CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Are Virtual Channels the Bottleneck of Priority-Aware Wormhole-Switched NoC-Based Many-Cores?

Borislav Nikolic
Hazem Ali
Stefan M. Petters
Luís Miguel Pinho

CISTER-TR-130901
Version:
Date: 09-10-2013

Are Virtual Channels the Bottleneck of Priority-Aware Wormhole-Switched NoC-Based Many-Cores?

Borislav Nikolic, Hazem Ali, Stefan M. Petters, Luís Miguel Pinho

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Preemptions via virtual channels have been proposed as a means to introduce the notion of priorities and real-time concepts in wormhole-switched NoC-based architectures. This work presents a holistic approach, which utilises a novel three-staged mapping method in order to assess what should be the physical characteristics of the platform (and its interconnect), such that real-time guarantees can be provided, assuming a given workload. We estimate the ”gap” between platform characteristics required for the real-time analysis and those of currently available many-core platforms and propose to employ the existing feature of many-core platforms in order to significantly reduce this gap. The experiments demonstrate that virtual channels, an essential prerequisite for the real-time analysis, are not the bottleneck. The approach presented in this paper can help system designers to select/design the most suitable platform for a given workload, such that all temporal constraints are met and over-dimensioning is avoided.

Are Virtual Channels the Bottleneck of Priority-Aware Wormhole-Switched NoC-Based Many-Cores?*

Borislav Nikolić, Hazem Ismail Ali, Stefan M. Petters and Luís Miguel Pinho
CISTER/INESC-TEC, ISEP
Polytechnic Institute of Porto, Portugal
{borni, haali, smp, lmp}@isep.ipp.pt

ABSTRACT

Preemptions via virtual channels have been proposed as a means to introduce the notion of priorities and real-time concepts in wormhole-switched NoC-based architectures. This work presents a holistic approach, which utilises a novel three-staged mapping method in order to assess what should be the physical characteristics of the platform (and its interconnect), such that real-time guarantees can be provided, assuming a given workload. We estimate the "gap" between platform characteristics required for the real-time analysis and those of currently available many-core platforms and propose to employ the existing feature of many-core platforms in order to significantly reduce this gap. The experiments demonstrate that virtual channels, an essential prerequisite for the real-time analysis, are **not** the bottleneck. The approach presented in this paper can help system designers to select/design the most suitable platform for a given workload, such that all temporal constraints are met and over-dimensioning is avoided.

1. INTRODUCTION

Many-core platforms are the new frontier technology in the real-time embedded domain. These devices offer various beneficial possibilities; e.g. to implement new or enhance existing functionalities or to perform energy/thermal management and fault tolerance. However, the real-time analysis of many-cores is a challenging topic, most prominently because of contention for shared resources, like memory controllers and the interconnect medium. As the number of integrated cores within many-core platforms increases, contentions for shared resources also increase, and consequently latencies of operations involving those resources increase as well. Thus, before these platforms can be incorporated into the real-time embedded domain, analyses are needed which (i) provide temporal guarantees with as little pessimism as possible, (ii) allow the efficient utilisation of the underlying platform.

Traditional real-time approaches mostly study many-core platforms from the perspective of computational requirements. One well-established area is called *scheduling theory*, where some of the end objectives are to organise the workload execution and provide real-time guarantees, while at the same time minimising the required processing resources. The benefits are that over-dimensioning can be avoided, and a platform with the least possible resources can be selected or designed, such that it meets the requirements (i.e. provides temporal guarantees) for a given workload. However, very few works have explored the requirements regarding the characteristics of the underlying interconnect medium, which are necessary in order to derive end-to-end temporal guarantees for many-core systems.

The Network-on-Chip architecture [2] (NoC) became a prevailing interconnect medium and mainstream in many-cores [11,25], due to its scalability potential [14]. Most NoCs perform data transfer via *wormhole switching technique* [19], due to its good throughput and small buffering requirements. Currently available NoC-based, wormhole-switched many-cores employ a wide range of diverse strategies when implementing a wormhole switching technique, e.g. different sizes of basic transferable units – *flits*, different router operating frequencies, different arbitration policies, a (lack of) support for virtual channels, etc. These design trade-offs have a significant impact on both, performance and analysis.

In this work we focus on the aforementioned aspect, and, assuming a given workload, identify which are necessary characteristics of the interconnect medium, such that real-time guarantees can be derived. Specifically, we elaborate on required number of virtual channels and link capacities. The objective is to help system designers to select the platform (and the interconnect medium) with sufficient characteristics, and prevent over-dimensioning, which carries a significant importance in the embedded domain, since more resourceful systems usually consume more power. To facilitate this, we propose a heuristic-based, three-staged approach which maps application workload to the given platform, with the dual objective of fulfilling the timing constraints of the communication and minimising the resources of the interconnect architecture. The proposed mapping

*This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within VIPCORE (FCOMP- 01-0124-FEDER-015006), SENODs (FCOMP-01-0124-FEDER-012988) and SMARTS (FCOMP-01-0124-FEDER-020536) projects and by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grants SFRH/BD/79872/2011 and SFRH/BD/81087/2011.

RTNS 2013 , October 16 - 18 2013, Sophia Antipolis, France

http://dx.doi.org/10.1145/2516821.2516845.

method is used as a framework to conduct a comprehensive study of different design-choices related to NoC-based, wormhole switched many-cores, and observe their influence on the provided guarantees. The experiments demonstrate that (i) allowing more than the minimal number of required virtual channels during the mapping process in most cases is not beneficial, which is highly counter-intuitive and directly answers the question posed in the title of the paper, and (ii) mapping the workload with the primary objective of minimising the number of virtual channels leads to a near-optimal solution.

2. RELATED WORK

Contrary to the popular belief, a wormhole switching technique is not a novelty in academia, nor industry, but was introduced more than 20 years ago. However, it was largely neglected due to the fact that an alternative – *store-and-forward switching technique* was providing satisfactory results. As the amount of data that has to be transferred kept increasing, the buffering within routers became more and more challenging, which lately brought wormhole switching back into the focus. Nowadays, this technique is predominantly employed in many-core architectures [11, 25]. Furthermore, these platforms mostly route the packets via a static, dimension-ordered *XY routing policy* and a *round-robin arbitration* is used in routers.

If a platform provides only a single virtual channel [25], complex interference patterns may occur [15]. Several techniques were proposed to obtain upper bounds on the worst-case packet traversal delays [7–9], assuming the most common setup: a single virtual channel and a round-robin arbitration. However, such platforms are by design not well-suited for the real-time analysis, hence proposed methods obtain either pessimistic results [8, 9], or have complexity and scalability issues [7]. Conversely, if multiple virtual channels [5, 6] are provided within the platform [11], the benefits are twofold: (i) by avoiding idle routers the efficiency (throughput) of the wormhole switching is significantly improved [5, 6], and (ii) a notion of packet priorities can be introduced and packets can preempt each other within the network. The second aspect is particularly important for the real-time domain. By employing the aforementioned assumptions (i.e. distinctive per-packet priorities, per-priority virtual channels and flit-level preemptions [24]) Shi and Burns proposed a real-time communication analysis [21] with the objective to derive the worst-case traversal delays of individual packets. The same authors extended the analysis to allow several packets to share the same priority [22], known as *priority-share policy*. The latter approach is useful in scenarios where the number of virtual channels provided by the platform is less than the number of packets.

Assuming the aforementioned analysis with distinctive per-packet priorities, Mesidis and Indrusiak [18] and Racu and Indrusiak [20] proposed workload mapping approaches based on genetic algorithms, which objective is to derive a mapping where all timing constraints posed on individual packets are fulfilled. Shi and Burns elaborated on the analysis based on the priority-share policy and proposed a mapping method [23], which besides temporal constraints also focuses on minimising the number of consumed virtual channels.

This paper can be considered as the continuation of the previously mentioned works. However, rather than perceiving the platform as a given, closed system and providing a simple "yes/no" answer regarding the ability to fulfil all temporal constraints, we explore different platform properties and characteristics and observe to what degree they influence provided guarantees. This approach should help system designer to select/design the platform with minimal resources possible, such that all timing requirements are still met. Together with the attempt of Shi and Burns to minimise the number of virtual channels [23], this is the pioneering work in the real-time domain addressing resource minimisation for many-cores employing wormhole-switched priority-aware NoCs.

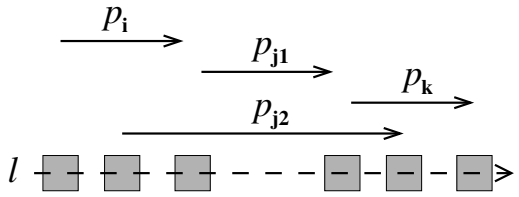


Figure 1: Example of contending packets

3. BACKGROUND / PRELIMINARIES

In this section we will introduce the basic concepts, which will help to subsequently describe the mapping algorithm (Section 4).

3.1 Wormhole Switching

Unlike traditional store-and-forward switching, where an entire packet indivisibly travels between consecutive routers from the source to the destination, wormhole switching divides a packet into small fixed-size elements called *flits*. The first flit establishes the path, and the rest follows in a pipeline manner, i.e. when the first flit progresses from one router to the next, the rest of the flits may also progress by one router. With this technique, each router on the path of the packet needs to store only one of its flits at any time instance, which significantly relaxes buffering requirements.

For wormhole-switched interconnects the XY routing technique became the predominant choice. The reason is twofold: (i) XY routing is deadlock and livelock free [10] and (ii) the paths are static and deterministic, which makes this policy particularly suitable for the real-time domain. With this policy flits of the packet travel firstly on the x-axis, and upon reaching the x-coordinate of the destination continue the traversal on the y-axis.

Traditional interconnects utilising the wormhole switching technique have only a single channel per direction per link, or what is better known as *single virtual channel*. An example is Tilera family of processors [25]. In this regime, once flits occupy buffers in the routers located on the path of the packet, other packets, which traverse the same path, have to wait until buffers become empty, i.e. until the flits of the existing packet leave contending buffers. This can cause very complex contention scenarios [15], where a packet can be blocked not only by packets with which it shares a part of the path, but also by those with which it does not. Figure 1 gives an illustrative example. Packets p_i, p_{j1}, p_{j2} and p_k contend for some links on the path l, and shaded rectangles represent routers. The packet p_i can be blocked by the packet p_{j2}, which in turn can be blocked by the packet p_k. Thus, a packet p_i can suffer blocking from p_k, even though they do not have a common part of the path.

Assuming a single virtual channel, if a packet suffers blocking, all its flits stall within currently occupied routers and prevent any packet from progressing in the same direction until itself is able to progress again (i.e. it is not blocked any more). This can cause poor performance and as a means to overcome this problem *multiple virtual channels* have been proposed [5, 6]. A virtual channel is nothing more than an additional buffer in the router, which allows to store stalled flits of a blocked packet and offer the progress to some other non-blocked packet. This implies that in an example illustrated in Figure 1, if p_{j2} is blocked by p_k, p_i can freely progress utilising one virtual channel, while stalled flits of p_{j2} are stored in the other. Single-Chip-Cloud Computer [11] (SCC) employs wormhole switching with virtual channels. Notice that virtual channels have additional significance in the real-time domain, as the same can be used for flit-level preemptions [24] based on packet priorities.

3.2 Existing Real-Time Analyses

Shi and Burns proposed a real-time analysis [21] to compute the worst-case delay of a packet, assuming that each packet has an implicit or constrained deadline and a distinctive priority. An additional assumption is that there exist per-priority virtual channels. The worst-case delay of a packet p, noted down as $R(p)$, consists of several terms (Equation (1)), namely isolation delay $C(p)$, lower-priority blocking $B(p)$ and higher-priority interference $I(p)$.

$$R(p) = C(p) + B(p) + I(p) \tag{1}$$

The first term is in the literature also known as *basic network latency*. It is equal to the delay of the first flit to reach the destination router, augmented by the processing delay of all flits at the destination router (Equation (2)). d_{sw} and d_t denote the latency to switch the crossbar and transfer one flit from one router to another, respectively. $nhops(p)$ and $size(p)$ symbolise the number of hops and the size of the packet p, while $size(f)$ represents the size of one flit.

$$C(p) = nhops(p) \times (d_{sw} + d_t) + \left\lceil \frac{size(p)}{size(f)} \right\rceil \times d_t \tag{2}$$

A packet p can additionally suffer lower-priority blocking within every router on its path. Equation (3) covers the worst case, where, at the moment when the first flit of p reaches any router on its path, a lower-priority packet just started the transfer.

$$B(p) = nhops(p) \times (d_{sw} + d_t) \tag{3}$$

Finally, a packet can suffer interference from higher-priority packets which share a part of the path with it, called *directly interfering packets*. Let $\mathcal{P}_D(p)$ be a set of directly interfering packets of p. Then, the higher-priority interference that p can suffer is given with Equation (4), where $T(p')$ denotes the minimum inter-arrival period of a directly interfering packet p'.

$$I(p) = \sum_{\forall p' \in \mathcal{P}_D(p)} \left\lceil \frac{R(p) + R(p') - C(p')}{T(p')} \right\rceil \times (C(p') + B(p')) \tag{4}$$

Notice the additional term in the ceiling brackets: $R(p') - C(p')$. It covers the case where two consecutive occurrences of p' can be distanced by less than $T(p')$, that is, there might exist a directly interfering packet of p' termed $p'' \in \mathcal{P}_D(p')$, which can cause interference to the first occurrence of p' and force it to appear as late as $\epsilon + R(p') - C(p')$, while the next (uninterfered) occurrence of p' can happen as early as $\epsilon + T(p')$. Thus, as p' can exhibit an aperiodic occurrence pattern, considering its periodic occurrences can be an unsafe assumption, and the additional term $(R(p') - C(p'))$ accounts for the worst-case. This is a well-known fact in the wormhole switching and for more details an interested reader is advised to consult the work of Shi and Burns [21]. Note, a packet can additionally suffer *release jitter*, which is defined as the maximum deviation of successive packets released from its period [21]. In this work we assume that release jitters are equal to zero. As the interference component (Equation (4)) gives a recursive notion to the worst-case delay (Equation (1)), $R(p)$ is computed iteratively, until reaching a fixed point (if one exists).

The requirement of the aforementioned analysis is that each packet has a distinctive priority and that the platform provides per-priority virtual channels, which, in some cases, may be very demanding. In order to decrease the number of needed virtual channels, the same authors proposed the analysis where multiple packets can share the same priority, called a priority-share policy [22]. However, the existence of packets with the same priority brings significant overheads and more complex blocking and interference patterns, similar to those involving a single virtual channel (see Subsection 3.1). In order to circumvent this problem the authors propose to group all packets with the same priority into a single entity called *composite packet* – $\widehat{p}$. Let $\mathcal{P}_C(\widehat{p})$ be a set of packets constituting $\widehat{p}$. Now, the isolation delay and the blocking delay of a composite packet $\widehat{p}$ are equal to the sum of isolation and blocking delays of packets from $\mathcal{P}_C(\widehat{p})$, i.e. $C(\widehat{p}) = \sum_{\forall p \in \mathcal{P}_C(\widehat{p})} C(p)$ and $B(\widehat{p}) = \sum_{\forall p \in \mathcal{P}_C(\widehat{p})} B(p)$. Finally, let $\mathcal{P}_D(\widehat{p})$ be a set of packets which can cause direct interference to any packet from $\mathcal{P}_C(\widehat{p})$ and hence to $\widehat{p}$. Formally:

$$\forall p : \exists p' \in \mathcal{P}_C(\widehat{p}) \wedge p \in \mathcal{P}_D(p') \Rightarrow p \in \mathcal{P}_D(\widehat{p}) \tag{5}$$

Interference that $\widehat{p}$ suffers is equal to the sum of interferences caused by all packets from $\mathcal{P}_D(\widehat{p})$ (Equation (6)).

$$I(\widehat{p}) = \sum_{\forall p' \in \mathcal{P}_D(\widehat{p})} \left\lceil \frac{R(\widehat{p}) + R(p') - C(p')}{T(p')} \right\rceil \times (C(p') + B(p')) \tag{6}$$

The worst-case delay of a composite packet: $R(\widehat{p}) = C(\widehat{p}) + B(\widehat{p}) + I(\widehat{p})$ presents an upper-bound on the delays of all packets from $\mathcal{P}_C(\widehat{p})$, i.e. $R(p) = R(\widehat{p}), \forall p \in \mathcal{P}_C(\widehat{p})$. Notice two potential sources of pessimism: (i) all same-priority packets are grouped in one entity, although some of them can be treated independently (ii) not all directly interfering packets can cause interference during an entire period $R(\widehat{p})$. We illustrate this with an example given in Figure 1 and with the packet parameters given in Table 1. For the clarity of the example, let us take a simplifying assumption that lower-priority blocking does not exist, i.e. $B(p) = 0, \forall p$.

Table 1: Example of packets

Packet	Priority	C	D = T	B
p_i	P_i	1	3	0
p_k	$P_k < P_i$	1	3	0
p_{j1}	$P_j < P_k$	1	10	0
p_{j2}	$P_j < P_k$	1	10	0

p_i and p_k do not suffer interference, hence $R(p_i) = C(p_i) + B(p_i) = C(p_i) = 1$ and $R(p_k) = C(p_k) + B(p_k) = C(p_k) = 1$.

As p_{j1} and p_{j2} have the same priority, they can be grouped within $\widehat{p_j}$.

$$R(\widehat{p_j}) = C(p_{j1}) + C(p_{j2}) + \cancelto{0}{B(p_{j1})} + \cancelto{0}{B(p_{j2})} + \sum_{\forall p \in \{p_i, p_k\}} \left\lceil \frac{R(\widehat{p_j}) + R(p) - C(p)}{T(p)} \right\rceil \times \left(C(p) + \cancelto{0}{B(p)} \right)$$

$$R(\widehat{p_j})^0 = 1 + 1 + \left\lceil \frac{0+1-1}{3} \right\rceil \times 1 + \left\lceil \frac{0+1-1}{3} \right\rceil \times 1 = 2$$

$$R(\widehat{p_j})^1 = 1 + 1 + \left\lceil \frac{2+1-1}{3} \right\rceil \times 1 + \left\lceil \frac{2+1-1}{3} \right\rceil \times 1 = 4$$

$$R(\widehat{p_j})^2 = 1 + 1 + \left\lceil \frac{4+1-1}{3} \right\rceil \times 1 + \left\lceil \frac{4+1-1}{3} \right\rceil \times 1 = 6$$

$$R(\widehat{p_j})^3 = 1 + 1 + \left\lceil \frac{6+1-1}{3} \right\rceil \times 1 + \left\lceil \frac{6+1-1}{3} \right\rceil \times 1 = 6$$

However, by analysing the given example, it can be concluded that the worst-case scenario for p_{j1} occurs when it gets blocked by p_{j2} which in turn gets preempted by both p_i and p_k. Hence $R(p_{j1}) = 4$. Similarly, the worst-case scenario for p_{j2} occurs when it gets preempted by p_i, then blocked by p_{j1}, preempted by p_k and finally again preempted by p_i. Hence $R(p_{j2}) = 5$.

The purpose of the aforementioned example is to show the pessimism related to the priority-share policy and we recognise this area as a potential topic for future work. However, in this paper we elaborate on an alternative approach which is described in the very next subsection. It exploits novel features of current many-core platforms and allows to minimise the number of virtual channels, while still obtaining the worst-case delays with the analysis based on per-packet distinctive priorities.

3.3 Dynamically Changing Virtual Channels

The two aforementioned analyses assume that each packet traverses its entire path through the same, statically assigned virtual channel. The SCC [11] platform provides 8 virtual channels, but also offers the possibility to dynamically change virtual channels that one packet traverses [12]. This feature was described using an analogy about how cars switch lanes on the highway. We propose to use this possibility in the following way: (i) each packet still maintains its priority constant during an entire traversal, (ii) a packet may occupy different virtual channels within routers on its path, (iii) the virtual channel for a given packet at a given router is decided by considering all packets contenting for that router, and assigning a unique virtual channel to each one of them. This approach still allows to perform the analysis based on distinctive priorities, and yet dramatically reduces the requirement for virtual channels, as illustrated with an example given in Figure 2 assuming the parameters given in Table 2.

Table 2: Example of packets

Packet	Priority	C	D = T	B
p_i	P_i	1	3	0
p_j	$P_j < P_i$	1	3	0
p_k	$P_k < P_j$	1	10	0
p_m	$P_m < P_k$	2	10	0

Now, let us consider 3 different techniques: (i) analysis with distinctive priorities – DP, (ii) analysis with priority-share policy (PS), where p_i and p_j are grouped within the same priority and the same is done with p_k and p_m, and (iii) analysis with distinctive priorities and the support for dynamic changes of virtual channels (DDP). The worst-case delays and the requirements for virtual channels are given in Table 3. The calculation steps were omitted.

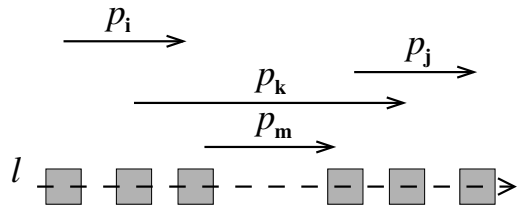


Figure 2: Example of contending packets

Table 3: Worst-case delays and needed virtual channels

Analysis	$R(p_i)$	$R(p_j)$	$R(p_k)$	$R(p_m)$	VC_{min}
DP	1	1	3	3	4
PS	2	2	9	9	2
DDP	1	1	3	3	2

Notice that DDP approach performs the same analysis as DP, but requests virtual channels equal to the maximum number of contentions at any router, which is in this example 2. This is the first work that proposes the aforementioned feature of SCC platform to be used for minimising virtual channels and it raises two fundamental questions: (i) **does SCC provide enough virtual channels to satisfy the requirements of present and future real-time embedded applications**, and if so, (ii) **is priority-share policy needed after all?** We will try to answer these questions in the sections dedicated to evaluations and conclusions.

4. PROPOSED APPROACH

This section presents an additional terminology (Subsection 4.1) that is essential for the full comprehension of the system model. Then, a detailed description of the proposed mapping algorithm is given (Subsections 4.3 - 4.6), as well as main motives and objectives for developing such an approach (Subsection 4.2).

4.1 Terminology

The platform under consideration Π is a homogeneous NoC-based wormhole-switched many-core system which consists of $n \times n$ tiles. Although some platforms allow the tile to be shared by multiple cores [11], in this work we assume that a tile consists of a single core and a single router, like the system depicted in Figure 3 [25]. Thus, $\Pi = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_{n^2-1}, \mathcal{M}_{n^2}\}$. Horizontally and vertically neighbouring cores are (via routers) connected by two unidirectional links. All links have the same capacity $B_{link} = \frac{size(f)}{d_{sw}+d_t}$, expressed as the size of one flit, divided by the time needed to transfer it between two neighbouring routers. Furthermore, a platform consists of $\widehat{v}_{lim}$ virtual channels, such that $\widehat{v}_{lim} \geq \widehat{v}$, where $\widehat{v}$ represents the maximum number of contentions occurring at any router.

The workload is described by a task-set $\tau = \{\tau_1, ..., \tau_x\}$. As the focus of this work is on the interconnect medium, the execution parameters of individual tasks are out of scope of this paper. We assume that an entire task-set τ is by some fully-partitioned [17] or semi-partitioned [4, 13] scheduling

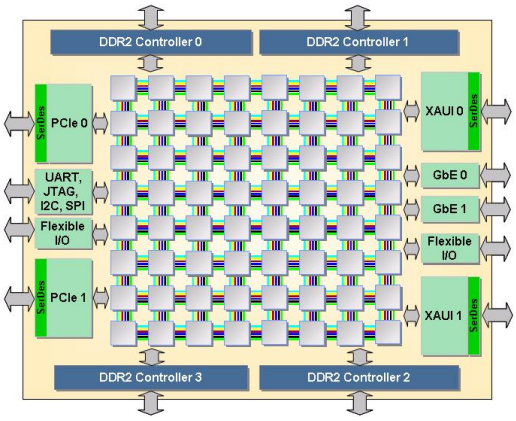


Figure 3: TILEPro64 Platform

approach converted into a mega-task-set $\widehat{\tau} = \{\widehat{\tau}_1, ..., \widehat{\tau}_{n^2}\}$, where each mega-task $\widehat{\tau}_i \in \widehat{\tau}$ represents a union of entire tasks and/or fractions of tasks from τ, such that $\widehat{\tau}_i$ can be scheduled on one core. Notice that there are as much mega-tasks as there are cores in the system, hence each mega-task should be mapped to a unique core.

Furthermore, each task $\tau_i \in \tau$ is characterised by a set of outgoing packets destined towards other tasks. Packets represent inter-task communication. Each mega-task $\widehat{\tau}_i \in \widehat{\tau}$ inherits the packets of tasks constituting it. A packet p_j is characterised by a unique priority – P_j, a minimum inter-arrival period $T(p_j)$, a deadline $D(p_j) \leq T(p_j)$, a packet size – $size(p_j)$, a source mega-task – $\widehat{\tau}_S$, a destination mega-task – $\widehat{\tau}_D$ and a distance between them expressed by (i) a number of hops – $nhops(p_j)$ and (ii) a set of traversed links – $path(p_j)$. A packet has *met its deadline* if its worst-case traversal delay $R(p_j)$ is less than or equal to the deadline, i.e. $R(p_j) \leq D(p_j)$. If all packets of all mega-tasks meet their deadlines, the system is considered as *feasible*.

4.2 Mapping Objectives

The main objective of the mapping process is to efficiently map a given mega-task-set to a given platform: $\widehat{\tau} \rightarrow \Pi$. A method should be used by a system designer, and should ideally derive a mapping plan (solution) which is feasible. In cases where this requirement can not be fulfilled, the method should provide information about the service that a given platform Π can provide to a given workload $\widehat{\tau}$, e.g. assuming initial packet sizes the system is not feasible, however, assuming that each packet has a half of its initial size the system is feasible. This information is useful for two reasons. First, it gives system designer feedback on how close/far the current configuration is from one that can guarantee the feasibility. Consequently the system designer can reconfigure the platform characteristics and repeat the mapping process until finding the configuration which guarantees feasibility with as less resources as possible. Second, variable packet sizes are in some scenarios acceptable [3] and proportional to the provided quality of service. Hence, when mapping such a workload the system designer might find it more convenient to use a platform where a workload is feasible with a certain quality (packet sizes), rather than buying a much more expensive platform which will guarantee the best quality (feasible system with initial packet sizes). This approach will also allow us to study the effect of different platform characteristics on the provided guarantees and will help us to identify the bottlenecks of interconnects in commodity many-cores.

4.3 Mapping Method Overview

To achieve the aforementioned objectives, the proposed approach is designed in a configurable way that allows to analyse a wide range of different scenarios, arising from the possibility to explore different platform configurations and different workloads (as minutely explained in Section 5), with the central motive of finding an adequate platform for a given workload. The proposed method consists of three mapping stages: (i) *initial phase*, (ii) *VC minimisation phase* and (iii) *workload-exploration phase*.

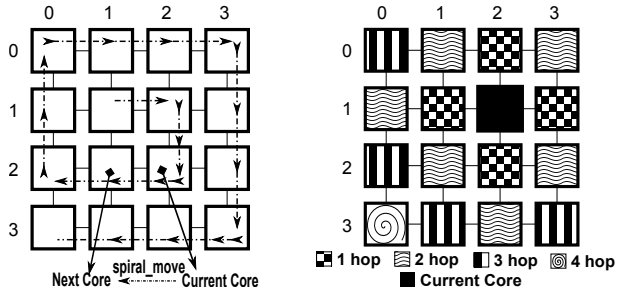
The initial phase is inspired by the mapping algorithm of dataflow applications on many-core platforms [1], where the objective is to map the mega-task-set $\widehat{\tau}$ on the platform Π in such a way that heavily communicating mega-tasks are mapped as close to each other as possible. This strategy should help in reducing contentions and provide a "good" starting point (initial solution) for further optimisations during subsequent phases. While mapping, the initial phase does not consider packet nor platform properties (i.e. packet sizes, timing constraints, link bandwidths, available virtual channels).

The main goal of the VC minimisation phase is to take the solution of the initial phase as an input, and optimise it in such a way that the number of needed virtual channels $\widehat{v}$ is minimised. The VC minimisation phase is implemented as a Simulated Annealing (SA) meta-heuristic [16]. As the single objective of the VC minimisation phase is to minimise the number of virtual channels, this phase also does not consider packet nor platform properties.

The main role of the final, workload-exploration phase is to discover and quantify the guarantees that can be provided for a given workload $\widehat{\tau}$ by a given platform Π. It takes the output of the VC minimisation phase as an input, which assures that the starting point is the solution with the minimal number of virtual channels $\widehat{v}$, and performs the optimisation with the objective to derive a feasible solution (if possible), assuming specific platform characteristics (i.e. link bandwidths B_{link} and available virtual channels $\widehat{v}_{lim} \geq \widehat{v}$) and workload temporal constraints. The workload-exploration phase is also implemented as a SA meta-heuristic, and it should ideally derive a feasible solution. In cases where it is not possible, the workload-exploration phase investigates to what percentage of initial sizes all packets have to be uniformly reduced, such that the feasibility can be guaranteed, and consequently tries to find a solution where the reduction is the least possible.

4.4 Initial Phase

The main criteria of the initial phase is to map communicating mega-tasks near to each other by using a core selection methodology proposed by Ali et al. [1], that proved to decrease communication overhead and response time of applications. This methodology consists of two main functions, *spiral_move* and *find_nearest_core*, as shown in Figure 4. The first function, *spiral_move*, defines a fixed spiral path on the platform that is followed while mapping the mega-task-set $\widehat{\tau}$. The *spiral_move* function returns the next core on the spiral path every time it is called, as shown in Figure 4a. The second function, *find_nearest_core*, takes a reference core as



(a) *spiral_move* (b) *find_nearest_core*

Figure 4: Core Selection methodology [1]

an input and starts searching for a free core one hop away from the reference core. If not possible, it searches for a free core two hops away, and so on, until finding a possible core to map the mega-task. The search criteria starts by finding the nearest core in this order: North, South, East and West. The first core that the *find_nearest_core* function finds is returned for mapping. Figure 4b shows the searching regions, classified according to the distance from the reference core.

Algorithm 1: Initial phase algorithm

$\widehat{\boldsymbol{\tau}}$: unsorted mega-task-set , $\widehat{\tau} = \{\widehat{\tau}_1, \ldots, \widehat{\tau}_{n^2}\}$.
$\widehat{\boldsymbol{\tau}}^{\boldsymbol{ord}}$: set of ordered $\widehat{\tau}$.
$\widehat{\boldsymbol{\tau}}_{\boldsymbol{i}}^{\boldsymbol{ch}}$: set of child mega-tasks of a mega-task $\widehat{\tau}_i$.
$\boldsymbol{\mathcal{M}_k}$: core index.

```
begin
    τ̂^ord = orderMegaTaskSet(τ̂);
    foreach τ̂_i in τ̂^ord do
        if τ̂_i is not mapped then
            M_k = spiral_move();
            map(τ̂_i, M_k);
        τ̂_i^ch = getTaskChildSet(τ̂_i);
        if τ̂_i^ch is not empty then
            foreach τ̂_j in τ̂_i^ch do
                M_k = find_nearest_core(τ̂_i);
                map(τ̂_j, M_k);
end
```

The mapping algorithm of the initial phase (shown in Algorithm 1), starts by sorting the mega-task-set in a non-increasing order of total number of packets (sent and received). Then, it picks the mega-tasks in order and checks whether they are mapped or not. If the mega-task $\widehat{\tau}_i$ was already mapped, the algorithm checks the mapped mega-task (parent) for its children (mega-tasks that either receive or send packets to the mapped mega-task) and maps them near to their parent using *find_nearest_core* function. On the other hand, if the picked up mega-task was not mapped yet, the algorithm calls the *spiral_move* function to determine the next free core on the spiral path to map it. Then, it maps its children mega-tasks using *find_nearest_core* function in the same way as explained previously.

4.5 VC Minimisation Phase

The goal of this phase is to minimise the number of needed virtual channels $\widehat{v}$, considering the solution produced by the initial phase as a starting point. This phase is described with Algorithm 2 and it is implemented with SA meta-heuristic.

Algorithm 2: Virtual channel minimisation algorithm

$\boldsymbol{t_{max}}$: maximum temperature of SA.
$\boldsymbol{t_{min}}$: minimum temperature of SA.
$\boldsymbol{t_{step}}$: decremental step of temperature decrease.
$\boldsymbol{t_{curr}}$: current temperature.
$\boldsymbol{c_t}$: constant representing the number of iterations in each t_{step}.
$\boldsymbol{\widehat{v}_{new}, \widehat{v}_{old}}$: new and old value of number of needed VCs.
$\boldsymbol{\bar{v}_{new}, \bar{v}_{old}}$: new and old value of average number of contentions.
$\boldsymbol{r}$: random number [0,1].
$\boldsymbol{P_w}$: probability of making a transition from the current state.

```
begin
    for (t_curr = t_max; t_curr ≥ t_min; t_curr −= t_step) do
        it = 0;
        while it ≤ c_t do
            {τ̂_i, τ̂_j : (τ̂_i ∧ τ̂_j) ∈ τ̂, τ̂_i ≠ τ̂_j} // randomly picked
            Swap(τ̂_i, τ̂_j);
            v̂_new = calculateMaxContentions();
            if v̂_new ≤ v̂_old then
                if (v̄_new < v̄_old) ∨ (r < P_w × t_curr/t_max) then
                    v̂_old = v̂_new;
                    v̄_old = v̄_new;
                else
                    Swap(τ̂_i, τ̂_j);
            else
                Swap(τ̂_i, τ̂_j);
            it++;
end
```

Through the exploration of the solution space, the algorithm makes a transition from one solution to another with respect to an *acceptance test*. This test consists of two conditions, of which at least one must be satisfied in order to accept the new solution. These two conditions are: (i) the average number of contentions of the new solution is less than the old one ($\bar{v}_{new} < \bar{v}_{old}$), and (ii) the acceptance probability test function ($r < P_w \times {}^{t_{curr}}/_{t_{max}}$), where r is a random number between $[0, 1]$, P_w is the probability of accepting a new transition, and (t_{curr}, t_{max}) are the current and maximum temperature of the algorithm, respectively. As is visible, the acceptance probability test is a function of t_{curr}. This means that at high temperatures the algorithm has a higher tendency to accept worse solutions in an attempt to transition to a new solution space where it can find a better solution. As the algorithm temperature cools down, this tendency decreases and the algorithm starts to lock on the best solution in the current neighbourhood.

As shown in Algorithm 2, the algorithm starts its iterations by setting the current temperature t_{curr} to a high value t_{max} enough to be able to explore the solution space. Then, it picks randomly two mega-tasks $\widehat{\tau}_i$ and $\widehat{\tau}_j$ and swaps them. Consequently, the rerouting of their respective incoming and outgoing packets is performed. If the number of needed VCs for the new solution $\widehat{v}_{new}$ is less than or equal to the number of needed virtual channels from the old solution $\widehat{v}_{old}$, i.e. $\widehat{v}_{new} \leq \widehat{v}_{old}$, the new solution is eligible to evaluate its acceptance probability using the acceptance test previously described, otherwise, it is rejected. In cases where the new solution progresses to the acceptance test and it is proved to be true, the algorithm accepts the new solution and updates the values of $\widehat{v}_{old}$ and $\bar{v}_{old}$. Otherwise, it reverts back to the old solution.

4.6 Workload-Exploration Phase

The goal of the workload-exploration phase is to maximize the size of all packets $\widehat{S}$ traversing the NoC, such that packets' timing constraints are satisfied, and that the num-

ber of employed virtual channels does not exceed the limit – $\widehat{v}_{lim}$, which represents the number of channels provided by the platform Π. Similar to the VC minimisation phase, this phase is also based on a SA meta-heuristic, but with a different goal, which is in this case to maximise $\widehat{S}$. This algorithm also makes a transition from one solution to another with respect to an acceptance test. However, the acceptance test of the workload-exploration phase differs in the first condition, where it compares the new and the old values of maximum feasible packet sizes, $(\widehat{S}_{new} > \widehat{S}_{old})$, in order to accept new solutions.

Algorithm 3: Packet size optimisation algorithm

```
S_new, S_old: new and old maximum feasible packet size.
v_lim: number of VCs provided by the platform.
begin
    for (t_curr = t_max, Ŝ_old = 0; t_curr ≥ t_min; t_curr −= t_step) do
        it = 0;
        while it ≤ c_t do
            {τ̂_i, τ̂_j : (τ̂_i ∧ τ̂_j) ∈ τ̂, τ̂_i ≠ τ̂_j} // randomly picked
            Swap(τ̂_i, τ̂_j);
            v̂_new = calcMaxContentions();
            if v̂_new ≤ v̂_lim then
                Ŝ_new = calcMaxPktSizes();
                if (Ŝ_new > Ŝ_old) ∨ (r < P_w × t_curr/t_max) then
                    Ŝ_old = Ŝ_new;
                    if Ŝ_new == 1 then
                        exit;
                else
                    Swap(τ̂_i, τ̂_j);
            else
                Swap(τ̂_i, τ̂_j);
            it++;
end
```

As shown in Algorithm 3, the algorithm starts its iterations by setting all packet sizes $\widehat{S}_{old}$ to zero and the current temperature t_{curr} to a high value t_{max} enough to be able to explore the solution space. Then, it picks randomly two mega-tasks $\widehat{\tau}_i$ and $\widehat{\tau}_j$ and swaps them. This also requires the rerouting of their respective incoming and outgoing packets. If the number of needed VCs of the new solution $\widehat{v}_{new}$ exceeds the number of virtual channels provided by the platform $\widehat{v}_{lim}$, the new solution is rejected. Otherwise, the new solution is considered to calculate its $\widehat{S}_{new}$ value using *calcMaxPktSizes* function. This function is responsible for incrementing sizes of all packets of all mega-tasks uniformly, and consequently testing whether all packets' timing constraints are still satisfied. The function returns the value $\widehat{S}_{new}$, which represents the maximum packet sizes, such that the feasibility is preserved. After computing $\widehat{S}_{new}$, the new solution is ready to evaluate its acceptance probability using the acceptance test. In cases where the new solution passes the acceptance test, the algorithm accepts the new solution and updates the value of $\widehat{S}_{old}$. Otherwise, it reverts back to the old solution and tries a different swap. After several iterations, the maximum packet size $\widehat{S}_{new}$ can reach the value of 1. This means that the current mapping of the mega-task-set $\widehat{\tau}$ on the platform Π is feasible with initial sizes of all packets. If $\widehat{S}_{new}$ is less than 1, this means that the system can guarantee the feasibility, but only assuming that sizes of all packets are uniformly reduced to $max\{\widehat{S}_{old}, \widehat{S}_{new}\}$.

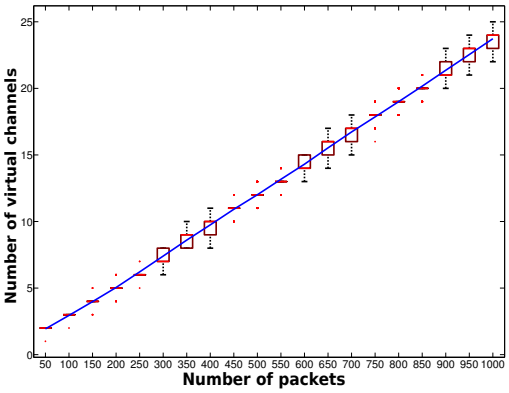


Figure 5: Virtual channels do scale wrt packets

5. EVALUATIONS AND EXPERIMENTS

In this section we employ the proposed mapping approach and test how platform characteristics influence the derived feasibility guarantees. Specifically, we observe how guarantees change with varying number of virtual channels and link bandwidths. In order to provide a complete and comprehensive study, and cover a wide range of scenarios ranging from lightly to extremely loaded networks, different workloads are generated and consequently analysed.

5.1 Experimental Setup

The analysis parameters are given in Table 4, where packet sizes are randomly generated values, assuming a uniform distribution. For each packet a source and a destination mega-task $\widehat{\tau}_S$ and $\widehat{\tau}_D$ are randomly selected.

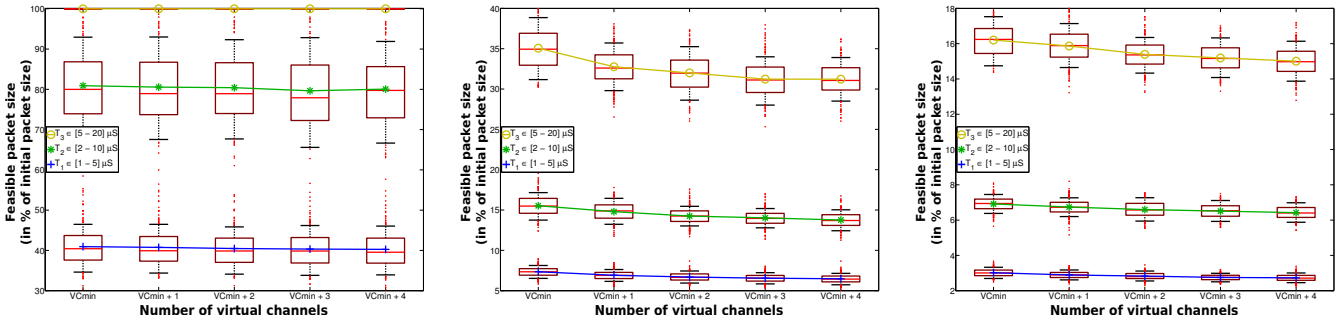
Table 4: Analysis parameters

Platform size – $n \times n$	**10 × 10**
Mega-task-set size $\lvert\widehat{\tau}\rvert$	**100**
Router switch latency – d_{sw}	**1 cycle**
Router transfer latency – d_t	**3 cycles**
Flit size – $size(f)$	**16B**
Packet sizes – $size(p)$	**[32B - 32kB]**

5.2 Experiment 1: Do Virtual Channels Scale?

As explained in Section 4, our approach is based on the assumption that a platform provides at least $\widehat{v}$ number of virtual channels, where $\widehat{v}$ is equal to the maximum number of contentions at any router. In this experiment we wanted to test how (un)realistic that requirement is. In other words, we identify the number of needed virtual channels for a given workload, and observe the change of this value when the number of packets increases. For that purpose, we generated different workload categories, ranging from 50 to 1000 packets for an entire mega-task-set. Each category consists of 1000 mega-task-sets. For each mega-task-set we performed the mapping (only initial and VC minimisation phases of the proposed approach) and computed the minimum number of virtual channels needed to support it – $\widehat{v}$. The results are given in Figure 5, where a horizontal axis corresponds to the number of packets and a vertical axis stands for the number of virtual channels. The whiskers were set to 25^{th} and 75^{th} percentile.

As already known, the NoC architecture is widely accepted due to its scalability potential related to traffic in



(a) Number of packets = 100 (b) Number of packets = 300 (c) Number of packets = 500

Figure 6: Additional virtual channels do **not** help in feasibility guarantees

general, hence an intuitive guess would be that the number of needed virtual channels also scales. Figure 5 confirms that assumption with an almost-linear dependency between the number of packets and virtual channels, and also gives a quantitative estimate regarding how many channels are needed for various workloads. In particular, currently available SCC platform, with its 8 virtual channels, can accommodate around 300 packets. On average, an addition of one channel increases the potential of the platform to accept 50 new packets. Recall that the traditional distinctive-priority approach requires that the number of virtual channels is equal to the number of packets, and notice the reduction achievable with our approach where packets are allowed to dynamically change virtual channels, i.e for 1000 packets our method requires, on average, only 23 channels. We further elaborate on practical implications of these findings in Section 5.5.

5.3 Experiment 2: Do Virtual Channels Help?

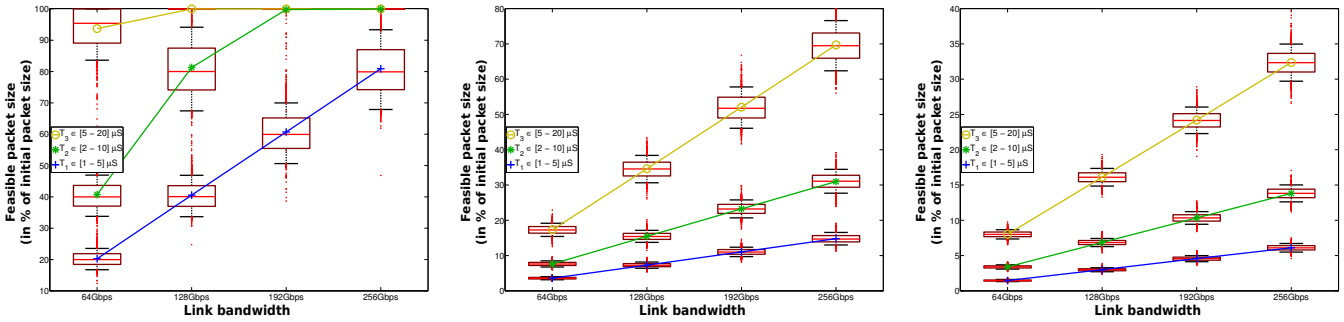
The previous experiment demonstrated that the proposed approach efficiently maps the workload, such that the number of needed virtual channels is minimised to the level beyond any expectation we had. This implies that the mapping process evenly distributes the contentions across the entire platform and avoids hotspots. However, as the minimisation of router contentions (i.e. virtual channels) is the central criteria, this can result in solutions where some packets are forced to traverse long routes, which can in turn have an impact on feasibility. Conversely, placing highly communicative mega-tasks close to each other would minimise the packet routes and might potentially improve feasibility, but would create hotspots and cause more contentions within some routers. Thus, our intuitive guess was that feasibility highly depends on the number of available virtual channels, and we wanted to quantify the trade-off, that is, to what degree can feasibility be improved by employing an additional virtual channel.

We generated 3 workload categories containing 100, 300 and 500 packets. For each category we had 3 subcategories with the following packet minimum inter-arrival periods: $T_1 \in [1-5]\mu s$, $T_2 \in [2-10]\mu s$ and $T_3 \in [5-20]\mu s$. For each subcategory 200 mega-task-sets were generated and consequently mapped. The results are given in Figure 6, where a horizontal axis stands for the number of virtual channels provided by the platform – $\widehat{v}_{lim}$, and the vertical axis stands for the provided feasibility guarantees, that is, to what percentage of initial sizes all packets have to be uniformly reduced, such that the system is feasible. A value VC_{min} corresponds to the number of virtual channels obtained by the VC minimisation phase, i.e. $VC_{min} = \widehat{v}$.

This experiment reported highly unexpected and unintuitive results, which imply that adding virtual channels might negatively impact the efficiency of the mapping process and produce a solution which is worse than the one obtained with minimum number of virtual channels $\widehat{v}$. Our initial though was that these surprising results might be related to specific workloads or the consequence of some parameters and inherent properties of SA meta-heuristic itself. To rule out those possibilities, we performed the experiment several times with different SA parameters and, as described above, with very diverse traffic loads. However, the results remained very similar and consistently demonstrated a systematic decrease in feasibility guarantees as the number of virtual channels increased. Thus, the only left explanation for these unexpected findings is that virtual channels are **not** bottlenecks. We interpret this in the following way: minimising virtual channels (contentions) indeed contributes to feasibility. As this objective tends to distribute the contentions on the grid as evenly as possible, it might cause longer traversal paths of some packets. However, as the load is equally spread across all the links, even assuming those longer paths better guarantees can be provided. Conversely, minimising packet distances causes hotspots and even if some packet traverses a short distance, links it consumes might be heavily loaded, hence highly impacting its feasibility. Therefore, we believe that (i) the solution obtained with the minimal number of virtual channels is one of near-optimal solutions, and (ii) adding more virtual channels, in most cases, unnecessarily expands the solution space which causes SA to drift away from the "good" solution space and frequently conclude the search with a worse solution. Of course, this can not be analytically and/or experimentally proven as workload mapping is a NP-Hard problem, which is computationally intractable even for small grids, e.g. 4×4 [10]. However, small scale experiments with exhaustive enumeration and different meta-heuristics are the possibilities to further support or deny our claims, and we see these activities as potential future work. The implications of these surprising findings are further discussed in Section 5.5.

5.4 Experiment 3: What is the Bottleneck?

As previous experiment provided experimental evidence that virtual channels might not be the bottleneck, in order to test the limits of the platform in this experiment we focused on another parameter – link bandwidth. This char-



(a) Number of packets = 100 (b) Number of packets = 300 (c) Number of packets = 500

Figure 7: Link bandwidths are the bottlenecks

acteristic is, as explained in Subsection 4.1 computed as the size of one flit (in most cases equal to the width of the link) divided by the time needed to transfer one flit between two neighbouring routers, which is mostly dependant on the frequencies on which routers operate. Individual link bandwidth of SCC platform is around 256 Gbps, while Tilera platforms provide around 166 Gbps. Assuming a variety of network loads, in this experiment we analysed how different link bandwidths influence provided feasibility guarantees.

We generated 3 workload categories consisting of 100, 300 and 500 packets. Each category has 3 subcategories, which differ in packet minimum inter-arrival periods: $T_1 \in [1-5]\mu s$, $T_2 \in [2-10]\mu s$ and $T_3 \in [5-20]\mu s$. For each subcategory 1000 mega-task-sets were generated and consequently mapped, assuming that platform provides only the minimal number of virtual channels, i.e. $\widehat{v}_{lim} = \widehat{v}$. The results are given in Figure 7, where the horizontal axis stands for the link bandwidth, and the vertical axis represents the provided feasibility guarantees, expressed as the maximum percentage of initial packets sizes to which all packets have to be reduced, such that the system is feasible.

This experiment reported results which are expected, suggesting that the derived guarantees linearly depend on link bandwidths. The trend is similar for scenarios which cover moderately and extremely loaded networks. The only exceptions are cases for 100 packets and T_1 as well as T_2 periods (Figure 7a), where, due to lighter load, solutions can be found for which the feasibility of initial packet sizes can be guaranteed even with smaller bandwidths. Thus, in general, link bandwidths can be perceived as the bottleneck of the system, which coincides with the intuition.

5.5 Discussions

The proposed technique to employ the existing feature of SCC platform in order to minimise the number of needed virtual channels (see Subsection 3.3) significantly relaxes the requirements for platform characteristics which are needed for the real-time analysis. Through experiments we have demonstrated that (i) the number of needed virtual channels linearly scales with the increasing traffic and (ii) the number is not unreasonably high and should be achievable with forthcoming generations of many-core platforms. We have shown that limiting the number of channels to the minimum while mapping the workload is, in most cases, a beneficial approach and leads to a near-optimal solution. The aforementioned facts altogether answer the question posed in the title of this paper. However, this answer raises another question, "After all, is priority-share policy needed?". As we have seen in this paper, the distinctive-priority analysis can be performed with little overheads in terms of needed virtual channels, and we believe that the future real-time-oriented interconnect mediums will provide sufficient virtual channels to afford per-packet distinctive priorities and avoid the priority-share policy and the pessimism related to it.

Furthermore, link bandwidths were recognised as the bottleneck of the system. As this characteristic is equally important in high-performance and general-purpose computing, which are the main drivers for the advancements in interconnect mediums, we believe that link bandwidths are going to continue their increase in future years, which will be also appreciated from the real-time perspective.

6. CONCLUSIONS

As many-core platforms slowly but steadily pave their path into the real-time embedded domain, a quest for appropriate interconnect architectures still continues with an ever increasing importance. NoCs with flit-level preemptions via virtual channels have been proposed as one possible solution to introduce priority-awareness and real-time concepts into interconnect mediums. Consequently an adequate analysis was proposed to compute the worst-case traversal delays of individual packets.

In this paper we took a holistic approach and tried to estimate the "gap" between the existing many-core platforms and hypothetical many-core platforms suitable for the real-time domain, in other words, to investigate how (un)realistic are the requirements to make platforms suitable for the real-time analysis. In that vain we proposed a technique which utilises an existing feature of the SCC platform in order to significantly reduce the number of needed virtual channels. The experiments show that currently available many-core platforms are not far away from the "ideal" platforms required by the real-time domain, and we believe that in future years this difference will diminish. Furthermore, we proposed a three-staged mapping process which employs the existing analysis extended with the introduced resource reduction technique and derives a workload mapping plan which provides guarantees that all packets meet their deadlines. The mapping process served as a framework to explore the influence of different platform characteristics on the derived guarantees. This helped us to (i) recognise the bottleneck of the system, and (ii) minimise the resources of the platform such that the guarantees still hold. The second fact can be especially valuable for system designers, who can test different platform characteristics when mapping a given

workload, in order to design/select the platform which fulfils its purpose with as few resources as possible. This can reduce design costs, but also save the money both when buying the platform and through reduced power consumption.

As directions for future work, we see several possibilities. We plan to extend the approach to consider the on-core schedulability and positions of individual tasks when mapping the workload, which was in this work simplified to the level of mega-tasks. A mapping approach can be extended to take into account and minimise power consumption on cores as well as within the network. Finally, we plan to adapt the approach for specific workloads, e.g. streaming applications, dataflow applications.

7. ACKNOWLEDGMENTS

We would like to thank Dr Leandro Soares Indrusiak from the University of York, UK for valuable comments and discussions which helped us while delivering this paper.

8. REFERENCES

[1] H. I. Ali, L. M. Pinho, and B. Akesson. Critical-Path-First Based Allocation of Real-Time Streaming Applications on 2D Mesh-Type Multi-Cores. In *19th RTCSA*, Aug 2013.

[2] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *The Comp. J.*, 35(1):70 –78, jan 2002.

[3] B. Bessette, R. Salami, R. Lefebvre, M. Jelinek, J. Rotola-Pukkila, J. Vainio, H. Mikkola, and K. Jarvinen. The adaptive multirate wideband speech codec (amr-wb). *Trans. Speech & Audio Processing*, 10(8):620–636, 2002.

[4] K. Bletsas and B. Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. In *30th RTSS*, 2009.

[5] W. Dally. Virtual-channel flow control. *Trans. Parall. & Distr. Syst.*, 3(2):194 –205, Mar 1992.

[6] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *Trans. Computers*, 1987.

[7] D. Dasari, B. Nikolić, V. Nelis, and S. M. Petters. A tighter analysis of the worst-case end-to-end communication delays in massive multicores. In *WIP Session 33rd RTSS*, 2012.

[8] T. Ferrandiz, F. Frances, and C. Fraboul. A method of computation for worst-case delay analysis on spacewire networks. In *IEEE Int. Symp. Industrial Emb. Syst.*, 2009.

[9] T. Ferrandiz, F. Frances, and C. Fraboul. Using network calculus to compute end-to-end delays in spacewire networks. *SIGBED Rev.*, 2011.

[10] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *8th ASPDAC*, 2003.

[11] Intel. *The Single-chip Cloud Computer*. `www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html`.

[12] Intel. *Single-Chip-Cloud Computer*. `www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-article.pdf`.

[13] S. Kato, N. Yamasaki, and Y. Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *21st ECRTS*, 2009.

[14] N. K. Kavaldjiev and G. J. M. Smit. A survey of efficient on-chip communications for soc. In *Symp. Emb. Syst.*, 2003.

[15] B. Kim, J. Kim, S. Hong, and S. Lee. A real-time communication method for wormhole switching networks. In *1998 Int. Conf. Parall. Processing*, Aug 1998.

[16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[17] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 1973.

[18] P. Mesidis and L. Indrusiak. Genetic mapping of hard real-time applications onto noc-based mpsocs – a first approach. In *6th ReCoSoC*, 2011.

[19] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *The Comp. J.*, 26, 1993.

[20] A. Racu and L. Indrusiak. Using genetic algorithms to map hard real-time on noc-based systems. In *7th ReCoSoC*, Jul 2012.

[21] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Int. Symp. Netw.-on-Chip*, 2008.

[22] Z. Shi and A. Burns. Real-time communication analysis with a priority share policy in on-chip networks. In *21st ECRTS*, 2009.

[23] Z. Shi and A. Burns. Schedulability analysis and task mapping for real-time on-chip communication. *Real-Time Syst. J.*, 2010.

[24] H. Song, B. Kwon, and H. Yoon. Throttle and preempt: a new flow control for real-time communications in wormhole networks. In *1997 Int. Conf. Parall. Processing*, Aug 1997.

[25] Tilera. *TILE64™ Processor*. `www.tilera.com/products/processors/TILE64`.