I/O contention aware mapping of multi-criticalities real-time applications over many-core architectures

Laure Abdallah and Mathieu Jan
CEA, LIST, Embedded Real Time Systems Laboratory
F-91191 Gif-sur-Yvette, France
Email: Firstname.Lastname@cea.fr

Jérôme Ermont and Christian Fraboul
IRIT INP-ENSEEIHT, Université de Toulouse
F-31000 Toulouse, France
Email: Firstname.Lastname@enseeiht.fr

***Abstract*—Many-core architectures are more promising hardware to design real-time systems than multi-core systems as they should enable an easier mastered integration of a higher number of applications, potentially of different level of criticalities. However, the worst-case behavior of the Network-on-Chip (NoC) for both inter-core and core-to-Input/Output[1] (I/O) communications of critical applications must be established. The mapping over the NoC of both critical and non-critical applications has an impact on the network contention these critical communications exhibit. So far, all existing mapping strategies have focused on inter-core communications. However, many-cores in embedded real-time systems can be integrated within backbone Ethernet networks, as they mostly provide Ethernet controllers as I/O interfaces. In this work, we first show that Ethernet packets can be dropped due to an internal congestion in the NoC, if these core-to-I/O communications are not taken into account while mapping applications. To solve this issue, we show on an avionic case study the benefits of the core-to-I/O contention-aware mapping strategy we propose.**

I. Introduction

The continuous need in increased computational power has fueled the on-going move to multi-core architectures in hard real-time systems. However, multi-core architectures are based on complex hardware mechanisms, such as for instance advanced branch predictors whose temporal behavior is difficult to master. Many-core architectures are instead based on simpler cores, so the timing predictability of cores are thus easier to analyze [10]. Besides, they should enable the safe simultaneous integration of both critical and non-critical applications [2]. Many-cores are thus promising hardware to host such a mix of real-time applications with different levels of criticalities. Note that we consider only two levels of criticalities in the remainder of this work: critical and non critical. The main challenge however lies in the ability to analyze the Worst-Case Traversal Time (WCTT) of critical flows exchanged within the Network-on-Chip (NoC). How hard real-time tasks that generate these critical flows, but also the non critical tasks, are mapped within cores, is therefore of utmost importance to control the contention over the NoC and thus the WCTT of flows. In the remainder of this paper, we simply say that we map flows over a NoC to avoid linking flows to tasks.

The efficiency of a mapping strategy over a NoC can be evaluated using different performance metrics. Hard-real time applications rely on the latency metric, as the goal is to minimize the WCTT of flows. Several contention aware mapping strategies (for instance [3], [11], [14]) have thus been proposed but for inter-core communications only. To the best of our knowledge, none consider communications between cores and with I/O interfaces, that we call core-to-I/O communications[1]. However, many-cores mainly provide DDR and Ethernet controllers only as I/O interfaces. For instance, Tilera [12] provides 3 Ethernet and 4 DDR controllers, while MPPA [5] from Kalray provides 8 Ethernet and 2 DDR controllers. Theses many-cores are thus more tailored to host embedded applications whose I/O data are exchanged using Ethernet packets. *In embedded real-time systems, many-cores can be used as processing elements within a backbone Ethernet network*, such as AFDX for the avionic domain or Ethernet AVB in the automotive field.

Mapping strategies for many-cores should therefore also take into consideration core-to-I/O communications, in addition to other core-to-core and core-to-memory communications. To demonstrate this strong requirement, the first contribution of this paper is to show that Ethernet packets can be dropped due to an NoC congestion, if I/O requirements are not taken into account when mapping applications. To this end, we rely on a case study from the avionic domain. It is made of a critical Full Authority Digital Engine (FADEC) application and a non-critical Health Monitoring (HM) application of the engine, used for recognizing incipient failure conditions. We thus propose an approach to map critical and non critical real-time applications over many-cores that reduces the WCTT of core-to-I/O communications. It is based on an existing strategy but in which we treat core-to-I/O communications as first class citizen. Our algorithm first assigns for each application to map a region within the NoC. Then, each task of an application is mapped within its region, so that the paths used by core-to-I/O communications from the Ethernet controller exhibit the lowest contention possible. We show for two variants of our case study that our algorithm successfully find a mapping that avoids Ethernet packets, whose payload are making the core-to-I/O communications, to be dropped. This demonstrates the benefits of our proposal compared to a state of the art mapping strategy that fails to do so.

II. Problem formulation

To illustrate the problem we address, we use an avionic case study made of a FADEC and an HM application. For the FADEC, 1270 bytes of sensors data from the engine are received by an Ethernet interface. These data are then divided and distributed to 6 tasks, noted t_{f0} to t_{f5}. These 6 tasks exchange 211 bytes of data between them. All these tasks also send 211 bytes of data to a last task noted t_{f6}. Then, t_{f6} stores 110 bytes within a DDR interface and sends 64 bytes of actuators data through an Ethernet interface. On the other hand,

[1] We use this term for both core communications from or to I/O interfaces.

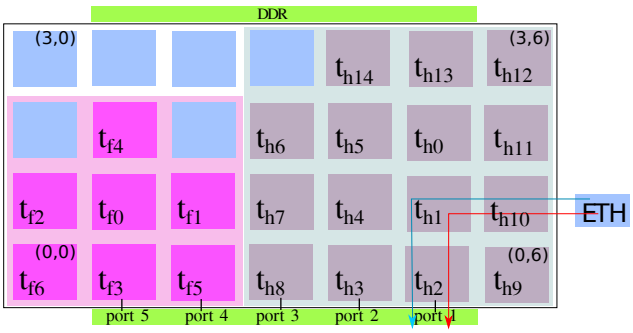

Fig. 1. Arbitrary mapping of FADEC and HM applications.

the HM application continuously receives through an Ethernet interface, a set of frames representing data to be processed in order to anticipate engine failures. The size of a frame is 130 KBytes and a set is made of 30 frames. When a set of frames is received, every two frames are assigned to a different task amongst 15 tasks, noted t_{h0} to t_{h14}. When the processing takes place, task t_{hi} also sends 112 bytes of data to t_{hi+1}, with $i \in [0, 14]$. Finally, all these tasks, finish their processing by storing their frames into the DDR.

Figure 1 shows an arbitrary mapping of these 2 applications over a 7×4 mesh NoC, shipped with a single giga-Ethernet interface. This Ethernet interface is thus shared between the two applications. This NoC configuration is not artificial as it can be seen as a subset of an initial larger $N \times N$ NoC that therefore leads to consider several instances of the problem we introduce in this section. A core of the NoC is identified by its (x,y) coordinates and we assume that (0,0) is located on the bottom left of the NoC. The square 3×3, whose left corner is located at (0,0) defines the regions where the tasks of the FADEC application are mapped. The square 4×4, located at (0,3), defines the region where the HM applications is mapped. Let us now describe the steps input I/O data received by an Ethernet interface go through in order to be used by a core of the NoC (blue and red arrows for respectively the HM and FADEC applications). We assume that Ethernet packets have a Maximum Transmit Unit (MTU) of 1500 bytes. We further assume that NoC packets can be made of up to 19 flow control digits (flits), as in the Tilera many-core. A flit is equal to 32 bits. The size of an Ethernet packet is thus generally several factors higher than the size of a NoC packet. Several NoC packets are therefore needed to transmit to a core an Ethernet payload, that must therefore be buffered within the Ethernet interface. Reaching a destination core can either be done directly or through an intermediate DDR controller. This choice is left to the user, and the last option is the case that we consider in this work. Note that I/O FADEC data are sent to the port 1 of the DDR, and can later be used from for instance either the port 4 or 5 of the DDR.

Flits of NoC packets are transmitted one by one by routers, i.e. in a pipeline way, by relying on wormhole switching strategy, with an dimension ordered XY routing policy and Round-Robin Arbitration (RRA) within routers. As routers have buffers of a few flits due to the area cost of memories in chips, flits of a same packet can thus be present on different routers. A NoC congestion occurs when a contention between two flows a at given router propagate backward due to the credit-based mechanism, preventing flits of other flows to also make progress. **In this work, we first consider the paths taken by flows originating from an Ethernet interface when reducing the contentions**, as we claim that many-core will be used as processing elements within a backbone Ethernet network. **If a NoC congestion occurs on one of these paths, the estimated WCTT of a flow can be higher than the arrival delay of the next incoming Ethernet packet**. In that case, this Ethernet packet may be dropped due to the lack of space in the Ethernet buffer. Previous Ethernet packets could indeed be stored in this buffer while waiting associated NoC flows can progress towards the DDR interface. Note that in this work we neglect the time a DDR request takes, however this delay would only increase the global one.

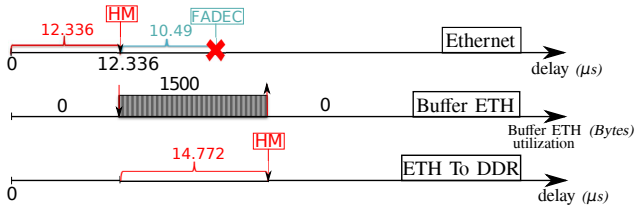

Fig. 2. Timeline of Ethernet and NoC packets showing that the FADEC Ethernet packet is dropped when the mapping of Figure 1 is used.

At both the Ethernet and NoC levels (and up to the DDR only), the Figure 2 shows the timeline of a frame from the HM application followed by one from the FADEC application. When the HM frame with payload size of 1500 bytes arrives at the single giga-Ethernet interface, i.e. after 12.336 μs (link traversal), it is stored into the interface buffer. Thus, if we consider that the Ethernet buffer size is 2 KB, as in Tilera, the buffer can only receive frames with 500 bytes of payload. Then FADEC frame can only be stored if all the HM frame has been transmitted to the DDR, as its size is 1270 bytes. The transmission of HM frame at the NoC level will use 20 packets of 19 flits and one packet of 15 flits. The WCTT of each HM packets on the NoC takes 701.5 ns. This value is obtained by reusing an existing method that gives the tightest WCTT values over RRA-based NoC [1]. Other strategies will therefore lead to even higher WCTT values and worsen the situation. We define the worst-case scenario as when each of these NoC packets will be blocked at each router by all NoC flows that can be encountered. So, as the HM frame is decomposed into 21 NOC packets, the global transmission delay of an HM frame to the DDR through the NoC will take t_1 = 14.772 μs. However, the transmission of the FADEC frame on Ethernet takes t_2 = 10.49 μs (transmission of 1270 bytes of payload at 1Gb/s). As the Ethernet buffer still contains the HM frame when the FADEC frame arrives, as $t_1 > t_2$, then FADEC frame is dropped. Therefore, the mapping proposed in Figure 1 does not take into account the I/O requirements leading to losing frames. The goal of the paper is then to propose a mapping approach considering the I/O transmission on the NoC.

III. Limitations of existing work

However, to the best of our knowledge, no mapping strategies for many-cores take into account core-to-I/O communications. We thus briefly review in this section existing contention-aware mapping strategies for core-to-core communications and discuss their limitations with respect to our problem.

Different strategies exist to reduce the number of contention flows on the path of the transmitted flows ([3], [11], [14], [13]) using minimization functions. However, all these approaches consider only the mapping of a single application.

Conversely, [6] considers the mapping of several applications by arbitrarily dividing the NoC into clusters. Each cluster is dedicated to an application. Then, within each cluster a congestion-aware mapping heuristic, similar to one in [14], minimizes the bandwidth utilization of NoC links. Authors of [4] enhance the definition of clusters for applications by making them near convex regions. Generating non-contiguous regions is avoided, thus reducing the congestion that can occur between applications, called the external congestion. But, due to the first core selection policy when building regions, mapping of an application over a fragmented region is possible [9]. In order to solve this problem, [9] propose to select the core having the most available neighbors (up to 4) to avoid region fragmentation and thus decrease both internal and external congestions. This solution, called CoNA, only considers direct neighbors when selecting the first core and then still lead to fragmentation of areas [7]. Smart Hill Climbing (SHiC) approach [7] considers a new metric called square-factor (SF) for selecting the first core when building regions. The SF of a core is the maximal size of the square area in which that core can be put in, to which the number of free cores around this square is added. The first core is then the one having a SF greater or equal to the size of the application to be mapped, i.e. the number of cores that are needed assuming a core can only execute a single task. [8] adapts SHiC so that contiguous regions as used to map critical applications, in order to reduce contentions, while non-critical applications are mapped over non-contiguous regions to increase the system throughput.

SHiC is the best congestion-aware mapping approach which is the closest related work to ours. The mapping done in Figure 1 has been in fact obtained using SHiC. As we can see, SHiC, and also all others existing mapping strategies, does not consider where I/O interfaces are located within the NoC when mapping applications. The NoC core-to-I/O flows may therefore suffer from external congestions, if the applications are not mapped close to the I/O interfaces they use. However, these interfaces could be shared between several applications. Besides, the internal mapping of applications also influence the WCTT of these core-to-I/O flows. Mapping the most communicating task to the first selected core, as most existing strategies do, may no longer be appropriate. The number of contentions core-to-I/O flows experiences should instead be reduced, so that their WCTT are decreased and avoid dropping incoming I/O packet. Finally, SHiC defines strict contiguous regions preventing the mapping of a set of applications whose total size is equal to the size of NoC.

IV. Proposal overview

To overcome these limitations, our approach also relies on a two steps process to map applications of different levels of criticalities. In this paper, instead of giving the mapping algorithms, we propose to describe our approach using on overview of these two steps. We currently assume a single Ethernet controller for the I/O interface, a single critical application amongst a set of non-critical applications and a single core-to-I/O flow per application, called core-to-Ethernet.

The first step of our mapping process, called the *external mapping*, assigns to each application a region and determine its shape. Our second step then maps the tasks of each application within its assigned region. When the sum of the size of each application is equal to the size of the NoC, i.e there is no free cores available, the external mapping starts with the critical application and assigns to it a region next to the Ethernet interface. In the other case, the external mapping consider the non-critical applications first. It starts to map them from the opposite side where the Ethernet interface is located. The critical application is thus the last one to be considered, in order to gather remaining free cores within its corresponding region. These cores will be used to provide more laxity in the second step of our mapping process when mapping the critical application. When assigning a region to an application, we consider the minimal rectangular shape that corresponds to its size and favors biggest shapes in next assignment of regions.

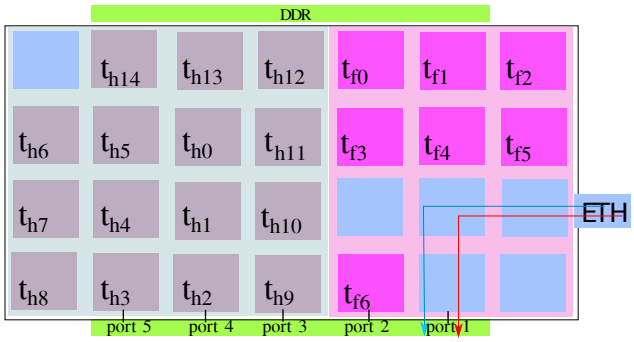

Fig. 3. Mapping of the FADEC and HM applications using our approach.

The second step of our mapping process is called the *internal mapping*. When mapping applications, its goal is to reduce the WCTT of NoC core-to-I/O flows for both the critical and non-critical applications. To this end, the internal mapping reduces the number of contentions on the paths taken by these flows according to different specific rules. On the example of Figure 3, the available free cores within the region of the critical application are put in priority on the path taken by core-to-I/O flow. Note that for application that are not mapped on the paths taken by core-to-I/O flows, we rely on the SHiC strategy.

Figure 3 shows the final mapping that our approach generates for our case study. Remember that it is composed of 28 cores, as it is a 7×4 mesh NoC. The FADEC application requires 7 cores, while the HM application requires 15 cores. The total size of both applications is thus 22, leaving initially 6 free cores. The external mapping therefore starts by assigning a region to the HM application. To this end, a 4×4 square region is defined and located at (0,0). One free core is thus lost and left unused when defining this region. The FADEC application is then assigned a 3×4 rectangular region, located at (0,4), and that integrates the 5 remaining free cores. The internal mapping uses 4 free cores, amongst the 5 available in its region, in a 2×2 square area next to the Ethernet controller.

V. Preliminary evaluation

First, let us assume that the internal mapping maps the applications by following the SHiC strategy. Compared to the mapping shown by Figure 1, the mapping of applications is thus simply permuted. For this mapping, the WCTT for reaching the DDR by a NoC packet of the core-to-Ethernet flow of the HM application is $610\ ns$. This leads to a WCTT for the core-to-Ethernet flow of $12.8\ \mu$s. This delay is thus reduced compared to the one shown by Figure 2. However, it is still higher than what is required to avoid dropping the the FADEC Ethernet packet. This demonstrates the need for a

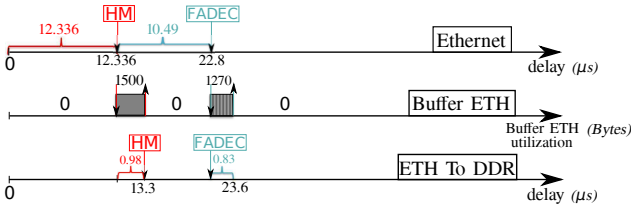

Fig. 4. Timeline of Ethernet and NoC packets showing that the FADEC Ethernet packet is no longer dropped when mapped using our approach.

strategy in the internal mapping that further reduces the WCTT of the NoC core-to-I/O flows of non-critical applications, in addition to the critical one.

Let us now assume that our internal mapping takes advantage of the free cores gathered by our external mapping. This corresponds to the mapping shown by Figure 3. The timeline of Figure 4 shows that the FADEC Ethernet packet is no longer dropped in that case. The free cores are indeed located on the path taken by the Ethernet-to-core NoC flows of the FADEC and HM applications. These flows are thus no longer blocked while progressing towards the DDR. The WCTT of NoC core-to-Ethernet packets (made of 19 flits) from the HM application is thus reduced to $47.15\ ns$. The global WCTT of the corresponding NoC flow of the HM application is then $0.98\ \mu$s thanks to our approach where the internal mapping reduces the number of blocking flows with the core-to-Ethernet flow. The HM Ethernet packet is therefore removed from the Ethernet buffer before the FADEC Ethernet packet arrives to the Ethernet interface. The global WCTT of the NoC core-to-Ethernet flow from the FADEC application is $0.83\ \mu$s.

We also evaluated our approach when all the cores of the NoC are used. To this end, we added two tasks t_{f7} and t_{f8} to the FADEC application. These additional tasks have the same characteristics as the tasks t_{f0} to t_{f5}. For the HM application, we reduced the number of tasks from 15 to 12 tasks. We assume a reduced 7×3 mesh NoC. Compared to the mapping shown by Figure 1, SHiC maps t_{f7} and t_{f8} at respectively (2,0) and (2,2). In this case, the same timeline as the one of Figure 2 is obtained. The FADEC Ethernet frame is thus still dropped. When however considering our approach, the external mapping assigns to the FADEC application a 3×3 square region next to the Ethernet interface. The internal mapping of the FADEC application leads to a WCTT of $483.5\ ns$ for the NoC core-to-Ethernet packets of the HM application. This corresponds to a global WCTT of $10.15\ \mu$s for reaching the DDR. The FADEC frame can thus reaches the Ethernet interface after the removal of the HM Ethernet packet from the Ethernet buffer. This example shows that our approach seems promising even if all the core of the NoC are used.

VI. Conclusion and Future Work

Existing contention-aware mapping strategies aim to minimize the inter-core congestion without taking into account requirements of I/O communications of applications. However, a NoC is mostly connected to other external systems through several Ethernet interfaces. The WCTT of NoC core-to-Ethernet flows depends on the congestions generated by the mapping of both critical and non-critical applications.

In this paper, we first show on an avionic case study that the solution generated by a state of the art contention-aware mapping strategies even lead to drop Ethernet packets used by a critical application, when mapped together with a non-critical application. We then show on the same case study that a two steps mapping strategy solve this issue. An external mapping step assigns the critical application near the considered Ethernet interface and reclaim free cores for the definition of the region where the critical application will be mapped in. Within this region, an internal mapping step reduces the contention both NoC core-to-Ethernet critical and non-critical flows experience. Work underway shows that our approach can be successfully applied over other case studies. Next step is the generalization and formalization of the internal mapping rules.

Further work includes the development of a software tool implementing our mapping strategy as well as consider all types of flow in our mapping strategy. We are also interested in generalizing our algorithm by supporting additional core-to-I/O flows, several critical applications and I/O interfaces.

References

[1] 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)*, pages 59–68, Siegen, Germany, June 2015.

[2] A. Burns, J. Harbin, and L. S. Indrusiak. A wormhole noc protocol for mixed criticality systems. In *Proc. of the IEEE 35th Real-Time Systems Symposium, RTSS*, pages 184–195, Rome, Italy, December 2014.

[3] C.-L. Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. In *IEEE Intl. Conf. on Computer Design (ICCD)*, pages 164–169, 2008.

[4] C.-L. Chou, U. Y. Ogras, and R. Marculescu. Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1866–1879, 2008.

[5] 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)*, pages 97:1–97:6, 2014.

[6] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes. Dynamic task mapping for mpsocs. *Design & Test of Computers*, 27(5):26–35, 2010.

[7] 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*, page 39, 2013.

[8] M. Fattah, A.-M. Rahmani, T. C. Xu, A. Kanduri, P. Liljeberg, J. Plosila, and H. Tenhunen. Mixed-criticality run-time task mapping for noc-based many-core systems. In *22nd Euromicro Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, pages 458–465. IEEE, 2014.

[9] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila. Cona: Dynamic application mapping for congestion reduction in many-core systems. In *30th Intl. Conf. on Computer Design (ICCD)*, pages 364–370, 2012.

[10] 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*, pages 63–72, Madrid, Spain, July 2014.

[11] A. Racu and L. S. Indrusiak. Using genetic algorithms to map hard real-time on noc-based systems. In *7th Intl. Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, 2012.

[12] Tilera corporation. *Tile processor user architecture manual*, Nov. 2011. UG101.

[13] B. Yang, L. Guang, T. Säntti, and J. Plosila. Tree-model based contention-aware task mapping on many-core networks-on-chip. *Communications in Information Science and Management Engineering*, 2012.

[14] C. Zimmer and F. Mueller. Low contention mapping of real-time tasks onto tilepro 64 core processors. In *18th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 131–140, 2012.