. The automata form a hierarchical structure. The upper part of Figure 2 represents the perspective structure of the root automaton. The root automaton is constructed such that it can represent the execution of all functions of the system. Each function consists of one or more abstract tasks. These abstract tasks are represented as locations (states) in the root automaton. An abstract task is decomposed into more specific and exchangeable automata. A task that can be processed by one component is called a *module* and the one that has to be processed by more than one component is called a *strategy*. The decomposition is continued until an automaton can be processed by a single component, that is, until all of the tasks are represented as modules.

Such a modeling allows us to calculate the processing time of a function by summing up processing times of relevant *modules*. These processing times are recalculated at runtime to update the attributes of automata. To represent the hierarchal task structure, a parent automaton and child automata are synchronized by a synchronizing channel. We use the UPPAAL's channel for this purpose.

When a time constraint violation is detected in a verification, the verification mechanism searches for a suitable configuration by replacing a module or strategy with an alternative one. The verification process continues until an updated configuration, i.e., a set of automata, satisfies all the time constraints.

IV. Preliminary Evaluation

We conducted a preliminary case study using simulation. We designed a self-adaptive UAV (Unmanned Aerial Vehicle)

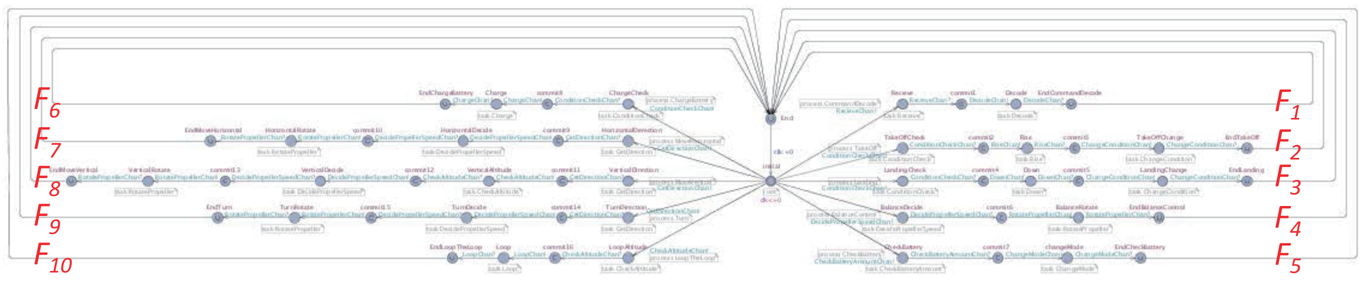


Fig. 3. The root automaton of a UAV controller system example. Loops labeled "F_x" correspond to individual functions. This automaton is synchronized with other automata that represent strategies and modules.

controller based on our approach. We defined 10 functions for the controller, such as take off, go forward, and turn left (right), and decomposed them to define 4 strategies and 17 modules. Figure 3 illustrates the root automaton of the controller. We also defined time constraints, such as "*E<> EndTakeOff and clk <= 100*" for the function F_2 in Figure 3.

We constructed a mock-up self-adaptation mechanism and components. The mock-up self-adaptation mechanism in the self-adaptation layer was implemented on our programming framework [11], which provides a general MAPE loop mechanism, i.e., features for monitor, analysis, planing, and execution. It sends processing time of components to the scheduling layer. The mock-up components also print out logs if their states are changed. We defined time constraints for each function and initial processing time of each task. We simulated two scenarios and observed that time constraint violations triggered an adaptation: when we injected a fault or a large processing time, our framework detected violations and started to search for a new configuration. In both scenarios, the verifications were correctly finished and the results, i.e., new configurations, were implemented by changing component states. We observed that the scheduling layer found suitable configurations in these case studies; however, we have to evaluate the performance of our approach using larger examples.

V. Conclusion and Future Work

This paper described our approach to dealing with adaptation under time constraints. Our approach uses a self-adaptive mechanism as the basis and assembles it with a scheduling engine that uses a dynamic verification mechanism, which can deal with time constraints. In this paper, we mainly explained a scheduling mechanism that uses a verification technique. The scheduling mechanism dynamically uses UPPAAL not only to verify time constraints in the current configuration but also to find a suitable configuration if it is required.

One of the main remaining parts of our study is a refinement of the scheduling mechanism. The current mechanism is optimistic about the time for scheduling, that is $t_{calc} + t_{find}$. Experimental results indicate that the scheduling requires sufficiently short time; however the scheduling layer has to manage time strictly. We have to evaluate the scalability of the mechanism by conducting experiments with a large number of tasks and their candidate tasks for adaptation. Note that if we construct the middleware to make the scheduling and self-adaptation layers work in parallel, we can manage t_{exec} and $t_{calc} + t_{find}$ independently.

We may need to extend the scheduling engine. The current engine determines a suitable configuration under the assumption that tasks are executed serially. If a system permits a parallel execution of tasks, the scheduling result would be improved. In this case, we will embed an existing scheduling algorithm that can deal with a parallel scheduling into the mechanism. We also plan to enhance the self-adaptation middleware. In order to be able to consider various delicate time constraints related to t_{exec}, t_{calc}, and t_{find}, not only at design time but also at runtime, we will implement some useful APIs that directly interact with the scheduling layer.

References

[1] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, and et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Dagstuhl Seminar 10431*, 2011.

[2] E.-R. Olderog and H. Dierks, *Real-Time Systems: Formal Specification and Automatic Verification*, 1st ed. New York, NY, USA: Cambridge University Press, 2008.

[3] A. Block, B. Brandenburg, J. H. Anderson, and S. Quint, "An adaptive framework for multiprocessor real-time system," in *Proc. of the 2008 Euromicro Conference on Real-Time Systems (ECRTS 2008)*, 2008, pp. 23–33.

[4] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley, "Performance specifications and metrics for adaptive real-time systems," in *Proc. of the 21st IEEE Conference on Real-time Systems Symposium (RTSS 2000)*. IEEE CS, 2000, pp. 13–23.

[5] T. F. Atdelzater, E. M. Atkins, and K. G. Shin, "Qos negotiation in real-time systems and its application to automated flight control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170–1183, Nov 2000.

[6] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time," *IEEE Trans. on Software Engineering*, vol. 42, no. 1, pp. 75–99, Jan 2016.

[7] D. Garlan, R. T. Monroe, and D. Wile, "Foundations of component-based systems," G. T. Leavens and M. Sitaraman, Eds. Cambridge University Press, 2000, pp. 47–67.

[8] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[9] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a Nutshell," 1997.

[10] F. Giunchiglia and P. Traverso, "Planning as model checking," in *Proc. of the 5th European Conference on Planning: Recent Advances in AI Planning (ECP '99)*. Springer-Verlag, 1999, pp. 1–20.

[11] H. Tsuda, H. Nakagawa, and T. Tsuchiya, "Towards self-adaptation on real-world hardware: A preliminary lightweight programming framework," in *Proc. of the IEEE 9th Inter. Conf. on Self-Adaptive and Self-Organizing Systems (SASO'15)*, Sept 2015, pp. 176–177.

Towards Robust and Cost-Effective Critical Real-Time Systems under Thermal-Aware Design

Javier Pérez Rodríguez and Patrick Meumeu Yomsi
CISTER Research Centre, ISEP, Polytechnic Institute of Porto, Portugal
Email: {perez; pmy}@isep.ipp.pt

***Abstract*—The advent of multi-core platforms in critical real-time domains such as the avionics, automotive and railways to achieve higher and higher computing performance has turned the view on thermal concerns of the underlying chip die while it is still mandatory to meet all the temporal constraints. As a matter of fact, high chip temperature may not only degrade system performance and reliability, but it may also damage the chip permanently. In this paper, we propose a methodology to address this problem, based on fixed task-to-core mapping and per-core analysis to derive a sound system model without feedback loop. To this end, it is important to have a better and deeper understanding of the existing thermal models in the literature. This is the main contribution of this research.**

1. Introduction

For several decades now, critical real-time systems have consistently and continuously been under the spotlights of experts from both industry and academia. This is because they exposed stringent functional and non-functional requirements that have to be met, otherwise catastrophic consequences may occur. In general, these systems are modeled by using a finite set of recurrent tasks to be executed on a targeted hardware platform (e.g., the Intel Core2 from Intel, the 4-core Arm V7 Raspberry Pi 3 B and B+ from Arm; the TMS-320-C6678 from TI, the Tile-Gx3000 from Tilera, and the MPPA-256 architecture from Kalray) and each task commonly consists of a potentially infinite number of instances (jobs). Each job is characterized by four parameters: (1) a release time, which defines the instant time at which the job becomes available and ready for execution; (2) a worst-case execution time, which defines an upper-bound on the actual execution time of the job on the targeted platform; (3) a minimum inter-arrival time, which defines how frequent is the release of a new job[1]; and finally (4) a deadline, which defines a time window, from the release, wherein the job has to complete its execution. While each task's functional correctness is important for these systems, the time at which the result is produced is also central. To this end purpose, several factors have to be considered at the system design time. Examples include the task interactions, concurrency, and interference at the software level; and the mechanisms governing the execution of the tasks (preferably with a great level of details) at the hardware level. To date, an entire body of knowledge, techniques and methodologies have been proposed in the literature on the topic, some of which are now mature, especially for single core platforms. However, new challenges arise almost on a daily basis. This is due to the ever growing complexity and computational demand of the applications at the software level and/or the non-disclosure of valuable and detailed information on the targeted platform by the hardware vendors. Despites these noticeable limitations and the constant necessity for miniaturization of the emerging hardware components, we have been witnessing the integration of more and more processing elements in smaller silicon areas in order to achieve better performance. As a matter of fact, the integration scale has been doubling every three years [1], [2]. From a software viewpoint, this has resulted in forcing the processor to execute workloads at high frequencies most of the time. Hence, (i) the necessity for hardware miniaturization on one side; and (ii) the ever increasing computational demand of the applications on the other side, put together, have highlighted a serious problem: the soaring power dissipation of the integrated circuits, which in turn translates in temperature dissipation. Obviously, high temperatures create a number of problems, because transistors may fail to switch properly and therefore can lead to transient and/or permanent errors for the entire system. Specifically, an increment in the temperature until an uncontrolled value can affect drastically the runtime behavior of the tasks, and also the platform. This phenomenon holds true irrespective of hosting the execution of the tasks on a platform with a single or several cores. According to Borkar [1], the price for cooling down a watt of temperature in a processor is about $\$1 - \$3$ or more. Consequently, this opens a broad avenue for research for the design of cost-effective and more robust critical real-time systems in critical real-time domains such as the avionics, automotive and railways. To the best of our knowledge, the thermal problem for critical real-time systems has been addressed in the literature by either switching off some core(s) [3], [4] or by re-scaling the cores speed [5], [6], [7], [8]. Roughly speaking, this means that the thermal management of the platform is handled

1. If the frequency is constant, then the task is said to be periodic.

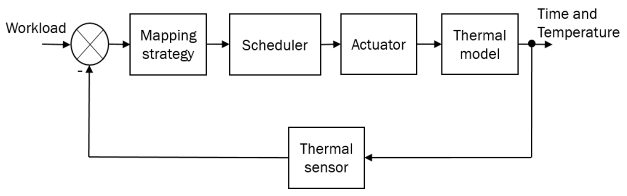


Figure 1. Classical control block diagram for thermal management.

by using a feedback control block diagram as illustrated in Figure 1. Here, action is taken only when the reported temperature by the thermal sensor rises above a predefined threshold. Below the threshold, no specific optimization and/or workload distribution strategy is used to maintain both the temporal and thermal behavior of the system. As a consequence, the time spent in cooling down the system may cause temporal changes in the original tasks schedule and then jeopardize the schedulability. Furthermore, not all platforms can support speed re-scalability, unfortunately. In this work, we argue and believe that the problem must be addressed from a different angle. In our opinion, it is possible to create a new "correct-by-construction" framework, preferably unique, wherein we model under the same umbrella both the temporal and thermal "on-core" and "un-core" activities for each processing element. For a given mapping, the on-core model will capture the activity (temporal and thermal) of the core under analysis, whereas the un-core model will capture the interference (temporal and thermal) imposed by other processing elements and share resources. As a result, it will become easier to derive an analysis that predicts the run-time behavior of the entire system without any need of a feedback loop (see Figure 2).

2. Problem statement

Nowadays, multi-core platforms are pervasive in numerous critical real-time systems due to the enormous computing capabilities they offer. While meeting domain-specific standards' requirements (e.g., the ARINC-653 and DO-178C in the avionics; and the ISO-26262 in the automotive) in terms of temporal requirements, our main objective is to address the following question. As the adoption of a multi-core platform exposes the underlying chip die to several heating sources and the temperature of each core can interfere with the thermal dissipation of the neighboring cores, how to adapt and/or design a robust and cost-effective thermal model of the platform that can easily be coupled with the adopted temporal model of the application so as to make it possible for the system designer to capture in an accurate manner both the chip-wide and the localized thermal behaviors of the system at run-time? The derived thermal model, associated with the temporal model, will allow for a sound thermal-aware schedulability analysis for the entire system.

3. Overview of existing thermal models

Before going into details, it is worth mentioning that power models as described in the literature have failed to manage temperature, despite the well-known duality between heat transfer and electrical phenomena. Consequently, to pave the way towards a convincing solution to the aforementioned problem, an educated strategy commands us to proceed by exploring all the thermal models that have been proposed in the literature in first place. In this regard, only two thermal models have been developed to the best of our knowledge: (1) a *coarse-grained* model referred to as TEMPEST [9], which uses a Resistance-Capacitance (RC) *parallel circuit* representation; and (2) a *fine-grained* model referred to as HotSpot [10], [11], which uses a RC *serial circuit* representation. Below, we briefly discuss their advantages and disadvantages.

- ▷ **TEMPEST.** This model has been proposed by Dhodapkar et al. [9]. Here, temperature is tracked only at a macro-architectural level, i.e., at the chip level. Consequently, this model is not flexible and allows only for chip-wide thermal-aware techniques such as Dynamic Voltage and Frequency Scaling (DVFS) [12] and Fetch Toggling [13] for reducing the processor peak temperatures. On the positive side, TEMPEST is easily portable to new hardware architectures and simple to implement because it makes it possible to safely bound the temperature of the underlying platform irrespective of the localization of eventual hotspots and it is agnostic to the hardware run-time mechanisms. However, it has been proven that localized heating occurs much faster than chip-wide. In this case, chip-wide treatments are too conservative, unfortunately.
- ▷ **HotSpot.** This model has been proposed by Skadron et al. [10]. In contrast to TEMPEST, temperature is tracked at the granularity of individual micro-architectural units and the equivalent RC circuits have at least one node for each unit. As such, this model allows for the detection of hotspots and to promptly activate a thermal response. The system designer can operate at block-level on the underlying platform or even below, and so, he can capture and handle the effects of hotspots more accurately. However, the model is way less portable and much more complex to implement as it requires a detailed understanding of the mechanisms governing the run-time behavior of most hardware components (e.g., branch predictor, load-store queue, D-cache etc.). In addition, the sampling rate at which the detection of new hotspots is performed have to be closely scrutinized as it plays a central role here, unfortunately.

4. Envisioned approach

From the discussion conducted in Section 3, it follows that the HotSpot model exposes better features than TEMPEST for the design of an accurate thermal-aware management technique upon a multi-core platform. However, the right level of abstraction that would make it unnecessary to model all the micro-architectural units and still achieve a sound analysis is missing. To fill this gap, we plan to proceed in three phases as follows.

First, we plan to revisit the task-to-core mapping strategies available in the literature in order to take into account the thermal profile of each task in our mapping procedure. During this phase, we will promote mapping strategies for which the increase of the overall platform temperature is as minimum as possible. This will be achieved by using a stochastic-based approach for example. Second, for the resulting mapping, we will adopt a per-core analysis and build a unique "correct-by-construction" framework wherein we model both the temporal and thermal "on-core" and "un-core" activities for each processing element. Our combined system model

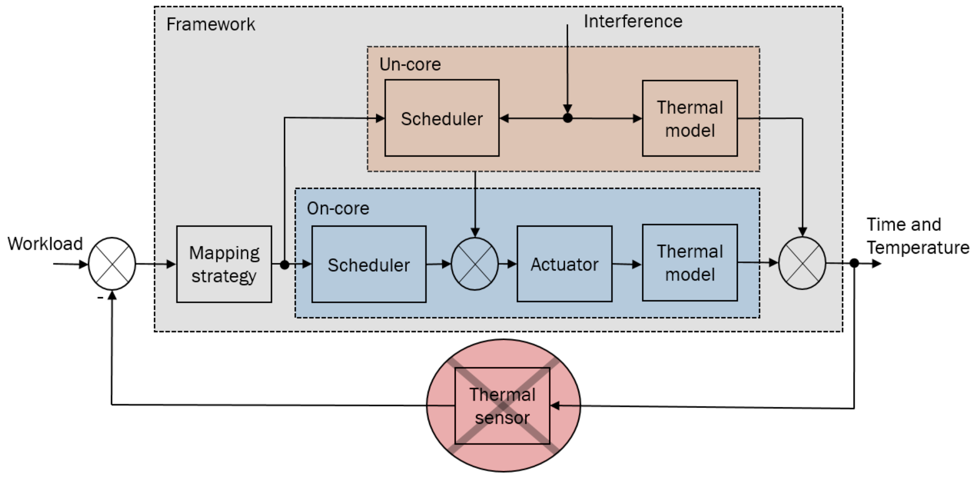


Figure 2. Envisioned control block diagram for thermal management.

will allow us not only to guarantee soundness, but also to optimize for thermal efficiency and thus costs. The on-core model will capture the activity (temporal and thermal) of the core under analysis, whereas the un-core model will capture the interference (temporal and thermal) imposed by other processing elements and share resources. Finally, we will derive an analysis that predicts the run-time behavior of the entire system from a temporal and thermal viewpoint without any need of a feedback loop (see Figure 2). Note that in presence of such a feedback loop, the imprecision of the thermal sensor may lead to an optimistic, if not wrong, analysis. Consequently, our proposed approach will rely on strong mathematical foundations based on an open control loop with perturbations for each core.

5. Conclusion

In this work, we detailed our research roadmap for the design of a robust and cost-effective critical real-time system under thermal-aware design. We revisited the thermal models available in the literature and briefly discussed their advantages and disadvantages. We reached the conclusion that the HotSpot thermal model exposes the most promising features to help us meet our objectives both from a temporal and thermal viewpoint, but it requires some adjustments. The main challenge is to find the correct level of abstraction that would make it unnecessary to model the thermal behavior of all micro-architectural units. Finally, we elaborated on the directions that we are planning to explore to derive our thermal-aware schedulability analysis for multi-core platforms.

Acknowledgment

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); by the European Union through the Clean Sky 2 Joint Undertaking, under the H2020 Framework Programme (H2020-CS2-CFP08-2018-01), grant agreement nr. 832011 (THERMAC).

References

[1] S. Borkar, “Design challenges of technology scaling,” *IEEE Micro*, vol. 19, no. 4, pp. 23–29, July 1999.

[2] R. Mahajan, “Thermal management of CPUs: A perspective on trends, needs and opportunities.” in *Proceedings on 8th International Workshop on THERMal INvestigations of ICs and Systems (THERMINIC)*, 10 2002.

[3] P. Kumar and L. Thiele, “Thermally optimal stop-go scheduling of task graphs with real-time constraints,” in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, Jan 2011, pp. 123–128.

[4] Y. Chandarli, N. Fisher, and D. Masson, “Response time analysis for thermal-aware real-time systems under fixed-priority scheduling,” in *IEEE 18th International Symposium on Real-Time Distributed Computing*, April 2015, pp. 84–93.

[5] N. Bansal, T. Kimbrel, and K. Pruhs, “Dynamic speed scaling to manage energy and temperature,” in *45th Annual IEEE Symposium on Foundations of Computer Science*, Oct 2004, pp. 520–529.

[6] N. Bansal and K. Pruhs, “Speed scaling to manage temperature,” in *STACS*, V. Diekert and B. Durand, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 460–471.

[7] S. Wang, Y. Ahn, and R. Bettati, “Schedulability analysis in hard real-time systems under thermal constraints,” *Real-Time Systems*, vol. 46, pp. 160–188, 10 2010.

[8] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos, “Feedback thermal control of real-time systems on multicore processors,” in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ser. EMSOFT. New York, NY, USA: ACM, 2012, pp. 113–122.

[9] A. Dhodapkar, C. How Lim, G. Cai, and R. Daasch, “TEM2P2EST: A Thermal Enabled Multi-model Power/Performance ESTimator,” in *Power-Aware Computer Systems*, 06 2001, pp. 112–125.

[10] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture: Extended discussion and results,” in *In Proceedings of the 30th Annual International Symposium on Computer Architecture*, 07 2003, pp. 2–13.

[11] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy, “Compact thermal modeling for temperature-aware design,” in *41st Design Automation Conference.*, July 2004, pp. 878–883.

[12] D. R. Sulaiman, M. Ibrahim, and I. Hamarash, “Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction,” *4th International Conference on Electrical and Electronics Engineering*, 12 2005.

[13] K. Skadron, T. Abdelzaher, and M. R. Stan, “Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management,” in *Proceedings Eighth International Symposium on High Performance Computer Architecture*, Feb 2002, pp. 17–28.