Real-time application mapping for many-cores using a limited migrative model

Borislav Nikolic´ · Stefan M. Petters

Abstract

Many-core platforms are an emerging technology in the real-time embedded domain. These devices offer various options for power savings, cost reductions and contribute to the overall system flexibility, however, issues such as unpredictability, scalability and analysis pessimism are serious challenges to their integration into the aforementioned area. The focus of this work is on many-core platforms using a *limited migrative model* (LMM). LMM is an approach based on the fundamental concepts of the *multi-kernel* paradigm, which is a promising step towards scalable and predictable many-cores. In this work, we formulate the problem of real-time application mapping on a many-core platform using LMM, and propose a three-stage method to solve it. An extended version of the existing analysis is used to assure that derived mappings (i) guarantee the fulfilment of timing constraints posed on worst-case communication delays of individual applications, and (ii) provide an environment to perform load balancing for e.g. energy/thermal management, fault tolerance and/or performance reasons.

Keywords

Real-time systems, Embedded systems, Multiprocessors, Many-core systems, Worst-case analysis, Limited migrative model

1 Introduction

Development trends in the semiconductor area reached the stage where further processing power enhancements related to single-core devices are no longer affordable (Lundstrom 2003). In order to meet the demands for more powerful computational devices, chip manufacturers took a design paradigm shift (Intel 2014; Tilera 2014), and started

integrating multiple cores within a single chip. Nowadays, platforms consisting of several cores (*multi-cores*) and more than a dozen cores (*many-cores*) became commonplace in many scientific areas (e.g. high-performance computing), while they are still an emerging technology in some other domains, like real-time embedded computing. This work focuses on the latter category.

In order to integrate many-core platforms into the real-time embedded domain, an OS paradigm is needed which allows the system designer to exploit the full potential of the underlying hardware resources, and yet assures predictability (Zimmer and Mueller 2012) and scalability (Baumann et al. 2009; Kumar et al. 2011), which are essential prerequisites for deriving real-time guarantees. We elaborate on a model which we call the *limited migrative model*, *LMM* hereafter (Nikolic´ and Petters 2012). It builds on top of the foundations of the *multi-kernel* paradigm (Baumann et al. 2009), which is a novel approach in the OS design (Baumann et al. 2009; Wentzlaff and Agarwal 2009; Le and West 2014), and a promising step towards scalable and predictable many-cores.

The *Network-on-Chip* (NoC) architecture (Benini and De Micheli 2002) became a predominant interconnect medium in many-cores, due to its scalability potential (Kavaldjiev and Smit 2003). On such platforms, a *wormhole switching* technique (Ni and McKinley1993) is widely applied (Intel 2014; Tilera 2014), because of its good throughput and has small buffering requirements (Kavaldjiev and Smit 2003).

Contribution In this work, we focus on the mapping of real-time application workload onto a NoC-based, wormhole-switched many-core platform, using LMM. First, we formulate the problem of real-time application mapping on such a system. Then, we present an extension to the existing analysis, which assures that all timing constraints posed on worst-case communication delays of individual applications are met. Finally, we propose a three-stage heuristics-based application mapping procedure, which utilises the aforementioned analysis during the mapping process. The novelty of our work is reflected by the fact that we study LMM, which is a model that allows application migrations as a means to perform load balancing for e.g. energy/thermal management, fault tolerance and/or performance reasons. The biggest contribution of our work is that it merges real-time concepts with those from the embedded and high-performance computing areas, resulting in a model which allows dynamically changing system behaviour, while still providing hard real-time guarantees.

2 LMM concepts and motivation

Unpredictability, scalability and analysis pessimism are some of the reasons that make the real-time analysis of many-cores a challenging subject. The existing state-of-the-art methodologies addressing many-cores from the real-time perspective can be broadly classified into two categories: *non-migrative approaches* and *migrative approaches*. First, we define these categories, and then position LMM in that context.

Non-migrative approaches are in the scheduling theory also known as *fully partitioned approaches*. The workload is statically assigned to cores at design time. Applications are migrationless and their execution is organised by independent single-core schedulers residing on each core and utilising one of existing single-core scheduling algorithms (e.g. Liu and Layland 1973). These approaches are inflexible and can

be very inefficient in scenarios where substantial load changes occur, and/or where run-time load balancing is required.

Migrative approaches are additionally divided into *semi-partitioned approaches*, *global approaches* and *hybrid approaches*. The first group (Bletsas and Andersson 2009; Kato et al. 2009) also assumes a static assignment of each application to a particular core (or cores if it migrates). However, a migrative application also obeys design-time decisions regarding its execution, i.e. it always executes the prescribed fraction of work on each of its cores. Thus, these methods are very similar to fully partitioned approaches, hence all conclusions regarding inflexibility to perform run-time load balancing also hold in this context. Conversely, global approaches (Baker et al. 2006; Baruah and Baker 2008) allow every application to migrate to any core within the platform. However, this amount of flexibility comes at a price: due to the necessity to maintain global structures (e.g. ready queue) within a centralised entity, scalability issues arise, and serious challenges occur when attempting implementations (Bastoni et al. 2010). Finally, hybrid approaches (Calandrino et al. 2007) (also known as *clustered approaches*) group cores into disjoint clusters, map distinctive subsets of applications to each cluster, and apply a global scheduling policy within each cluster. These approaches offer both runtime load balancing and scalability, however, they are inefficient in scenarios where migrations are driven by fault tolerance or energy/thermal management, where inter-cluster migrations are a necessary option.

LM M (Nikolic´ and Petters 2012) exploits and extends the concepts of the multi-kernel paradigm and poses a constraint that each application can execute only on a subset of cores, which are decided at design-time. Unlike in clustered approaches, an application is not tied to a particular cluster, but can be mapped to an arbitrary set of cores, which allows performing migrations driven by the aforementioned reasons. During runtime, an application can migrate between candidate cores and execute on any of them. Yet, the greatest distinction of *LM M* from all the existing approaches is that release/migration decisions are explicitly detached from scheduling decisions, and are made by the applications *themselves*, not by scheduler instances. That is, the application decides on which core it will release its job, while a local kernel on that core is responsible to schedule it in a single-core fashion. This contributes to the scalability of the approach, and yet gives the possibility to perform runtime load balancing. More details about *LM M* are given in Sect. 4.2.

3 Related work

Application mapping for many-cores has been one of the most investigated topics over the last decade, thus resulting in a vast amount of works. In this section we give a brief overview of the state-of-the-art methods by mentioning only works that are relevant for our cause. An interested reader may consult a comprehensive survey of Sahu and Chattopadhyay (2013), which covers scientific publications related to the entire application mapping topic.

The problem of application mapping is equivalent to the quadratic assignment problem, which is NP-Hard, hence searching for the optimal solution can be prohibitively expensive even for small NoC platforms, e.g. 4×4 cores (Hu and Marculescu 2003). Therefore, the existing approaches are predominantly heuristics-based. Lei and

Kumar (2003) assumed different processor types and developed a two-stage genetic algorithm, where the workload is firstly mapped to a specific processor type and in the second pass to a particular processor. Moein-darbari et al. (2009) use a modification of the aforementioned heuristics, called the chaos-genetic algorithm. By employing the branch-and-bound technique Hu and Marchulescu (2003) investigate mappings which minimise the power consumption within the network. The authors present a power model to calculate the energy spent by the NoC infrastructure, which is used as an objective function to evaluate derived mappings. The concept of minimising the energy consumption was further extended to include performance enhancements (Chou et al. 2008), network contentions (Chou and Marculescu 2008) and runtime mappings (Chou and Marculescu 2007). Murali and De Micheli (2004) elaborate on bandwidth constraints and minimise the average communication delay, while Hung et al. (2004) study thermal-aware placements. Marcon et al. (2005) are the first to consider the ordering and dependencies among traffic messages, while Srinivasan and Chatha (2005) introduce per-message latency constraints. Ascia et al. (2004) present an approach where not one, but a group of mappings is considered (*Pareto mappings*), so as to derive a solution to the multi-objective approach. Kreutz et al. (2005) explore the same topic for interconnect topologies other than NoC, while Hu and Marculescu (2003) exploit the routing flexibility in order to reduce the routing energy consumption and improve the performance.

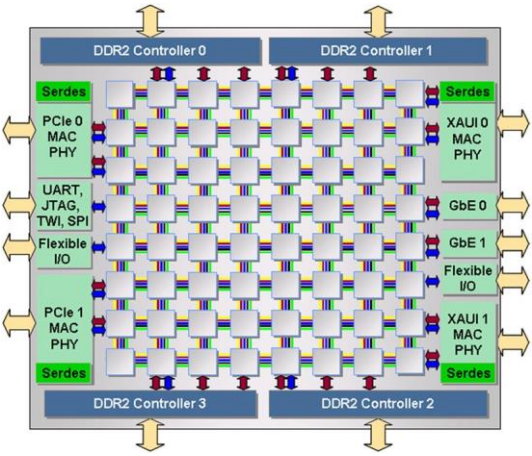
Assuming wormhole-switched priority-preemptive NoCs, Shi and Burns (2008, 2009) propose two worst-case communication delay analyses. The former analysis was combined with genetic algorithms with the objective of mapping hard real-time applications (Mesidis and Indrusiak 2011; Racu and Indrusiak 2012), while the latter approach was further combined with a priority assignment algorithm with the same aim (Shi and Burns 2010). Note that the works mentioned in this paragraph are the most relevant to our approach, as they study the application-mapping problem from the real-time perspective, where the main emphasis is on deriving a mapping such that all temporal constraints posed on communication delays are always fulfilled, even under the worst-case conditions. Also note that these approaches assume that each application is statically assigned to a specific core at design-time, and does not have the possibility to migrate (see fully partitioned approaches in Sect. 2), while we consider LMM, which offers flexibility to perform run-time load balancing via migrations. In the previous work, Nikolic´ et al. (2014) introduced techniques to make the LMM traffic deterministic, and subsequently proposed the worst-case communication delay analysis. In this work, we modify the existing analysis, so as to reduce its computational complexity, and go one step beyond by proposing a three-stage heuristics-based method to map hard real-time applications on NoCs using LMM.

4 Model

4.1 Platform

The platform under consideration is a homogeneous, NoC-based many-core system, consisting of $m \times n$ tiles, which are connected via a 2-D mesh interconnect. Each tile

Fig. 1 TILE64 processor

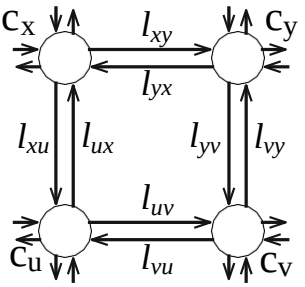


contains a single core and a single router. The data transfer over the mesh is performed by employing a deterministic, dimension-ordered *XY routing policy*, which is deadlock and livelock free (Hu and Marculescu 2003). With this policy, a packet firstly travels on the x-axis, and upon reaching the x-coordinate of the destination, traverses along the y-axis. Furthermore, a packet transfer is performed with the *wormhole switching* technique (Ni and McKinley 1993). In this regime, each data packet is, prior to sending, divided into small fixed-size elements called *flits*. A flit is a basic transferable unit across the NoC interconnect. All flits constituting one packet are sequentially sent, where the header flit establishes the path, while the rest follow in a pipeline manner. Each packet is prioritised and there exist virtual channels (Dally 1992; Dally and Seitz 1987). A virtual channel is implemented as an additional buffer within every port of every router, and it is used to enforce flit-level preemptions (Song et al. 1997). The number of needed virtual channels is equal to the maximum number of contentions for any port, which guarantees that each packet will have an available virtual channel in every port along its path (Nikolic´ et al. 2013). In this paper, the terms *packet* and *message* are used interchangeably.

Note that our target platform is very similar to some existing many-core platforms, such as the Single-Chip-Cloud Computer (Intel 2014) and the TILE64 family of processors (Tilera 2014) (see Fig. 1). Specifically, both the mentioned platforms contain a 2-D mesh interconnect, utilise the XY routing policy and perform the data transfer with the wormhole switching technique. Moreover, the SCC interconnect contains 8 virtual channels, which are on average sufficient to accommodate up to 400 different priority levels on a 10×10 NoC (Nikolic´ et al. 2013).

The platform $\Pi(C, L)$ is described by a direct graph (Fig. 2), where vertices $C = \{c_1, c_2, \ldots, c_{m \cdot n}\}$ represent cores, and edges $L = \{l_{ij}\}$ denote physical links of the interconnect. Every existing link $l_{ij} = (c_i, c_j, b_{ij}) \mid \{i, j\} \in C$ between the cores c_i and c_j is characterised with b_{ij}, which describes its physical characteristics (i.e. the bandwidth). In this work we assume that all links have identical characteristics $b_{ij} = b = \frac{size(f)}{d_r + d_l}$; the bandwidth is expressed as the size of one flit $size(f)$,

Fig. 2 Platform n



divided by the time needed to transfer it between two neighbouring cores $d_r + d_l$. The terms d_r and d_l correspond to the latency of one flit to traverse the router and the link, respectively.

4.2 Software layers in *LM M* (Nikolic´ and Petters 2012)

As mentioned in Sect. 1, *LM M* builds on top of the multi-kernel paradigm, which means that each core runs an independent kernel instance. Kernels communicate with each other and constitute the basic communication infrastructure. Each kernel exposes some of its functionalities to applications located on its core via system calls. Applications invoke system calls in order to interact with other applications residing on the same or other cores. Furthermore, each kernel performs the scheduling on its core in a single-core fashion. Practical examples of multi-kernel OSs are *Barrelfish* (Baumann et al. 2009), *fos* (Wentzlaff and Agarwal 2009) and *Quest-V* (Li and West 2014).

We consider a sporadic application-set with constrained deadlines. Once during every inter-arrival period, an application releases a job, which has to complete before the new inter-arrival period begins. As discussed (Sect. 2), a job can be released only on a subset of cores, which are, for each application, selected at design-time. Once released, a job has to complete on the selected core, it cannot migrate (job-level migrations are not allowed).

Unlike in traditional real-time approaches, in *LM M* the application *itself* makes release/migration decisions. In order to do that, each application uses entities called *dispatchers*. An application has a dispatcher on each of its candidate cores. Before the job release, each dispatcher of that application gets the information from its local kernel whether the guarantees can be provided regarding the execution of that job on that core. That is, the dispatcher checks whether the job can be safely scheduled on its core without missing a deadline. Recall, that each kernel schedules the workload on its core in a single-core fashion. Once each dispatcher gets that information from its kernel, dispatchers of the same application (each located on a different core), communicate with each other. The purpose of their communication is to elect one dispatcher, called the *master dispatcher*, which will subsequently release the job on its core on behalf of the entire application. The aforementioned inter-dispatcher communication is called the *agreement protocol* (Nikolic´ and Petters 2012). In this work, we assume that the

Fig. 3 *LM M*

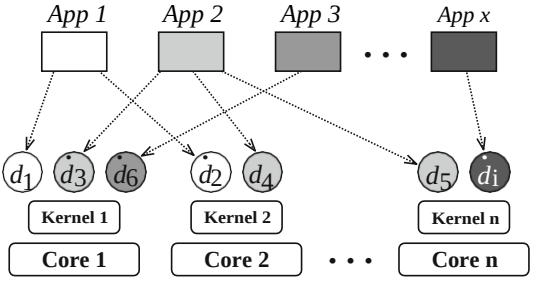
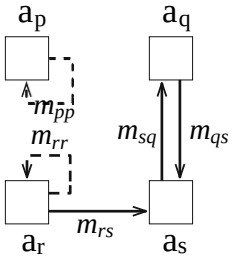


Fig. 4 Workload *E*



applications are single-threaded, implemented as tasks, therefore at any time instant there can be only one master per application.

When the new inter-arrival period begins, the master is responsible to initiate the new instance of the agreement protocol, so as to elect a core (dispatcher) for the release of the next job. The other dispatchers that participate in the protocol are called the *slave dispatchers*. If the outcome of the protocol is that the newly elected dispatcher is not the same as the previous master, then the migration occurs; the previous master becomes the slave, while the newly elected dispatcher becomes the master. Additionally, the execution context has to be transferred from the old to the new master. The protocol execution is termed the *intra-application communication*. Perceived from the application's perspective, its dispatchers exchange one master token, thus a master is only a temporary role for a dispatcher. We call this property the *master volatility*, and it has several implications which are covered in Sect. 5.4. Figure 3 gives a graphical representation of $LM\ M$, where current master dispatchers can be distinguished by dots over their names.

Besides protocols, applications can also communicate with each other for e.g. synchronisation or data-sharing purposes. We term this process the *inter-application communication*, and it is performed by an exchange of messages between current master dispatchers of interacting applications.

Execution workload $E\ (A, M)$ is described by a direct graph (Fig. 4), where vertices $A = \{a_1, a_2, ..., a_{x-1}, a_x\}$ denote the applications comprising the application-set, and edges $M = m_{ij}$ symbolise the intra- and inter-application traffic. An application $a(P_a, T_a, W_a, I_a, D_a, M_a) \in A$ has a unique priority P_a, a minimum inter-arrival period T_a, a temporal constraint on its worst-case communication delay $W_a \leq T_a$ (explained in Sect. 4.3), an importance of its distributed mapping I_a

(explained in Sect. 6), a set of dispatchers D_a and a set of messages M_a. A message $m_{ij} = (d_i, d_j, P_{ij}, size, path, nhops) \{d_i, a_j\} \in A$ between a source dispatcher d_i and a destination dispatcher d_j is characterised by a priority P_{ij}, the amount of exchanged data $size(m_{ij})$, and a traversed path expressed by (i) a set of links $path(m_{ij})$, (ii) a number of hops $nhops(m_{ij})$. If m_{ij} is the intra-application message, then d_i and d_j belong to the same application a, thus $P_{ij} = P_a$ (dashed lines in Fig. 4). Conversely, if m_{ij} is the inter-application message, then d_i and d_j are the current masters of the interacting applications a_i and a_j, thus $P_{ij} = P_i$ (solid lines in Fig. 4), i.e. an inter-application message always inherits the priority of a sender application.

4.3 Work objectives

The goal of this work is to propose an application mapping method which finds a *feasible mapping* of a given application workload onto a given NoC platform (n). A mapping is feasible if all applications are *feasible*. An application a is feasible if its worst-case communication delay, termed $del(a)$, is less than, or equal to its timing constraint, i.e. $del(a) \leq W_a$. The worst-case communication delay of an application is equal to the sum of the worst-case delays of its intra-application and sent inter-application messages. The secondary objective is to provide a feasible mapping, such that the applications can perform migrations across spatially distributed dispatchers. This objective is motivated with the fact that far migrations are the efficient means to implement energy/thermal management, and increase the resilience to core/cluster malfunctions.

The mapping process consists of several stages. The first is to map the application-set on the given platform (Sect. 7). The second is to perform the analysis and check if the derived mapping is feasible (Sect. 5). Finally, assuming that the feasible mapping was found, the goal is to optimise it, with respect to the secondary objectives (Sect. 6). Note that these three stages are not necessarily performed sequentially, but can be interleaved.

5 Worst-case communication delay analysis

5.1 Preliminaries

In this subsection, we will briefly cover the components that constitute the worst-case traversal delay of a single message m, termed $del(m)$.

5.1.1 Isolation delay

Each message m traverses an XY-routed path between the source and the destination dispatcher. The delay of a message when traversing in isolation is in the literature known as the *basic network latency* (Duato et al. 2002) (Eq. 1). It is equal to the time it takes a header flit to establish the path and reach the destination, augmented by the

transfer time of the rest of the flits.

$$del_I(m) = nhops(m) \cdot (d_r + d_l) + \left\lceil \frac{size(m)}{size(f)} \right\rceil \cdot d_l \qquad (1)$$

5.1.2 Lower priority blocking

Due to the flit-level preemptions, *m* can be blocked by the lower-priority traffic. The worst-case scenario occurs if *m* reaches the router just at the moment when a lower-priority message started its transfer, hence *m* can be blocked at most for the duration of one flit transfer time before it preempts. This scenario can occur within each router on the path of *m* (Eq. 2).

$$del_B(m) = nhops(m) \cdot (d_r + d_l) \qquad (2)$$

5.1.3 Same priority blocking

Messages of the same priority share the same virtual channel and may compete for the same physical resource if they have a common part of the path. This can cause complex blocking scenarios (Shi and Burns 2009, 2010), described below.

Definition 1 (*Directly blocking message*) If a message $m^{\mid}$ has the same priority as the message under analysis *m*, and shares a part of the path with *m*, it is considered as a *directly blocking message* of *m* and belongs to the set $M_{DB}(m)$. Formally:

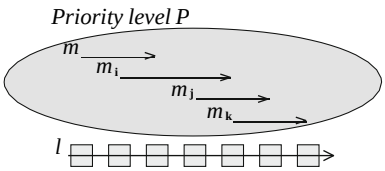
$$\forall m^{\mid} \in \boldsymbol{M} : m^{\mid} \neq m \wedge P_{m^{\mid}} = P_m \wedge path(m^{\mid}) \cap path(m) \neq \emptyset \Rightarrow m^{\mid} \in \boldsymbol{M}_{DB}(m)$$

Definition 2 (*Indirectly blocking message*) If a message $m^{\mid\mid}$ has the same priority as the message under analysis *m* and does not share a part of the path with *m*, but shares it with another message $m^{\mid}$ which is either directly or indirectly blocking message of *m* (recursive definition), then $m^{\mid\mid}$ is considered as an *indirectly blocking message* of *m* and belongs to the set $M_{IB}(m)$. Formally:

$$\forall m^{\mid\mid} \in \boldsymbol{M} : P_{m^{\mid\mid}} = P_m \wedge path(m^{\mid\mid}) \cap path(m) = \emptyset \ \wedge$$
$$\exists m^{\mid} \in (\boldsymbol{M}_{DB}(m) \cup \boldsymbol{M}_{IB}(m)) \wedge path(m^{\mid\mid}) \cap path(m^{\mid}) \neq \emptyset \Rightarrow m^{\mid\mid} \in \boldsymbol{M}_{IB}(m)$$

A message under analysis *m* can suffer the blocking from its directly blocking message $m^{\mid} \in M_{DB}(m)$, but also from its indirectly blocking message $m^{\mid\mid} \in M_{IB}(m)$. We explain this with an illustrative example given in Fig. 5, where rectangles depict routers, and arrows symbolise messages (the arrow tail represents the source and the arrow head represents the destination of the message). All the messages have the same priority and traverse some links of the path *l*. By Definitions 1–2, the blocking relationships of the message *m* are as follows: $M_{DB}(m) = \{m_i\}$, $M_{IB}(m) = \{m_j, m_k\}$. It is visible that *m* can suffer the blocking from m_i, which can in turn suffer the blocking from m_j, which can in turn suffer the blocking from m_k. Thus, *m* can suffer the blocking from each of these messages $\{m_i, m_j, m_k\}$. The maximum delay caused by

Fig. 5 Example of blocking



one traversal of a blocking message (either directly or indirectly) is equal to the sum of its isolation delay and its lower priority blocking delay, computed by Eqs. 1 and 2, respectively.

Note, by considering that each message has to complete its traversal within a temporal constraint posed on its application $W_a \leq T_a$, messages belonging to different inter-arrival periods of the same application cannot exist concurrently and hence cannot cause the blocking.

5.1.4 Interference

Additionally, *m* can suffer the interference from the higher-priority messages.

Definition 3 (*Directly interfering message*) If a message m^{I} has a higher priority than the message under analysis *m*, and shares a part of the path with *m*, it is considered as a *directly interfering message* of *m* and belongs to the set $M_{DI}(m)$. Formally:

$$\forall m^{I} \in M : P_{m^{I}} > P_m \wedge path(m^{I}) \cap path(m) \neq \emptyset \Rightarrow m^{I} \in M_{DI}(m)$$

m can be preempted by every $m^{I} \in M_{DI}(m)$ and consequently suffer the interference. The maximum delay caused by one traversal of m^{I} is equal to the sum of its isolation delay and its lower-priority blocking delay.

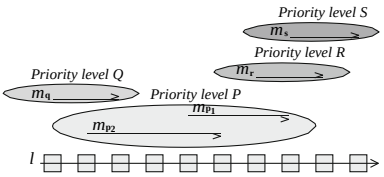
Definition 4 (*Indirectly interfering message*) If a message m^{II} has a higher priority than the message under analysis *m* and does not share a part of the path with *m*, but shares it with another message m^{I} which is either directly or indirectly blocking message of *m*, then m^{II} is considered as an *indirectly interfering message* of *m* and belongs to the set $M_{II}(m)$. Formally:

$$\forall m^{II} \in M : P_{m^{II}} > P_m \wedge path(m^{II}) \cap path(m) = \emptyset \ \wedge$$
$$\exists m^{I} \in \{M_{DB}(m) \cup M_{IB}(m)\} \wedge path(m^{II}) \cap path(m^{I}) \neq \emptyset \Rightarrow m^{II} \in M_{II}(m)$$

A message $m^{II} \in M_{II}(m)$ can preempt a message m^{I} which is already directly or indirectly blocking *m*, hence additionally contributing to the total delay of *m*. The maximum interference caused to *m* by one traversal of m^{II} is equal to the sum of its isolation delay and its lower-priority blocking delay.

An illustrative example of the aforementioned inter-message relationships is given in Fig. 6. Assuming the priorities $P_s > P_r > P_q > P_p$, the inter-message relationships of the message under analysis m_{p1} are as follows: $M_{DB}(m_{p1}) = \{m_{p2}\}$, $M_{DI}(m_{p1}) = \{m_r\}$, $M_{II}(m_{p1}) = \{m_q\}$. Observe, by definition, m_s does not

Fig. 6 Example of interference



belong to $\boldsymbol{M}_{II}(m_{p1})$, even though it is directly interfering with m_r, which belongs to $\boldsymbol{M}_{DI}(m_{p1})$. Due to the existence of per-priority virtual channels, m_s cannot directly cause the interference to m_{p1} (Shi and Burns 2008). However, m_s can cause the interference to m_r, influence its occurrence patterns and in that way passively contribute to the interference that m_r causes to m_{p1}. That is, m_s can cause two consecutive appearances of m_r to be distanced by less than the inter-arrival period. Thus, when computing the worst-case delay of m_{p1}, considering periodic appearances of m_r would be an unsafe assumption (Shi and Burns 2008). Note that the definition of indirect interferences in our work excludes m_s, while the other works include it (Shi and Burns 2008, 2009, 2010).

The worst-case traversal delay of m_{p1} is equal to the sum of (i) the isolation delay, (ii) the lower-priority blocking, (iii) the same-priority blocking and (iv) the higher-priority interference. Note that the messages may additionally suffer the *release jitter* (Shi and Burns 2010), which is defined as a maximum deviation of successive packet releases from its period. In this work, the release jitters are assumed to be zero.

After defining these components, in Sect. 5.2 we describe the existing method to compute the worst-case delay of a single message. Then, we propose a modification to reduce the computational complexity (Sect. 5.3). Finally we extend the analysis (i) to consider the entire application instead of the individual message and (ii) to be applicable to *LMM* (Sect. 5.4).

5.2 Existing analysis to compute worst-case message delays (Shi and Burns 2009)

5.2.1 Description

Shi and Burns (2009) proposed a method called the *priority share policy*, which computes the worst-case delays of messages, assuming that some might share the same priority. The model is constrained to scenarios where a deadline of each message is less than, or equal to its minimum inter-arrival period, which is also the case in our model, but expressed on an application-level ($\forall a \in A : W_a \leq T_a$). Due to the direct and indirect blockings, as well as the direct and indirect interferences, complex interference patterns might occur, as depicted in the previously presented examples (Figs. 5, 6). To overcome this problem, the authors proposed to group the same-priority messages into a new entity called the *composite message* $\mathring{m}$. Note that only the messages of the priority under analysis are grouped within a composite message, while messages with other priorities are still considered separately. Let $\boldsymbol{M}_C(\mathring{m})$ be a set of messages constituting $\mathring{m}$, i.e. all messages that have the same priority $P_{\mathring{m}}$. The isolation delay of the composite message $\mathring{m}$ is equal to the sum of the isolation latencies of all messages constituting $\boldsymbol{M}_C(\mathring{m})$ (Eq. 3).

$$del_I(\mathring{m}) = \sum_{\forall m \in \mathcal{M}_C(\mathring{m})} del_I(m) \tag{3}$$

Similarly, the lower-priority blocking of the composite message is equal to the sum of the lower-priority blockings suffered by its messages (Eq. 4).

$$del_B(\mathring{m}) = \sum_{\forall m \in \mathcal{M}_C(\mathring{m})} del_B(m) \tag{4}$$

As the messages with the same priority $P_{\mathring{m}}$ are all grouped within $\mathring{m}$, this implies that $\mathring{m}$ cannot suffer the same-priority blocking. That is, there are no other messages that are sharing the same priority with $\mathring{m}$ and are directly or indirectly blocking it (see Definitions 1–2). This further implies that the indirect interference cannot exist, as it requires at least one blocking message (see Definition 4). Hence, only direct interferences are possible. This significantly simplifies the analysis.

A message is a directly interfering message of $\mathring{m}$ if it can cause the direct interference to any message from $\mathcal{M}_C(\mathring{m})$, i.e. $\mathcal{M}_{DI}(\mathring{m}) = \bigcup_{\forall m \in \mathcal{M}_C(\mathring{m})} \mathcal{M}_{DI}(m)$. The maximum interference that $\mathring{m}$ can suffer within a time interval t is equal to the sum of the interferences caused by each of the directly interfering messages (Eq. 5). The individual terms are obtained by multiplying the maximum number of occurrences of a higher priority message $m^{\downarrow}$ within t, with the interference caused by a single occurrence of $m^{\downarrow}$. The term $del(m^{\downarrow})$ represents the worst-case delay of $m^{\downarrow}$, which is, due to the recursive notion, here used as a constant and explained later. The additional term in the ceiling brackets: $del(m^{\downarrow}) - del_I(m^{\downarrow})$ is called the *interference jitter* – $del_J(m^{\downarrow})$, and it accounts for the potentially non-periodic occurrence pattern of $m^{\downarrow}$. That is, due to its higher-priority interference, two consecutive occurrences of $m^{\downarrow}$ might be separated by less than $T_{a_{m^{\downarrow}}}$, and that case has to be covered. Specifically, the first occurrence of $m^{\downarrow}$ can be delayed by at most $del_J(m^{\downarrow}) = del(m^{\downarrow}) - del_I(m^{\downarrow})$. Note that $del_J(m^{\downarrow})$ has a non-zero value only if $m^{\downarrow}$ suffers direct interference from some other message that is not directly interfering with m. This is a well known issue in the wormhole switching, and for a more detailed explanation the reviewer is advised to consult the works of Shi and Burns (2008, 2009, 2010).

$$del_N(\mathring{m}, t) = \sum_{\forall m' \in \mathcal{M}_{DI}(\mathring{m})} \left\lceil \frac{t + \overbrace{del(m') - del_I(m')}^{del_J(m')}}{T_{a_{m'}}} \right\rceil \cdot \left(del_I(m') + del_B(m')\right) \tag{5}$$

The worst-case delay of a composite message $\mathring{m}$ (Eq. 6) is equal to the sum of (i) the isolation latency, (ii) the lower-priority blocking, (iii) the interference, of all messages constituting it. Note that in the interference component, the term t is substituted with the worst-case delay, thus giving it a recursive notion. Equation 6 is solved iteratively, until reaching a fixed-converging point (if one exists).

$$del(\mathring{m}) = \overbrace{del_I(\mathring{m})}^{\text{isolation}} + \overbrace{del_B(\mathring{m})}^{\text{blocking}} + \overbrace{del_N(\mathring{m}, del(\mathring{m}))}^{\text{interference}} \tag{6}$$

Table 1 Example of messages

Message	Priority	del_I	W	T
m_{p1}	P_p	2	20	25
m_{p2}	P_p	2	20	25
m_q	$P_q > P_p$	2	9	9
m_r	$P_r > P_q$	2	6	6
m_s	$P_s > P_r$	2	10	10

Note that this analysis holds only under the assumption that the same-priority messages are treated in FIFO order. This assumption assures that any two same-priority messages, even though they can have different periods, can directly block each other only once. As will be discussed later, in our approach such a restriction is not necessary.

Upon obtaining the worst-case delay of $\mathring{m}$, each message constituting it assumes the same value, i.e. $\forall m \in M_C(\mathring{m}) : del(m) = del(\mathring{m})$. In this way, the worst-case delay is computed for every priority level and every message within the system. Notice two things: (i) the approach can be pessimistic, as it groups the same-priority messages, while many of them do not influence each other and can be treated independently and (ii) assuming that the same-priority messages belong to the same application (i.e. have the same deadline and the inter-arrival period), the obtained value represents not only the worst-case delay of a single message, but the worst-case delay of the joint traversal of all the same-priority messages. The second fact is further exploited in Sect. 5.4.

5.2.2 Numerical example

Consider the example given in Fig. 6 with the message parameters given in Table 1, where the message m_{p1} is under analysis, and the goal is to compute its worst-case delay - $del(m_{p1})$.

For the clarity of the example we take a simplifying assumption that the lower-priority blocking does not exist ($del_B = 0$). We start by computing the worst-case delay of the highest-priority message m_s. Its delay is equivalent to its isolation latency, $del(m_s) = del_I(m_s) = 2$. The message m_r can suffer the interference from m_s, so its delay is:

$$del(m_r) = del_I(m_r) + \left\lceil \frac{del(m_r) + del_J(m_s)}{T_{a_{m_s}}} \right\rceil \cdot del_I(m_s) = 2 + 2 = 4$$

The message m_q cannot suffer the interference, so its delay is equal to its isolation latency, $del(m_q) = del_I(m_q) = 2$.

The messages m_{p1} and m_{p2} share the same priority P_p, hence are grouped within m°_p. The isolation latency of m°_p is equal to the sum of the isolation latencies of both messages, $del_I(m^{\circ}_p) = del_I(m_{p1}) + del_I(m_{p2}) = 4$. Moreover, m°_p has two interfering messages: m_q and m_r. The worst-case delay of m°_p is:

$$del(\hat{m_p}) = del_I(\hat{m_p}) + \sum_{\forall m' \in [m_q, m_r]} \left\lceil \frac{del(\hat{m_p}) + del_J(m')}{T_{a_{m'}}} \right\rceil \cdot del_I(m')$$

$$del(\hat{m_p})^0 = 4 + \left\lceil \frac{0}{9} \right\rceil \cdot 2 + \left\lceil \frac{0+4-2}{6} \right\rceil \cdot 2 = 4 + 0 + 2 = 6$$

$$del(\hat{m_p})^1 = 4 + \left\lceil \frac{6}{9} \right\rceil \cdot 2 + \left\lceil \frac{6+4-2}{6} \right\rceil \cdot 2 = 4 + 2 + 4 = 10$$

$$del(\hat{m_p})^2 = 4 + \left\lceil \frac{10}{9} \right\rceil \cdot 2 + \left\lceil \frac{10+4-2}{6} \right\rceil \cdot 2 = 4 + 4 + 4 = 12$$

$$del(\hat{m_p})^3 = 4 + \left\lceil \frac{12}{9} \right\rceil \cdot 2 + \left\lceil \frac{12+4-2}{6} \right\rceil \cdot 2 = 4 + 4 + 6 = 14$$

$$del(\hat{m_p})^4 = 4 + \left\lceil \frac{14}{9} \right\rceil \cdot 2 + \left\lceil \frac{14+4-2}{6} \right\rceil \cdot 2 = 4 + 4 + 6 = 14$$

The worst-case delay of m_{p1} and m_{p2} is equal to the worst-case delay of the composite message, i.e. $del(m_{p1}) = del(m_{p2}) = del(m^{\circ}_p) = 14$.

This analysis faces two challenges. First, its computational complexity is significant and might not be acceptable for the application-mapping process, where the analysis has to be performed numerous times (see Sect. 5.3). Second, it suits the models in which all messages have their routes defined at design time, e.g. Shi and Burns (2010). However, under LMM, each application has the possibility to execute on multiple cores, hence routes of all messages related to one application will highly depend on which dispatcher is the current master at the moment of observation. In other words, the master volatility property causes non-deterministic message routes. This renders the analysis inapplicable to LMM (see Sect. 5.4). In the next two sections we describe the modifications of the existing analysis that are necessary in order to make it applicable to our cause - the application mapping assuming LMM.

5.3 Complexity reduction

5.3.1 Rationale

Notice that in the analysis presented in Sect. 5.2 the worst-case delay of a message directly depends on the worst-case delay of each of its interfering messages. Additionally, the worst-case delay of each of these messages also depends on the worst-case delay of their interfering messages. In the example presented in Sect. 5.2.2 and Fig. 6 the delay of m_{p1} and m_{p2} depends on the delay of m_r, while the delay of m_r depends on the delay of m_s. Therefore, any changes in the parameters of m_s (the position on the grid, the size, the priority, etc.) would require to recheck the feasibility of not only m_r, but also m_{p1} and m_{p2}. Thus, any manipulation with the parameters of a high-priority message, can cause a cascading effect and invoke the feasibility rechecks of many lower-priority messages. As a result, the entire feasibility rechecking process has an exponential complexity. However, during the application mapping process, messages can be subjected to frequent parameter manipulations with the objective of finding

a feasible mapping, hence, performing the analysis presented in Sect. 5.2 might be prohibitively expensive.

In order to reduce the complexity, we propose a modification to the existing analysis, such that it renders more pessimistic results, but significantly reduces the number of necessary feasibility rechecks. Specifically, we assume that the first occurrence of each interfering message $m^{\downarrow}$ can be delayed as much as possible such that its feasibility is preserved: $W_{a_{m^{\downarrow}}} - del_I(m^{\downarrow})$, and not like in the aforementioned analyses by $del_J(m^{\downarrow})$. In this way, the worst-case delay of a message m does not depend on the worst-case delay of its interfering message $m^{\downarrow}$ (i.e. $del(m) \not/ f(del(m^{\downarrow}))$), so any cascading effect is prevented and the analysis complexity is reduced to linear. The implications are that in the example illustrated in Fig. 6 any parameter manipulations performed on m_s would trigger only a feasibility recheck of m_r and will not influence m_{p1} and m_{p2} at all. The only change is in the computation of the interference component; instead of Eq. 5, Eq. 7 is used.

$$del_N(\hat{m}, t) = \sum_{\forall m' \in \mathcal{M}_{DI}(\hat{m})} \left\lceil \frac{t + W_{a_{m'}} - del_I(m')}{T_{a_{m'}}} \right\rceil \cdot \left(del_I(m') + del_B(m')\right) \quad (7)$$

5.3.2 Numerical example revisited

Let us return to the numerical example and see how this change influences the computed delays. Messages m_s and m_q do not suffer the interference, so their worst-case delays are $del(m_s) = del(m_q) = del_I(m_s) = del_I(m_q) = 2$.

The message m_r can suffer the interference from m_s and its delay is:

$$del(m_r) = del_I(m_r) + \left\lceil \frac{del(m_r) + W_{a_{m_s}} - del_I(m_s)}{T_{a_{m_s}}} \right\rceil \cdot del_I(m_s)$$

$$del(m_r)^0 = 2 + \left\lceil \frac{0 + 10 - 2}{10} \right\rceil \cdot 2 = 4$$

$$del(m_r)^1 = 2 + \left\lceil \frac{4 + 10 - 2}{10} \right\rceil \cdot 2 = 2 + 4 = 6$$

$$del(m_r)^2 = 2 + \left\lceil \frac{6 + 10 - 2}{10} \right\rceil \cdot 2 = 2 + 4 = 6$$

Notice, that the delay of m_r is now 6, while it was 4 with the previous approach. Now we compute the delay of the composite message m°_p.

$$del(\hat{m^{\circ}_p}) = del_I(\hat{m^{\circ}_p}) + \sum_{\forall m' \in [m_q, m_r]} \left\lceil \frac{del(\hat{m^{\circ}_p}) + W_{a_{m'}} - del_I(m')}{T_{a_{m'}}} \right\rceil \cdot del_I(m')$$

$$del(\hat{m^{\circ}_p})^0 = 4 + \left\lceil \frac{0 + 9 - 2}{9} \right\rceil \cdot 2 + \left\lceil \frac{0 + 6 - 2}{6} \right\rceil \cdot 2 = 4 + 2 + 2 = 8$$

$$del(\hat{m^{\circ}_p})^1 = 4 + \left\lceil \frac{8 + 9 - 2}{9} \right\rceil \cdot 2 + \left\lceil \frac{8 + 6 - 2}{6} \right\rceil \cdot 2 = 4 + 4 + 4 = 12$$

$$del(\bar{m}_p)^2 = 4 + \left\lceil \frac{12+9-2}{9} \right\rceil \cdot 2 + \left\lceil \frac{12+6-2}{6} \right\rceil \cdot 2 = 4+6+6 = 16$$
$$del(\bar{m}_p)^3 = 4 + \left\lceil \frac{16+9-2}{9} \right\rceil \cdot 2 + \left\lceil \frac{16+6-2}{6} \right\rceil \cdot 2 = 4+6+8 = 18$$
$$del(\bar{m}_p)^4 = 4 + \left\lceil \frac{18+9-2}{9} \right\rceil \cdot 2 + \left\lceil \frac{18+6-2}{6} \right\rceil \cdot 2 = 4+6+8 = 18$$

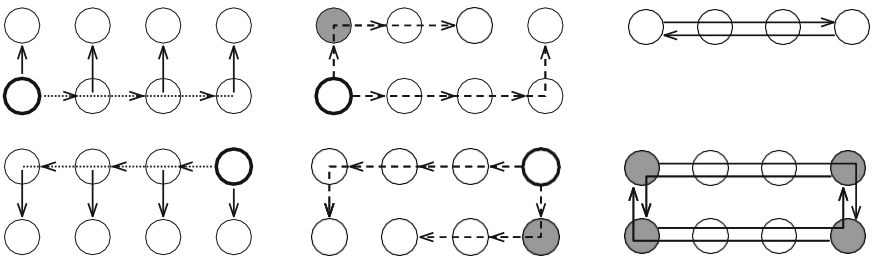
The delay of m°_p (and consequently m_{p1} and m_{p2}) is now 18, while it was 14 with the previous method.

5.4 Determinism and master-independence of message-paths

5.4.1 LMM challenges

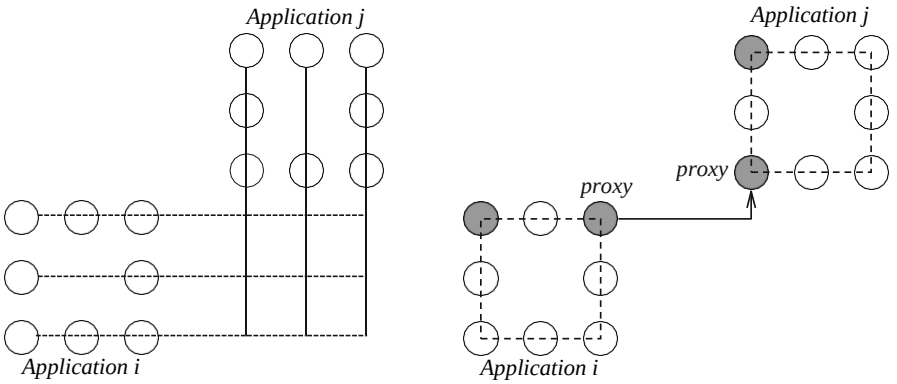
When considering *LM M* , the non-determinism of message paths is related to both intra- and inter-application messages. We explain this with two illustrative examples. In Fig. 7a the top and the bottom part present generated intra-application messages of the same application, captured at two different time instances (i.e. represented by emphasized circles different current masters communicate with all the slaves). As is obvious, depending on the master selection, two entirely different message-sets can be generated. Similar problem occurs when analysing the inter-application traffic. In Fig. 8a is given an example where two applications communicate. As stated in Sect. 4, the inter-application communication is performed by an exchange of messages between the current masters. Due to the master volatility property, a message between these applications can take any of the routes given in Fig. 8a.

One possible solution is to analyse all scenarios arising from the fact that any dispatcher of an application under analysis can be a master. However, this also requires to investigate all possible subscenarios, as every other application can have any of its dispatchers as a current master. Fixing a master dispatcher for each application, exhaustively enumerating all possible combinations and checking the feasibility with the analysis presented in Sect. 5.3 can be prohibitively expensive, as the complexity of the approach is $O(|D|^{|A|})$, where $|D|$ represents the number of dispatchers per



(a) Unconstrained messages **(b)** Re-routed messages **(c)** Supermessages

Fig. 7 Intra-application communication



(a) Without proxy dispatchers **(b)** With proxy dispatchers

Fig. 8 Intra-application communication

application and $|A|$ stands for the number of applications in the system. This can cause the analysis intractability even for few applications and dispatchers.

Another possibility is to consider a concurrent existence of all possible messages of every application. However, many messages are mutually exclusive, as each one of them can exist only under specific conditions (one specific dispatcher being the master). Thus, considering a concurrent existence of all possible messages can be overly pessimistic.

The problems of intractability and pessimism are the consequence of non-deterministic message paths. In order to solve this problem, we use the concepts introduced in the work of Nikolic´ et al. (2014), namely the *mapping* and *rerouting constraints*, the *supermessages* and the *proxies*.

5.4.2 Constraints, supermessages and proxy dispatchers

Constraint 1 (Nikolic´ et al. 2014) *Dispatchers of an application can be positioned only on the edges of a rectangular $a \times b$ structure, such that no corner is left unoccupied and $a, b \in \mathbb{N}$. The special case is a line-like shape, where one dimension of the application shape is equal to* 1*, that is $a = 1$ or $b = 1$.*

In Fig. 7c are depicted two applications, where dispatcher positions fulfil Constraint 1. In the lower part of Fig. 7c is the application with the rectangular shape of dimensions 4×2, while in the upper part of Fig. 7c is the application with a line-like shape of dimensions 4×1.

Constraint 2 (Nikolic´ et al. 2014) *Intra-application messages travel only on the edges of the shape its application is forming, and re-routing occurs where needed to comply with the global XY routing policy. An individual message rotation (i.e. clockwise or counterclockwise) is chosen such that the traversal distance is minimised.*

The top and the bottom part of Fig. 7b illustrate how the messages should be routed. The shaded dispatchers depict locations where reroutings occur. A rerouting is a fast

on-core routine, performed in an interrupt-like manner, which can be implemented by instrumenting the Hardwall^{TM} technology of Tilera platforms (Tilera 2014). Note that the rerouting mechanism is employed to assure that the traffic transfer complies with the global XY routing policy, which is the most common routing mechanism in the present many-core platforms. If the platform allows routing of messages across arbitrary paths, the reroutings are not needed.

Notice, that with these constraints the part of the NoC infrastructure that intra-application traffic uses is made deterministic. In order to make it also master-independent, a construct called the *supermessage* is used.

Definition 5 (*Supermessage*) (Nikolic´ et al. 2014) A supermessage is a message which connects (i) diagonally-placed dispatchers if an application has a rectangular shape, or (ii) terminal dispatchers if an application has a line-like shape.

An application with a rectangular shape has 4 supermessages, while an application with a line-like shape has only 2, and does not require reroutings (see Fig. 7c). Supermessages have three interesting properties, (i) they are master-independent, (ii) their number is substantially smaller than the number of possible intra-application messages (iii) the intra-application traffic of an application can be expressed as a linear combination of its supermessages (Nikolic´ et al. 2014).

In order to make the inter-application traffic also deterministic and master-independent, we introduce *proxy dispatchers*:

Definition 6 (*Proxy dispatcher*) (Nikolic´ et al. 2014) A proxy dispatcher is a dispatcher which is selected at design time, and which participates in the inter-application communication. It mediates the communication between its master and the proxy dispatcher of the interacting application.

An illustrative example of Definition 6 is given in Fig. 8b. In this scenario, an inter-application message is divided into 5 different components: (i) a message from the master sender to its proxy, (ii) a rerouting on the core of the proxy sender, (iii) a message between the proxies, (iv) a rerouting on the core of the proxy receiver, (v) a message from the proxy receiver to its master. Proxies are decided at design-time, thus a message between proxy dispatchers is deterministic and master-independent. Additionally, the communication between masters and their proxies, on both the sender and receiver side, complies with the rules of the intra-application traffic, thus can be expressed as a linear combination of supermessages.

To summarise, Constraints 1–2 and Definitions 5–6 allow expressing the intra- and inter-application traffic as a linear combination of supermessages and inter-proxy messages, which are deterministic and master-independent. In this paper we perform the analysis on such a model.

5.4.3 Performing the analysis

The analysis is very similar to that of Shi and Burns (2009), introduced in Sect. 5.2 and modified in Sect. 5.3. However, there are several differences: (i) deadlines are posed per-application and not per-individual message, (ii) the analysis involves reroutings

and, (iii) instead of once, each supermessage and inter-proxy message m can appear multiple times during one inter-arrival period of its application. Let m be a supermessage or an inter-proxy message, and let $O(m)$ be the maximum number of its occurrences, when the traffic of its application is expressed as a combination of supermessages and inter-proxy messages. Moreover, let supermessages inherit the priority of the corresponding application, while inter-proxy messages inherit the priority of the sender application. The motivation behind such a priority assignment is to make all messages contributing to the worst-case delay of one application (all its supermessages and sent inter-proxy messages) share the same priority. Subsequently, let all same-priority messages be grouped within a common *composite message*, a concept introduced in Sect. 5.2.

Assuming that the same-priority supermessages and inter-proxy messages are grouped within the composite message $\mathring{m}$, its delay (Eq. 8) is equal to the sum of (i) the isolation latency, (ii) the lower-priority blocking, (iii) the rerouting, of all messages constituting it, and (iv) the interference. $del_R(\mathring{m})$ represents the maximum rerouting delay of $\mathring{m}$. The work of Nikolić et al. (2014) describes the steps how to obtain $O(m)$ and $del_R(\mathring{m})$. Recall that every message in this analysis is either a supermessage or an inter-proxy message.

$$del(\mathring{m}) = \overbrace{\sum_{\forall m \in \mathcal{M}_C(\mathring{m})} del_I(m) \cdot O(m)}^{\text{isolation}} + \overbrace{\sum_{\forall m \in \mathcal{M}_C(\mathring{m})} del_B(m) \cdot O(m)}^{\text{lower-priority blocking}} + \overbrace{del_R(\mathring{m})}^{\text{rerouting}} + \overbrace{\sum_{\forall m' \in \mathcal{M}_{DI}(\mathring{m})} \left\lceil \frac{del(\mathring{m}) + W_{a_{m'}} - del_I(m')}{T_{a_{m'}}} \right\rceil \cdot (del_I(m') + del_B(m')) \cdot O(m')}^{\text{interference}} \tag{8}$$

Note that by allowing each message to appear multiple times during the inter-arrival period of its application, multiple direct same-priority blockings between any two messages are possible. Indeed, our approach allows such a possibility, and does not require the restriction that the same-priority messages should be treated in a FIFO order. In fact, in our approach the ordering of the same-priority messages is entirely irrelevant.

So far, we have explained how to compute the worst-case delay of a composite message. That delay presents not only the upper-bound on the worst-case delay of individual messages constituting it, but also on their joint delay. That is, the delay of a composite message from the aforementioned example $del(\mathring{m}_p) = 18$ presents an upper bound on the delay of both m_{p1} and m_{p2}, but also on their joint traversal. Thus, the worst-case delay of a composite message represents the joint worst-case delay of all messages having the same priority as the composite message. Therefore, if $\mathring{m}_a$ is the composite message of the application a, then the worst-case delay of a is equal to the worst-case delay of $\mathring{m}_a$, i.e. $del(a) = del(\mathring{m}_a)$. This infers that the application is feasible if the worst-case delay of its composite message is less than or equal to its temporal constraint: $del(\mathring{m}_a) \leq W_a$.

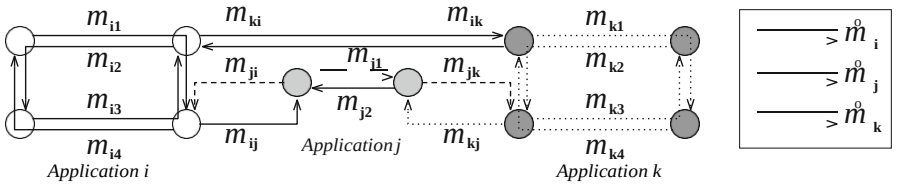


Fig. 9 Example of supermessages and inter-proxy messages

An illustrative example is given in Fig. 9, where three applications communicate with each other. Messages $m_{i\,1}$–$m_{i\,4}$ are the supermessages of a_i, messages $m_{j\,1}$ and $m_{j\,2}$ of a_j, and messages m_{k1}– m_{k4} of a_k. Moreover, messages m_{ij}, m_{ji}, m_{ik}, m_{ki}, m_{jk}, m_{kj} are the inter-proxy messages.

In order to test the feasibility, a composite message has to be constructed for the application under analysis, which is demonstrated with the following example. The composite message of a_i, denoted by $\mathring{m}_i$, includes the messages with the same priority as a_i: its supermessages and its sent inter-proxy messages (solid lines in Fig. 9), i.e. $M_C(\mathring{m}_i) = \{m_{i1}, m_{i2}, m_{i3}, m_{i4}, m_{ij}, m_{ik}\}$. The application a_j has the composite message $\mathring{m}_j$ which contains its supermessages and its sent inter-proxy messages (dashed lines in Fig. 9), i.e. $M_C(\mathring{m}_j) = \{m_{j1}, m_{j2}, m_{ji}, m_{jk}\}$. Finally, the application a_k has the composite message $\mathring{m}_k$, which includes its supermessages and its sent inter-proxy messages (dotted lines in Fig. 9), $M_C(\mathring{m}_k) = \{m_{k1}, m_{k2}, m_{k3}, m_{k4}, m_{ki}, m_{kj}\}$. The application-set is feasible if $del(\mathring{m}_i) \leq W_i \wedge del(\mathring{m}_j) \leq W_j \wedge del(\mathring{m}_k) \leq W_k$.

Notice, that supermessages and reroutings inherently bring additional pessimism. That is, each message may traverse only a fraction of the path of the supermessages which are used to express it. Additionally, reroutings induce additional delay and potentially "sacrifice" the performance, in order to achieve the predictability of message-paths. However, these concepts solve the issue of non-deterministic message paths, and dramatically reduce the number of messages (i.e. only supermessages and inter-proxy messages of each application are considered), which makes the analysis tractable.

6 Mapping quality

The multi-objective nature of the proposed approach is reflected by the fact that the goal is not just to provide a mapping which is feasible, but to provide a feasible mapping which maximises the abilities of the application-set to perform runtime load balancing via application migrations. There are two aspects which we consider relevant for the migrative potential of an application, namely *the dimensions* of the rectangular $a \times b$ shape its dispatchers are forming and *the distribution* of the dispatchers on that shape. The approach we apply when evaluating the mapping of the application can be summarised with the following two observations:

Observation 1 The greater the shape of the application is, the greater its migrative abilities are.

Observation 2 Assuming a fixed shape, the more even the distances between the application dispatchers are, the greater its migrative abilities are.

The reasoning is that, given that the migration has to occur, it is better to accommodate the execution on some far core, rather than on some near core, because far migrations allow efficient global load balancing, energy/thermal management and make the system more resilient to clustered failures (i.e. a part of the chip starts malfunctioning). Conversely, near migrations only partially solve these issues, or don't solve them at all. Thus, the intention behind our approach is to prevent near migrations by (i) maximising the shape of the application (Observation 1), and distributing its dispatchers on the shape, such that the inter-dispatcher distances are as even as possible (Observation 2).

Although it may seem that these objectives superficially contradict the feasibility requirement, this is not entirely true. If perceived as an optimisation problem, migrative abilities of applications are the objective function which has to be maximised, and the feasibility is a constraint which must be fulfilled. Subsequently, the solution should be found such that (i) all the applications are feasible, (ii) the dimensions of the application shapes are as big as possible, (iii) the distances between the dispatchers of the same application are as equal as possible.

In order to qualitatively evaluate different mappings, a proper metric has to be established such that it implements the aforementioned reasoning. Let (d_j, d_k) denote a pair of neighbouring dispatchers of one application, and let $nhops(d_j, d_k)$ be the distance between them, expressed in hops. Moreover, let $\mathcal{D}^P_{a_i}$ denote all such pairs of neighbouring dispatchers of the application a_i. A quality of a mapping of the application a_i, denoted by q_i (Eq. 9), is equal to the product of the distances between its neighbouring dispatchers, multiplied by its migration coefficient I_i.

$$q_i = I_i \cdot \prod_{\forall (d_j, d_k) \in \mathcal{D}^P_{a_i}} nhops(d_j, d_k) \tag{9}$$

The migration coefficient I_i was introduced in Sect. 4.2 and it symbolises the importance of application's spatially distributed mapping. In other words, the more the system would benefit from the application's spatially distributed mapping, the greater this parameter is. For example, a computationally demanding application may have a significant impact on the thermal properties of the core where it is executing (and its surrounding), therefore, having the possibility to perform far migrations of that application is desirable from the thermal perspective. Consequently, for every such application a_i its coefficient I_i should be set high. Similarly, allowing far migrations for a critical application may improve its resilience towards core/cluster failures. Thus, each such application a_i should have its I_i coefficient set high. In this work we do not elaborate on the values of migration coefficients of individual applications. In fact, we just assume that the values have already been specified, and that the same are used as a means to classify the applications according to the importance of their spatially distributed mappings.

There are three reasons why a product of inter-dispatcher distances is used as the evaluation metric. Primarily, because it is a computationally cheap operation. That is,

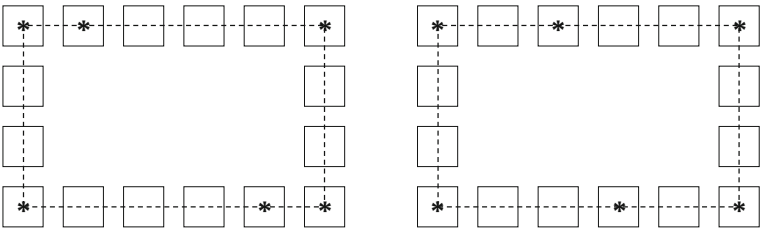


Fig. 10 Comparison of mappings

due to the rectangular or line-like structure of the application shape (Constraint 1), the inter-dispatcher distances can be obtained in a single traversal across the circumference of the application's shape in either clockwise or couter-clockwise direction. Since during the mapping process the evaluation of many different shapes of all applications will be performed, it is of paramount importance to limit its complexity. Secondly, the product of inter-dispatcher distances is monotonically increasing with the shape size (see Observation 1). Finally, when assuming that the shape has been decided, the product of inter-dispatcher distances reaches the maximum when all the distances are as even as possible (see Observation 2 and for the formal proof see Theorem 2 in Appendix).

Note that a zero-distance between dispatchers presents a special case where the same are located on a common core. In the assumed model, such a mapping is meaningless and the metric expressed with Eq. 9 also penalises such mappings, hence returning $q_i = 0$.

Assuming that the circumference of an application shape C and the number of dispatchers $|D|$ are given, choosing and applying an optimal dispatcher placement would be trivial: if possible, make all distances equal $\frac{C}{|D|}$, otherwise make distances either $\lfloor \frac{C}{|D|} \rfloor$ or $\lceil \frac{C}{|D|} \rceil$. However, the corners of the application shape pose implicit constraints regarding the inter-dispatcher distances and, in some cases, prevent optimal solutions. Also, not all the cores located on the edges of the application shape might be available due to schedulability reasons (Sect. 7.4), thus further preventing optimal solutions. Therefore, the purpose of the aforementioned evaluation metric is to provide a qualitative comparison between different possible suboptimal dispatcher placements, in cases when optimal ones are not possible.

The example in Fig. 10 illustrates how different mappings are evaluated. Two possibilities are presented, in both of them the application claims the same shape, but the placement of the inner dispatchers differs. All dispatchers are represented with a star sign. For clarity purposes, assume that the migration coefficient is equal to 1. By obtaining the values with Eq. 9, we find that the left solution has $q_i = 144$, while the right one has $q_i = 324$, which favours more the latter. This coincides with our reasoning and intentions. Also note that for a given example $C = 16$, $|D| = 6$, and $\frac{C}{|D|} = \frac{16}{6} = 2.67$. Therefore, the right mapping presents one of optimal solutions, due to the fact that its all dispatcher distances are either 2 or 3.

Upon defining the individual, per-application quality metric, now we define the quality of an entire solution (Eq. 10). It is equal to the sum of individual qualities of all applications comprising an application-set A.

$$Q = \sum_{\forall a_i \in \mathcal{A}} q_i \tag{10}$$

7 Mapping procedure

After defining the feasibility analysis (Sect. 5) and the qualitative metric for secondary objectives (Sect. 6), in this section we present the application mapping process. The proposed method consists of three stages: *Initial Phase* (IP), *Feasibility Phase* (FP) and *Optimisation Phase* (OP).

Before the mapping process begins, based on their priorities, all applications are divided into two groups: a set of high-priority applications H and a set of low-priority applications L (Eq. 11).

$$\begin{cases} P_a > P^{min} + P \cdot (P^{max} - P^{min}) \Rightarrow a \in \mathcal{H} \\ P_a \leq P^{min} + P \cdot (P^{max} - P^{min}) \Rightarrow a \in \mathcal{L} \end{cases} \tag{11}$$

P^{max} and P^{min} denote the maximum and the minimum system priorities, respectively, while the parameter P ($0 \leq P < 1$) is arbitrarily chosen by the system designer. All applications belonging to the group H will be mapped during IP and will not be subject to any changes during FP and OP. The rationale for this decision is that any change in the mapping of an application $a \in H$ (e.g. its shape, the position of its dispatchers) requires a new feasibility check of both a and all lower priority applications which composite messages are directly interfered by the composite message of a. Thus, the recalculation triggered by a high-priority application might be computationally expensive, as there might be a substantial number of directly interfered applications. Therefore, the parameter P is introduced as a means to control the complexity of the entire mapping process.

7.1 Initial phase (IP)

As already described, during this phase only the high-priority applications are mapped. The applications are sorted non-increasingly with respect to their priorities, and mapped sequentially in that order. By performing the mapping in this manner, in most cases, the mapping of one application will not have an impact on the feasibility of previously mapped (higher-priority) applications. Exceptions are the cases when there exists an inter-application message from some already mapped higher priority application $a^{\uparrow}$, to the currently mapping application a. In such cases, the mapping of a also maps the inter-proxy message between $a^{\uparrow}$ and a which has the priority of $a^{\uparrow}$ and belongs to $m^{\circ}_{a^{\uparrow}}$. This invokes an update of $m^{\circ}_{a^{\uparrow}}$, a feasibility recheck of $a^{\uparrow}$ and all interfered applications with intermediate priorities. Depending on the frequency of these situations, the complexity of IP ranges between $O(|H|)$ (no feasibility rechecks necessary) and $O(|H|^2)$ (the mapping of every application invokes feasibility rechecks), where $|H|$ stands for the amount of applications in H. As will be seen in Sect. 8, the actual complexity is closer to the former estimate (a linear complexity).

When mapping an application, we differentiate between several types of mappings. A *narrow mapping* presents a mapping where an application shape covers the minimum possible surface, i.e. a mapping where all dispatchers of an application occupy consecutive cores. For instance, the possible narrow mappings for a 4-dispatcher application are: 1×4, 4×1, 2×2. Conversely, a *wide mapping* is a mapping where an application shape covers the maximum possible surface, in most cases the boundaries of the grid. We refer to the surfaces of the narrow and the wide mapping as to S^{min} and S^{max}, respectively.

Since the applications from H are mapped during IP, and are not subject to any changes in the subsequent mapping phases, it is essential to dedicate a proper shape to each of these applications. Assuming wide mappings for most applications during IP will create significant high priority traffic within the network, which might cause the applications from L to be unable to reach the feasibility. Conversely, restricting the applications from H to claim the narrow mappings might unnecessarily preserve the network resources underutilised, as there might be very few applications in L. In order to manage this design trade-off we introduce the parameter G. It controls the mapping "greediness" of high-priority applications. G presents an upper-bound on the allowed application shapes. For instance, if $G = 0$ only the shapes which surface S is less than or equal to S^{min} are allowed. Similarly, if $G = 1$, shapes which fulfil $S \leq S^{max}$ are allowed, which includes all rectangular/linear shapes. If $0 < G < 1$, allowed shapes are calculated by Eq. 12. That is, if the mapping of a 4-dispatcher application has to be performed on a 8×8 platform with $G = 0.5$, then $S^{min} = 4$, $S^{max} = 64$ and $S \leq 34$. This excludes shapes $\{8 \times 8, 8 \times 7, 7 \times 8, 7 \times 7, 8 \times 6, 6 \times 8, 7 \times 6, 6 \times 7, 6 \times 6\}$ from the consideration.

$$S \leq S^{min} + G \cdot (S^{max} - S^{min}) \tag{12}$$

Algorithm 1 illustrates how the IP stage is performed. First, the high-priority applications are selected and sorted by priorities, non-increasingly (lines 1–4). The applications are treated sequentially; the values of S^{min} and S^{max} are obtained (line 6), and a shape surface threshold S is calculated (line 7). Then, only the shapes which surface is less than or equal to the calculated threshold S are selected and sorted by the shape surface, non-increasingly (lines 8–11). The mapping is attempted on the entire grid with the selected application and the biggest allowed shape (lines 14–15). Note that this process involves (i) the placement of the shape at the particular location on the grid, (ii) the generation of the supermessages, (iii) the assignment of the individual proxy roles for both the application under analysis and the other applications communicating with it, (iv) the generation of the inter-application messages assuming the elected proxies, (v) the generation of the composite message (vi) the feasibility check for the composite message. Each inter-application message sent to the currently mapping application has a higher priority, hence each such message has to be added to the respective composite message (lines 17–20). Consequently, the feasibility of the applications containing those composite messages has to be rechecked. This may also trigger the rechecks of other applications influenced by these updates, therefore a feasibility recheck is performed for each such application (lines 21–27).

Algorithm 1: $IP(A, P, G)$ The first mapping phase IP (Initial Phase)

```
   input : A, P, G
 1 foreach (a ∈ A : P_a > P^min + P · (P^max − P^min)) do
 2 |   add(a, H); // Select all high-priority applications
 3 end
 4 H.sort(P↓); // Sort by priority, non-increasingly
 5 foreach (a ∈ H) do
 6 |   S^min = a.min_area(); S^max = a.max_area(); // Compute S^min and S^max
 7 |   S = S^min + G · (S^max − S^min); // Find a shape surface threshold
 8 |   foreach (shape : shape.S() ≤ S) do
 9 |   |   add(shape, Allowed); // Select allowed shapes
10 |   end
11 |   Allowed.sort(S↓); // Sort by shape surface, non-increasingly
12 |   feasible = false;
13 |   while (feasible ≠ true ∧ Allowed ≠ empty) do
14 |   |   shape = Allowed.remove(); // Get the biggest existing shape
15 |   |   feasible = a.test(shape);
16 |   |   if (feasible == true) then
17 |   |   |   foreach (a' ∈ a.Senders()) // Update of composite messages
18 |   |   |   do
19 |   |   |   |   a'.update();
20 |   |   |   end
21 |   |   |   if (a.Senders() ≠ ∅) then
22 |   |   |   |   foreach (a' ∈ H : a'.recheck() == true) // Feasibility recheck
23 |   |   |   |   do
24 |   |   |   |   |   feasible = feasible ∧ a'.test(a'.shape);
25 |   |   |   |   |   if (feasible ≠ true) then break;
26 |   |   |   |   end
27 |   |   |   end
28 |   |   end
29 |   |   if (feasible == true) then
30 |   |   |   location = a.find_best_location(shape); // Find the best location
31 |   |   |   a.map(shape, location);
32 |   |   |   a.distribute_dispatchers(); // Maximise the mapping quality
33 |   |   end
34 |   end
35 |   if (feasible ≠ true) then Mapping.Failed(a); // Declare failure of IP
36 |
37 end
38 Mapping.Success(); // Declare success of IP
```

If the application can be mapped with the selected shape on multiple places of the grid, the location is found such that the worst-case delay of the composite message is minimised. In this way the algorithm searches for the position on the grid where the application suffers the least interference from the other traffic. Notice, that this strategy forces interacting applications to be mapped close to each other. A special case occurs when proxies of two interacting applications share the same core and the inter-proxy message does not exist. Furthermore, if there are several locations on the grid where the application can be mapped and for which the delay of the composite message is minimised, the approach selects the one for which the sum of the dispatcher distances from the center of the grid is minimised. The intention behind mapping the

application as close to the center of the grid as possible is as follows: any lower-priority application which is yet to be mapped, and which performs the communication with the said application, will have the possibility to evaluate more mapping options so as to minimise the communication penalty, while that would not be the case if the said application was mapped on the border of the grid. Note that these two location selection criteria are not explicitly mentioned in Algorithm 1, but we assume that this logic is encapsulated within the method in the line 30.

Once the best location is found, the application is considered mapped (line 31). Subsequently, the rest of dispatchers (if any) are positioned on the edges of the shape, such that the mapping quality q_i is maximised (line 32).

If the application cannot be mapped on the grid with the current shape, the mapping is attempted with the next shape from the collection of allowed shapes. The process repeats until the proper shape and location are found, such that the feasibility constraints are satisfied for the currently mapping and the previously mapped applications. If the feasibility cannot be reached with any of the shapes, the mapping process declares a failure (line 35). Conversely, when all the applications from H are mapped, IP declares a success (line 38).

7.2 Feasibility phase (FP)

This phase performs the mapping of low-priority applications, with the primary objective to derive a feasible solution where the entire application-set is mapped. Therefore, during FP, every application is mapped with the narrow mapping. FP is described by Algorithm 2. First, all low-priority applications are grouped in L and sorted by priority, non-increasingly (lines 1–4). The applications are treated sequentially; a surface of a narrow mapping S^{min} is found (line 6), and subsequently it is used to find all possible shapes which correspond to the narrow mapping of the application (lines 7–9). The mapping is attempted with the first shape from the list, without any preference, as all selected shapes represent narrow mappings (lines 12–13). The same logic used in IP applies here: if needed, the composite messages of higher-priority applications are updated (lines 15–18) and subsequently the feasibility of relevant applications is rechecked (lines 20–24). Similarly, if multiple grid locations are available for the same shape, the one with the minimum delay of the composite message is selected, while if more than one location report the same delay, the one closer to the center of the grid has the precedence (line 28). Once the best location is found, the application is mapped (line 29).

If the attempted shape violates the feasibility of the currently mapping application, or any other already mapped application, the mapping is attempted with the next one from the list of allowed shapes. The process is repeated until a shape is found such that all applications are feasible. If none of the shapes satisfies this requirement, the mapping process declares a failure (line 32). Conversely, once all the applications are mapped, FP declares a success (line 35). The computational complexity of FP also varies. If there are no feasibility rechecks necessary, the complexity is equal to $O(|L|)$, where $|L|$ denotes the number of low-priority applications. Conversely, if the mapping of every application requires feasibility rechecks, the complexity is $O(|L| \cdot |A|)$, where $|A|$ denotes the total number of applications in the application-set, i.e. $|A| = |H| + |L|$.

Algorithm 2: $FP(A, \mathrm{P})$ The second mapping phase FP (Feasibility Phase)

```
   input : A, P
 1 foreach (a ∈ A : P_a ≤ P^min + P · (P^max − P^min)) do
 2 |   add(a, L); // Select all low-priority applications
 3 end
 4 L.sort(P↓); // Sort by priority, non-increasingly
 5 foreach (a ∈ L) do
 6 |   S^min = a.min_area(); // Compute S^min
 7 |   foreach (shape : shape.S() == S^min) do
 8 |   |   add(shape, Allowed); // Select narrow-mapping shapes
 9 |   end
10 |   feasible = false;
11 |   while (feasible ≠ true ∧ Allowed ≠ empty) do
12 |   |   shape = Allowed.remove(); // Get the first allowed shape
13 |   |   feasible = a.test(shape);
14 |   |   if (feasible == true) then
15 |   |   |   foreach (a' ∈ a.Senders()) // Update of composite messages
16 |   |   |   do
17 |   |   |   |   a'.update();
18 |   |   |   end
19 |   |   |   if (a.Senders() ≠ ∅) then
20 |   |   |   |   foreach (a' ∈ A : a'.recheck() == true) // Feasibility rechecks
21 |   |   |   |   do
22 |   |   |   |   |   feasible = feasible ∧ a'.test(a'.shape);
23 |   |   |   |   |   if (feasible ≠ true) then break;
24 |   |   |   |   end
25 |   |   |   end
26 |   |   end
27 |   |   if (feasible == true) then
28 |   |   |   location = a.find_best_location(shape); // Find the best location
29 |   |   |   a.map(shape, location);
30 |   |   end
31 |   end
32 |   if (feasible ≠ true) then Mapping.Failed(a); // Declare failure of FP
33 |
34 end
35 Mapping.Success(); // Declare success of FP
```

As will be seen in Sect. 8, the actual complexity is closer to the former estimate (linear). Note that the output of FP is the first feasible solution of an entire application-set.

7.3 Optimisation phase (OP)

The objective of this phase is to improve on the solution received from FP. This is performed by attempting to extend the shapes of narrow mappings that low-priority applications claimed during FP. The process ends when no further extensions are possible. That also marks the end of the entire mapping process and the reached solution presents the final output. OP is depicted by Algorithm 3.

Similarly, applications from L are selected and sorted non-increasingly (lines 1–4), but during OP the parameter I. As mentioned in Sect. 6, it represents the significance of the spatially distributed mapping of an application, i.e. the more the system

Algorithm 3: *OP*(*A*, P) The third mapping phase *OP* (Optimisation Phase)

```
   input : A, P
 1 foreach (a ∈ A : P_a ≤ P^min + P · (P^max − P^min)) do
 2 |   add(a, L); // Select all low-priority applications
 3 end
 4 L.sort(I); // Sort by migration coefficient, non-increasingly
 5 foreach (a ∈ L) do
 6 |   feasible = true;
 7 |   while (feasible == true) do
 8 |   |   new_shape = a.expand(a.shape); // Find expanded shape to attempt
 9 |   |   feasible = a.test(new_shape);
10 |   |   if (feasible == true) then
11 |   |   |   foreach (a' ∈ A : a'.recheck() == true) // Feasibility re-checks
12 |   |   |   do
13 |   |   |   |   feasible = feasible ∧ a'.test(a'.shape);
14 |   |   |   |   if (feasible / true) then break;
15 |   |   |   end
16 |   |   |   if (feasible == true) then
17 |   |   |   |   a.shape = new_shape; // Claim expanded shape
18 |   |   |   |   a.rearrange_dispatchers(); // Maximise the mapping quality
19 |   |   |   end
20 |   |   end
21 |   end
22 end
23 Mapping.Success(); // Declare success of OP and entire mapping process
```

would benefit from the spatially distributed mapping of an application, the greater its parameter I is. The positive side is that applications for which distribution matters more are given the possibility to claim resources before others, thus increasing the chances of the mapping process to derive a good quality solution. The downside is that the number of necessary feasibility rechecks significantly increases; an expanding application can invoke not only feasibility rechecks mentioned in the previous mapping stages, but also of all its directly interfered lower-priority applications, which was not possible in the previous stages as applications were sorted by their priorities, non-increasingly. Thus, unlike the previous mapping stages, where feasibility rechecks were an exception, during *OP* the same will be performed regularly. The computational complexity of *OP* is identical to that of *FP*, however, as will be seen in Sect. 8, the actual complexity is closer to the higher (sub-quadratic) estimate.

The applications are treated sequentially; the expanded shape is found and the mapping attempted (lines 8–9). The process consists of an attempt to stretch the application shape: (i) the supermessages are re-generated, (ii) the proxy roles are re-assigned for both the application under analysis and the other applications interacting with it, (iii) the inter-application messages are re-generated assuming the newly elected proxies, (iv) the composite messages containing the modified inter-application messages are re-generated. As described above, this may require a significant amount of feasibility rechecks (lines 11–15). The application is expanded until any further stretches will cause infeasibility of either itself or any other application. Every expansion is followed by a rearrangement of dispatchers of an expanding application, so as to equalise the

inter-dispatcher distances as much as possible and improve the mapping quality (line 18). Once this process is performed for all applications from $\mathcal{L}$, the entire mapping process concludes (line 23) and returns the current solution as the final output.

7.4 Schedulability

As already mentioned (Sect. 4.2), the greatest distinction of *LM M* from the existing approaches is that release/migration decisions are explicitly detached from scheduling decisions. In other words, via its agreement protocol, each application *itself* decides on which of its candidate cores will the next job be released, while scheduling the job is the responsibility of the independent single-core scheduler that is located on the selected core.

The schedulability analysis of *LM M* is still an unexplored topic. Given that this aspect is not the objective of this work, and for the sake of completeness, here we address the schedulability by making the following simplifying assumptions. An application is *schedulable*, if at least one of its dispatchers is schedulable. A dispatcher is schedulable, if it can release a job on its core, on behalf of its application, which will not miss a deadline. A job does not miss a deadline, if its worst-case response time R_a (Burns and Wellings 2009) is less than or equal to its deadline D_a, when treating that core as an independent single-core system with the fixed-priority scheduling (jobs inherit dispatcher priorities, while dispatchers inherit application priorities). Specifically, a job of an application a has the deadline equal to the period of a, reduced by its communication deadline, i.e. $D_a = T_a - W_a$.

Let R_a^{min} be the minimum of the response times of the application a, on all the cores where it has dispatchers. Then, a is schedulable if $R_a^{min} \leq D_a = T_a - W_a$. Notice, that this simplistic approach assures that each application will always be able to execute on at least one of the cores of its dispatchers, while the possibility to migrate will depend on the outcome of online schedulability tests, performed by its dispatchers on their respective cores during agreement protocols at runtime.

At this stage several questions can be asked: (i) How to perform the schedulability analysis when applications have more sophisticated schedulability requirements, e.g. require multiple schedulable dispatchers (useful in scenarios with voluntary core shutdowns and core failures)? (ii) What is the most efficient way to assign priorities to dispatchers and jobs they release? (iii) Which single-core scheduling policy to implement within kernels? These important questions and open problems are guidelines for our future work.

8 Evaluations

In this section we perform the evaluation of the proposed approach. Specifically, through different case studies, we investigate the overall efficiency and applicability of the proposed application mapping method, as well as how different choices and trade-offs, achievable through parameter manipulations, influence the quality of derived solutions.

8.1 Case study 1: application shapes and overheads of constrained routes

In many situations, several shapes have similar or identical characteristics, especially during FVP when only narrow mappings are considered. In order to asses the differences between shape types and reason about their applicability, we analyse their characteristics which we consider relevant, namely a traversal distance of the average and the longest intra-application message. For easier calculation, in all cases narrow mappings are assumed, while for rectangular shapes only an even number of dispatchers is considered.

Equation 13 calculates the average traversal distance of an intra-application message, assuming a n-dispatcher application with a line-like shape. If the i-th dispatcher of a horizontally stretched application is the master, it communicates with $i-1$ dispatchers on the left and the rest $n-i$ dispatchers on its right (the terms in the brackets of Eq. 13). The outer summation is performed so as to account for all possible masters. In order to obtain the distance of the average message, we divide the result by the total number of messages (n masters sent $n-1$ messages each). Similarly, the value is calculated for the rectangular shape, where the message routes obey to Constraint 2 (Eq. 14). Finding the maximum message distance for both shapes is trivial (Eqs. 15, 16).

$$\text{Line-AVG} = \frac{\sum_{i=1}^{n}\left(\sum_{k=1}^{i-1} k + \sum_{j=1}^{n-i} j\right)}{n(n-1)} = \frac{n+1}{3} \tag{13}$$

$$\text{Rect-AVG} = \frac{\sum_{i=1}^{\left\lceil \frac{n-1}{2} \right\rceil} i + \sum_{i=1}^{\left\lfloor \frac{n-1}{2} \right\rfloor} i}{n-1} = \frac{n^2}{4(n-1)} \tag{14}$$

$$\text{Line-MAX} = n-1 \tag{15}$$

$$\text{Rect-MAX} = \left\lceil \frac{n-1}{2} \right\rceil \tag{16}$$

Figure 11 illustrates how the number of dispatchers influences the distance traversed by the average and the longest intra-application message of these two shape types. The rectangular shape type performs better than the line one, for both the average and the longest message, but at the expense of potential reroutings, which do not occur in the latter case. The decision regarding the shape type precedence can be made by the system designer, or left to the mapping process to decide.

In order to estimate the penalty of constrained intra-application message routes, traversal distances of the average and the longest message are computed for a rectangular shape with the free point-to-point communication between dispatchers (Eqs. 17, 18), and subsequently plotted in Fig. 11. Equation 17 is derived by using the intermediate results of Eq. 13.

$$\text{Free-AVG} = \frac{2\sum_{i=1}^{\frac{n}{2}}\left(\sum_{k=1}^{i-1} k + \sum_{j=1}^{\frac{n}{2}-i} j\right) + \frac{n^2}{4}}{\frac{n}{2}(n-1)} = \frac{n^2+3n-4}{6(n-1)} \tag{17}$$

Fig. 11 Shape comparison

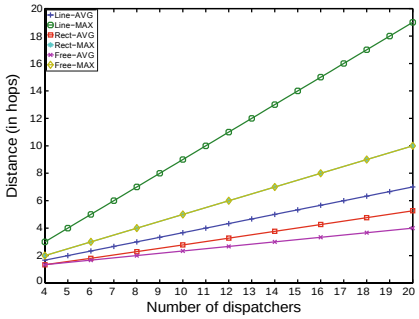
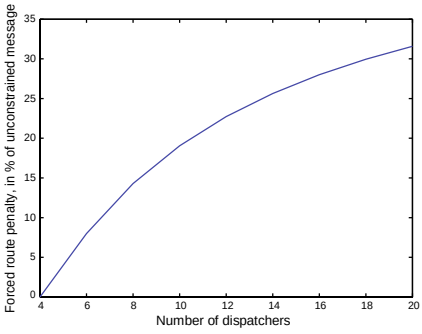


Fig. 12 Rerouting penalty estimation



$$\text{Free-MAX} = \left\lceil \frac{n-1}{2} \right\rceil \tag{18}$$

Figure 11 demonstrates that the removal of Constraint 2 improves the performance for the average message, but keeps the same longest message and, as recognised in Sect. 5.4, significantly complicates the analysis. Unlike Fig. 11, which visualises the overhead of forced paths expressed in absolute values, Fig. 12 depicts the same overhead expressed relatively. In practical terms, Constraint 2 causes a 19 % longer traversal path for the average intra-application message of a 10-dispatcher application with a rectangular shape.

8.2 Analysis and workload parameters

In the subsequent experiments we perform the mapping process, assuming the workload synthetically generated by using the parameters from Table 2. An asterisk sign denotes a randomly generated value, assuming a uniform distribution. To assure that all considered application-sets are indeed feasible, the individual per-application constraints on the worst-case communication delays were derived as follows. First, each application-set is mapped with the parameters $\forall a_i \in A : W_i = T_i - R_i^{min}$, and $P = 0$, where R_i^{min} was introduced in Sect. 7.4. This approach assures that all applications will be mapped during the HP phase with shapes that correspond to narrow mappings, and will consequently suffer small worst-case communication delays,

Table 2 Analysis and workload parameters

Routing delay + link transfer delay	3 + 1 cycles
Link bandwidth	16 bytes
Application utilisation	[0–0.7]*
Application periods	[30 mS–1 S]*
Application migration coefficient	[0–50]*
Number of dispatchers (migrative application)	[2–10]*
Agreement message size	64 bytes
Inter-application message size	[16–64]* bytes
Migrative applications	50 %
Probability of inter-application communication	5 %

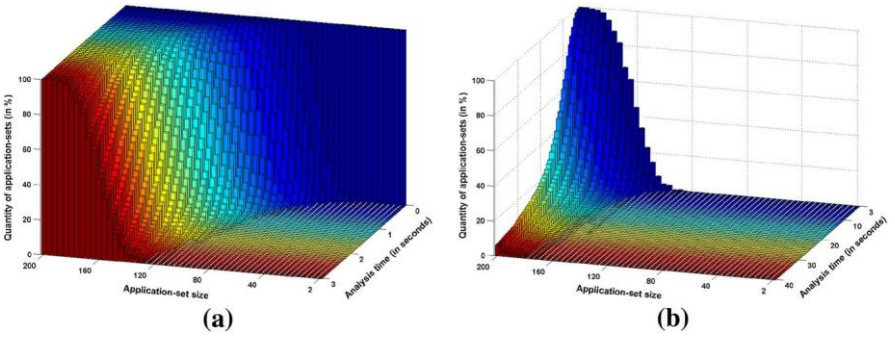


Fig. 13 The influence of the application-set size on the analysis time

called *narrow shape delays* (NSDs), hereafter. Thus, by assigning each application a communication constraint which is equal to, or greater than its respective NSD (i.e. $W_i \geq \text{NSD}_i$), we know that the given application-set is feasible with at least one selection of the mapping parameters ($P = G = 0$). In the subsequent experiments, we set the communication constraints to be equal to multiples of NSDs of respective applications, i.e. $\forall a_i \in A : W_i = n \cdot \text{NSD}_i$. Notice, that a bigger value of n gives applications more "freedom" to claim wider shapes, but also decreases the schedulability potential of applications, because bigger W_i also means tighter job deadlines ($D_i = T_i - W_i$).

8.3 Case study 2: scalability

The objective of this case study is to test the scalability potential of the proposed approach. We vary the number of applications in the range [2–200]. For each given value of the application-set size we generate and map 1000 application-sets on a 8 × 8 grid, in accordance with the values from Table 2. The other parameters are: $G = 0.3$, $P = 0.5$ and $\forall a_i \in A : W_i = 10 \cdot \text{NSD}_i$. We perform the timing analysis by capturing the duration of the mapping process of each run.

Figure 13a demonstrates the results. The horizontal axis stands for the application-set size. For clarity purposes we present a reciprocal cumulative graph, where the

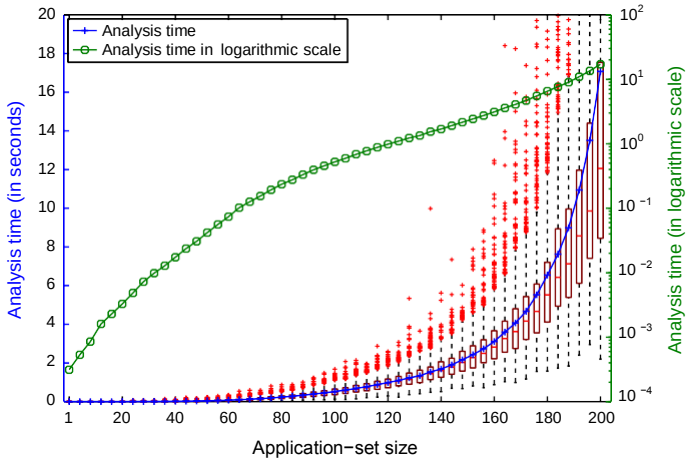


Fig. 14 The influence of the application-set size on the analysis time

vertical axis depicts the quantity of the runs that still did not complete the mapping process within a given time interval (depth axis). For small application-sets, almost all the runs complete very fast. As the number of applications increases, the mapping process takes more time, thus the slope of the curve flattens. At the end of the first observed period (0–3 s), almost all the smaller sets finished, while very few of the largest ones report the completion of the mapping process.

The second observed period (3–40 s) is presented in Fig. 13b. Most of the runs for application-sets up to 150 applications already finished, while the runs for the larger sets keep the completion rate steady. Note that the observation period is larger, thus the slope looks steeper. At the end of the observed period, only a small fraction of the largest application-sets did not complete.

The results show an obvious trend regarding the duration of the mapping process across application-set sizes, however, the mappings of two sets of the same size can report significantly different durations, sometimes by an order of magnitude. To emphasize this fact, we give a different representation of the same data in Fig. 14. The horizontal axis stands for the application-set size. The left and the right vertical axis represent the duration of the mapping process, in linear and logarithmic scale, respectively. Even though the whiskers were set to the 25th and 75th percentiles, which corresponds to the 99.3 % coverage for the normal distribution, it is visible that a non-negligible amount of runs falls outside the aforementioned area. This infers that the duration of the mapping process may hugely vary and highly depends on the application-set parameters.

Overall, the duration of the mapping process exponentially increases with the number of applications. However, it is averaging at 12 seconds for the sets consisting of 200 applications and we thus believe that the proposed approach is applicable to most of realistic scenarios in the real-time embedded domain.

We performed another scalability test by varying the grid size (from 8 × 8 to 16 × 16), while keeping all the other parameters at the same values. The size of the application-set is 150. Thousand sets were generated and subsequently mapped. Figure 15a shows

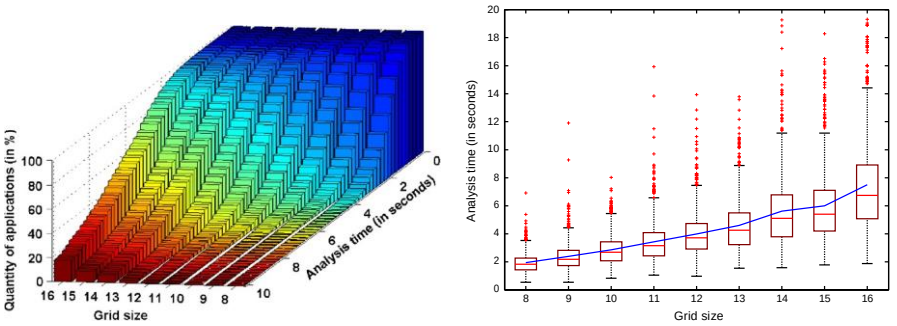


Fig. 15 The influence of the grid size on the analysis time

how the variation of the grid size (horizontal axis) influences the analysis time (depth axis). A cumulative reciprocal representation is used, where vertical axis stands for the number of application-sets for which the mapping did not finish within a given time.

It is visible that, within 6 s, in almost all cases the mapping completed for smaller grids, while larger grids are more time consuming and report fewer completions in the same time interval. Conversely to application-set size variations, the grid size variations cause a linear increase in the duration of the mapping process, visible in Fig. 15b. Note that these results infer that on average, the mapping process on a platform with 144 cores takes only 2 times more than on a 64-core one, assuming the same workload is mapped in both cases. The explanation is as follows. Even though larger grids require more cores to be checked while mapping, at the same time this gives the opportunity to map the applications in such a way that complex interference scenarios are avoided, which decreases the duration of feasibility rechecks, since fewer contentions occur. The number of outliers again confirms huge variations in duration times of the mapping process, demonstrating that for some specific application-sets the duration of the mapping process can significantly exceed the average time needed for that particular workload and platform size.

8.4 Case study 3: parameter P

The parameter P controls the amount of applications which will be grouped in H and hence mapped during IP, while the rest of the applications will undergo a two-stage mapping process in FP and OP. Note that FP and OP are computationally more intensive than IP, so mapping more applications during $FP + OP$ can cause a significant increase in the computation time. However, this process is more thorough in search and potentially has higher chances of finding better application mappings. Conversely, mapping most of the applications during IP may save the computation time, but makes the efficiency of the mapping process highly dependant on the right selection of the parameter G, as is shown in Case Study 4. Thus, an intuitive assumption is that the selection of the parameter P creates a trade-off between the analysis time and the solution quality.

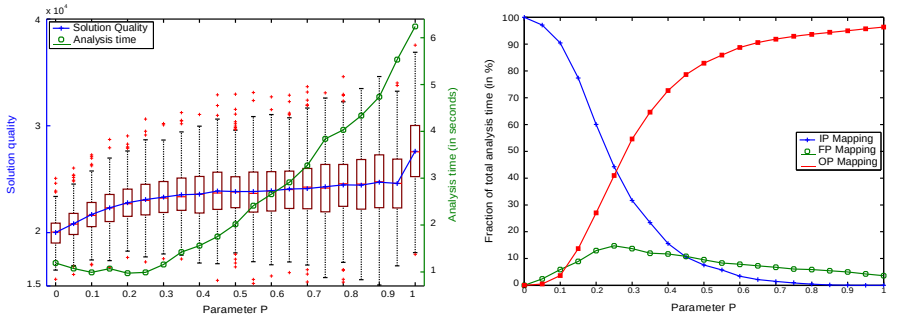


Fig. 16 The influence of the parameter P on the mapping process

Assuming $G = 0.3$ and $\forall a_i \in A : W_i = 10 \cdot \mathrm{NSD}_i$, all the application-sets are feasible across the entire observed domain, where we vary the parameter P in the range $[0 \ldots 1]$. The application-set size is 150, and 1,000 sets are created and mapped on a 8×8 platform, for each incremental step of P, as shown in Fig. 16a. We perform the timing analysis (the right vertical axis), but also collect the qualities of generated solutions (the left vertical axis), since the objective is to observe the effect of P on both the analysis time and the solution quality.

As expected, the increase of the parameter P causes the duration time of the mapping process to grow exponentially. As P increases, more applications undergo a more computationally extensive mapping process, which results in longer analysis times. However, the results report a logarithmic growth of the solution quality, almost on the entire domain. The only exception is the case when $P = 1$, that is, when all applications are mapped during FP and OP.

The results suggest that putting more applications in L and placing them during FP and OP might not produce a solution with a significantly better quality. The explanation is that the final mapping phase OP sorts the applications non-increasingly by the parameter I and tries to optimise their placements in that order. Since OP does not have a greediness control mechanism, every application will greedily assume the widest possible mapping such that its feasibility is preserved. Thus, most of the available network resources are consumed by the applications that are considered early during OP, which leaves the ones considered later to be able to barely optimise their placements, if at all. Another interesting conclusion is that mapping some applications during IP (i.e. $P < 1$) may prevent the OP phase to optimise the mapping in the most efficient way. This is confirmed with the substantial increase in the solution quality when all applications undergo the optimisation phase ($P = 1$).

In order to gain a more detailed insight into the influence of P on the duration of the mapping process, we observed the durations of the individual mapping phases. Figure 16b shows the fraction of the total analysis time that each phase consumes and compares them in relative terms. For small values of P, almost all applications are mapped during IP, hence leaving less workload for the subsequent phases. Until $P = 0.25$, IP witnesses an exponential decrease of the analysis time, while the other two phases report a linear increase. Notice, that although all three stages have the

sub-quadratic complexity, as P increases the complexity of OP becomes a dominant factor, while $P > 0.75$ results in scenarios where the analysis time is almost entirely spent in OP. This is explained with the amount of necessary feasibility rechecks, which in the first two stages occur rarely and cause their almost linear complexity, while in the last mapping stage are performed regularly, thus contributing to the true sub-quadratic complexity.

8.5 Case study 4: parameter G

The parameter G controls the amount of greediness allowed to high-priority applications. An intuitive reasoning suggests that the greater value of G causes the greater amount of traffic as a consequence of spatially distributed mappings of applications from H. Consequently, low-priority applications, which get placed during FP and OP, will have less possibilities to assume well distributed placements, since the network resources were greedily consumed by the applications from H. In some cases this may cause applications from L to be unable to claim feasibility even with narrow mappings, resulting in a failure of the mapping process. Conversely, smaller G may limit the distribution of high-priority workload and unnecessary preserve the network for applications from L, while there might be only few of them. This can significantly impact the solution quality.

The effects of the parameter G highly depend on the parameter P, since G applies only to high-priority applications, which amount is directly controlled by the parameter P. For small values of P, the influence of G is amplified the most. As P increases, the effects of G mitigate and at some point become negligible. To better observe the impact of G on the mapping process, we kept $P = 0$ and $\forall a_i \in A : W_i = 10 \cdot \mathrm{NSD}_i$. Notice, that this parameter selection caused some application-sets to be infeasible on the entire domain, where we varied the parameter G in the range [0–1]. Thus, in this part of the experiment we only consider those application-sets which are feasible on the entire domain. The application-set size is 150, and 1,000 sets are created and mapped on a 8×8 platform, for each incremental step of G, as shown in Fig. 17a. We perform the timing analysis (the right vertical axis), but also observe qualities of derived solutions (the left vertical axis), so as to study the effect of G on both the analysis time and the solution quality.

It is visible that the duration of the mapping process is constant and does not depend on the parameter G. However, G has a significant effect on the solution quality. Specifically, as G increases, high priority applications gain more freedom in assuming mappings wider than narrow ones, resulting in a constant increase of the solution quality. This effect is noticeable until $G = 0.75$. For higher values of G, the solution quality starts to decrease, since high G allows greedy mappings of high-priority applications, and hence preserves less resources for the later mapping stages. In fact, for $G > 0.75$, the network is so greedily consumed by high-priority applications, that low-priority ones barely claim the feasibility with narrow mappings. Note that high values of G not only significantly limit the optimisation of lower priority workload during OP, but also severely affect the feasibility, as explained below.

As already mentioned, Fig. 17a considered only those application-sets which were feasible across the entire domain of G. Now we investigate the impact of the parameter

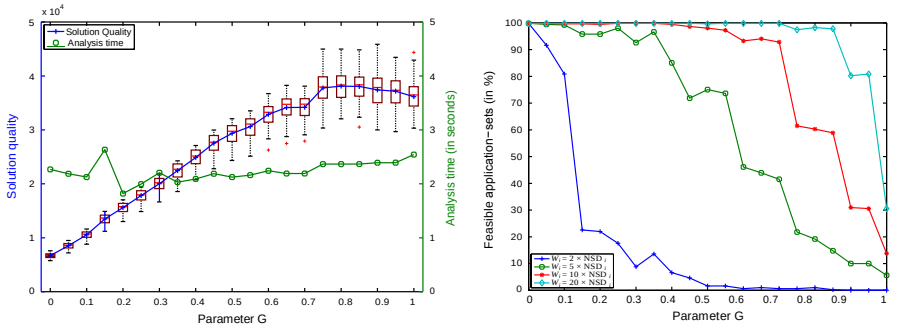


Fig. 17 The influence of the parameter G on the mapping process

G on the feasibility (Fig. 17b). The application-set size is 150, and 1,000 sets are created. For each generated application-set we vary the individual per-application communication constraints by allowing them to be 2, 5, 10 and 20 times greater than the respective NSDs. Subsequently, assuming again $P = 0$, we map them on a 8×8 platform and observe how the number of feasible application-sets (the vertical axis) changes with the parameter *G* (the horizontal axis).

It is visible that the parameter *G* significantly impacts the feasibility, and as *G* increases, the number of feasible application-sets decreases. This is expected, because giving more freedom to high-priority applications allows them to more greedily consume the available network resources. This inevitably leaves low-priority applications with fewer resources, which, in many cases are not enough to claim the feasibility even with narrow mappings. Notice, that even if the communication constraints are 20 times greater than the respective NSDs, when $G = 1$ only 30 % of the application-sets are feasible.

8.6 Discussion

The efficiency of the mapping process significantly depends on the right selection of the parameters *P* and *G*. But more than that, it highly depends on the parameters of the application-set upon which it is being applied. As observed through the experiments, there are no individual values of *P* and *G* which derive the best solution for every application-set. We used the experiments to identify and explain the general trends associated to these parameters, but also to show that different mapping strategies can in some cases provide competitive results, and conversely, the same mapping strategy may vary substantially in terms of efficiency when applied to different cases.

For instance, for sets with *tight communication constraints*, setting low values to *P* and *G* will very fast produce a feasible mapping but without a significant quality. Increasing *P* in such cases may result in a solution with the better quality, but at the expense of the additional time. If the computational complexity is not the problem, in such cases setting $P = 1$ is the preferable option. For sets where applications have *similar communication constraints* the best strategy is to invoke the balanced network consumption by keeping *P* low and *G* low or moderate, depending on the tightness

of the constraints. Conversely, the sets with *significant differences in communication constraints* will benefit the most from a spatial partialisation approach invoked by moderate or high values of P and high values of G. This parameter selection will allow only a limited number of applications to claim wide mappings and utilise the routes on the boundaries of the grid, and at the same time preserve the central cores free for the applications with tight constraints. Finally, for sets with *relaxed communication constraints* setting P low or moderate and G high may result in a mapping with the good solution quality, but may also deem the application-set infeasible with that particular parameter selection. In such cases, decreasing the parameter G until the application-set becomes feasible is a preferable option.

The advantage of the proposed method is that by setting P and G low, a feasible solution can be derived very fast. This option may be useful in scenarios where limited computational capacities are available, and the system designer is not very knowledgeable about the nature of the workload. Alternatively, if additional computational capacities are available, the designer might choose to increase P, until reaching a desirable trade-off between the solution quality and the computational complexity. Moreover, if the designer knows workload characteristics (e.g. the number of applications, the number of dispatchers, the tightness of the constraints), he can take that knowledge into account and make an informed decision regarding the parameter G, which may additionally improve the solution quality.

9 Conclusions and futurework

NoC-based many-core platforms are the next frontier technology in the real-time embedded domain. Their design not only allows cost reductions and power savings, but also contributes to the overall system flexibility. In this work we focus on many-cores using a *limited migrative model* (LMM). LMM extends the concepts of the multi-kernel paradigm, which is a novel OS design and a promising step towards scalable and predictable many-cores. The main contribution of this work is that we formulate the problem of application mapping on a many-core platform using LMM, and propose a three-stage process to solve it. The extended version of the existing analysis is utilised to assure that derived solutions (i) guarantee the fulfilment of timing constrains posed on worst-case communication delays of individual applications and (ii) provide an environment to perform load balancing for various beneficial reasons, e.g. energy/thermal management, performance enhancements or fault tolerance.

Regarding the future work, we plan to extend the approach to additionally consider memory operations of individual applications. Moreover, we plan to study the schedulability aspects of LMM and propose the schedulability analysis which will be considered during the application mapping process. Both these activities would help to derive a more complete analysis framework for LMM, which is a promising step towards the integration of many-core platforms into the real-time domain.

Appendix

Theorem 1 *Let* $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}$ *be a set of real number variables, such that the following holds:*

$$\sum_{\forall x_i \in X} x_i = C \tag{19}$$

$$x_i \geq 0, \forall x_i \in X \tag{20}$$

The function $f(X) = \prod_{\forall x_i \in X} x_i$ *has only one maximum on the domain, and that is the point:* $x_1 = x_2 = \cdots = x_n = \frac{C}{n}$.

Proof Proven directly. This is a constrained optimisation problem, with one constraint expressed by the equality (Eq. 19), and *n* constraints expressed by the inequalities (Inequality 20). If we (temporarily) exclude the inequalities from consideration, the extreme values of the function *f (X)*, subject to the equality constraint, can be found by the *Lagrange Multipliers Method*.

$$f(X) = \prod_{\forall x_i \in X} x_i, \quad g(X) = \sum_{\forall x_i \in X} x_i = C \quad \Rightarrow \quad \mathcal{L} = \prod_{\forall x_i \in X} x_i + \lambda \cdot \left(C - \sum_{\forall x_i \in X} x_i \right) \tag{21}$$

A new variable λ is called the Lagrange multiplier. According to the first derivative test, the necessary condition for the extreme point is that the partial derivative of the Lagrange function with respect to $\forall x_i \in X$ and λ is equal to 0 (see Eqs. 22–24).

$$\frac{\partial \mathcal{L}}{\partial x_1} = \frac{\partial f(X)}{\partial x_1} - \lambda \cdot \frac{\partial g(X)}{\partial x_1} = 0 \quad \Rightarrow \quad \prod_{\forall x_i \in X \setminus \{x_1\}} x_i = \lambda \tag{22}$$

$$\vdots$$

$$\frac{\partial \mathcal{L}}{\partial x_n} = \frac{\partial f(X)}{\partial x_n} - \lambda \cdot \frac{\partial g(X)}{\partial x_n} = 0 \quad \Rightarrow \quad \prod_{\forall x_i \in X \setminus \{x_n\}} x_i = \lambda \tag{23}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \quad \Rightarrow \quad \sum_{\forall x_i \in X} x_i = C \tag{24}$$

We analyse two cases: (1) $\lambda = 0$ and (2) $\lambda \neq 0$.

(1) $\lambda = 0$: This is possible only if at least two of the variables are also equal to 0, that is $\exists x_i \in X, \exists x_j \in X \setminus x_i = x_j = 0 \wedge i \neq j$. There exists an infinite amount of these points and they are both critical and stationary, and therefore should be further examined by the second derivative test.

(2) $\lambda \neq 0$: There exists only one point and that is $x_1 = x_2 = \cdots = x_n = \frac{C}{n}$. This point is also critical and stationary. It also holds for this point that it can be examined by the second derivative test.

Additionally, due to the inequality constraints ($x_i \geq 0, \forall x_i \in X$), it is necessary to check the boundaries of the solution space as well. The boundaries are represented with the solutions where only one of the variables is 0, (i.e. $\exists x_i \in X \wedge / \exists x_j \in X \mid x_i = x_j = 0 \wedge i \neq j$). Those are called boundary points and there exists no test to

Fig. 18 General Form of Bordered Hessian for *n* Variables and 1 Constraint

$$\begin{bmatrix} 0 & \frac{\partial g}{\partial x_1} & \frac{\partial g}{\partial x_2} & \cdots & \frac{\partial g}{\partial x_n} \\ \frac{\partial g}{\partial x_1} & \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial g}{\partial x_2} & \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g}{\partial x_n} & \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Fig. 19 Bordered Hessian for

$$\lambda = 0$$

$$\begin{bmatrix} 0 & 1 & \cdots & 1 & \cdots & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 & \cdots & z_{ij} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & \cdots & z_{ji} & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}$$

prove their properties, they have to be individually checked. In our case it is easy; it is obvious that those points represent the minima on the domain, since $f(X) = 0$ for all of them.

Now we proceed with the second derivative test for the cases (1) and (2). It is conducted in the form of the *Bordered Hessian*. The process consists of finding the first partial derivatives of $g(X)$ with respect to $\forall\, x_i \in X$, then finding the second partial derivatives of $f(X)$ also with respect to $\forall\, x_i \in X$ and finally putting them into the matrix called the Bordered Hessian. The general form of the Bordered Hessian is represented by Fig. 18.

(1) $\lambda \neq 0$: Fig. 19 presents the Bordered Hessian for the case where $\lambda = 0$. The variable z_{ij} stands for the second partial derivative with respect to the variables x_i and x_j and is described by Eq. 25. z_{ij} has a non-zero value only in cases where at most two variables are equal to zero, otherwise it is also equal to zero. A sufficient condition for the local maximum is that the Bordered Hessian is negative definite, i.e. the determinants of its principal minors alternatively change their signs (Eq. 26). However, from Fig. 19 it is visible that there always exists some $|H_i| \neq 0$, thus making this test inconclusive. In such cases, each of the points should be examined individually. Yet, in our case it is obvious that these points represent the minima on the domain, since $f(X) = 0$ for all of them.

$$z_{ij} = z_{ji} = \prod_{\forall x_k \in X \setminus \{x_i, x_j\}} x_k \tag{25}$$

$$|H_1| < 0, |H_2| > 0, \cdots \quad \Rightarrow \quad sign(|H_i|) = (-1)^i, \forall i \in \{1, \ldots, n\} \tag{26}$$

Fig. 20 Bordered Hessian for $\lambda \neq 0$

$$\begin{bmatrix} 0 & 1 & 1 & \dots & 1 & \dots & 1 \\ 1 & 0 & z_{12} & \dots & z_{1i} & \dots & z_{1n} \\ 1 & z_{21} & 0 & \dots & z_{2i} & \dots & z_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & z_{i1} & z_{i2} & \dots & 0 & \dots & z_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_{n1} & z_{n2} & \dots & z_{ni} & \dots & 0 \end{bmatrix}$$

(2) $\lambda \neq 0$: The Bordered Hessian for this case is represented by Fig. 20. For z_{ij} also holds Eq. 25, however, these are all non-zero values in this case. Note that since $x_i = \frac{C}{n}, \forall x_i \in X$, all the second partial derivatives are also equal (Eq. 27).

$$z_{ij} = z_{ji} = z = \left(\frac{C}{n}\right)^{n-2}, \quad \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\} \mid i \neq j \tag{27}$$

When computed, the determinants of the principal minors are $|H_1| = 2z$, $|H_2| = -3z^2$, $|H_3| = 4z^3, \dots, |H_n| = (-1)^n n z^n$. Since both n and z are strictly positive, the sign of the determinant depends only on the first term of the product: $(-1)^i$, and therefore alternatively changes when different principal minors are considered. This fulfils the sufficient condition for the maximum, so we conclude that the function $f(X)$ has one maximum on the domain, which is located in the point $x_1 = x_2 = \dots = x_n = \frac{C}{n}$ and the value is $max(f(X)) = \left(\frac{C}{n}\right)^n$. □

Theorem 2 *Let $X = \{x_1, x_2, \dots, x_n\} \in \mathbb{N}$ be the distances between the neighbouring dispatchers of an application, and let C be the circumference of the application shape. The product of the distances between the neighbouring dispatchers $f(X) = \prod_{\forall x_i \in X} x_i$ reaches the maximum when all the distances are as even as possible.*

Proof Proven directly. Note that the distances between the dispatchers are natural numbers, which is a subset of real numbers. We analyse two cases:

(1) $\frac{C}{n} \in \mathbb{N}$: In this scenario the results of Theorem 1 hold, i.e. the maximum on the continuous domain of real numbers (superset) is also the maximum on the discontinuous domain of natural numbers (subset). Therefore, in these scenarios the function $f(X)$ reaches the maximum when all the distances are equal, i.e. $\forall x_i \in X : x_i = \frac{C}{n}$.

(2) $\frac{C}{n} \notin \mathbb{N}$: In this case the maxima are different. As proven in Theorem 1, $f(X)$ has a unique maximum on a continuous real-number domain, inferring that the function is monotonically increasing from the boundary to the extremum, with respect to each variable, when treating all other variables as constants. Therefore, the maximum on a discontinuous natural-number domain is the point geometrically the closest to the continuous maximum, which corresponds to a set of solutions (multiple maxima) on a discontinuous domain where the distances between dispatchers are either $\lceil \frac{C}{n} \rceil$ or $\lfloor \frac{C}{n} \rfloor$, i.e. $\forall x_i \in X : x_i \in \{\lceil \frac{C}{n} \rceil, \lfloor \frac{C}{n} \rfloor\}$.

References

Ascia G, Catania V, Palesi M (2004) Multi-objective mapping for mesh-based noc architectures. In: Proceedings of the 2nd international conference on hardware/software codesign and system synthesis, pp. 182–187

Baker TP (2006) An analysis of fixed-priority schedulability on a multiprocessor. Real-Time Syst J 32:49–71

Baruah S, Baker T (2008) Schedulability analysis of global edf. Real-Time Syst J 38:223–235

Bastoni A, Brandenburg B, Anderson J (2010) An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers. In: Proceedings of the 31st IEEE real-time systems symposium

Baumann A, Barham P, Dagand PE, Harris T, Isaacs R, Peter S, Roscoe T, Schüpbach A, Singhania A (2009) The multikernel: a new os architecture for scalable multicore systems. In: ACM symposium on operating systems principles

Benini L, De Micheli G (2002) Networks on chips: a new soc paradigm. Comput J 35(1):70–78

Bletsas K, Andersson B (2009) Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. In: Proceedings of the 30th IEEE real-time systems symposium

Burns A, Wellings A (2009) Real-time systems and programming languages. Addison-Wesley Educational Publishers Inc

Calandrino J, Anderson J, Baumberger D (2007) A hybrid real-time scheduling approach for large-scale multicore platforms. In: Proceedings of the 19th euromicro conference on real-time systems

Chou CL, Marculescu R (2007) Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. In: Proceedings of the 5th international conference on hardware/software codesign and system synthesis

Chou CL, Marculescu R (2008) Contention-aware application mapping for network-on-chip communication architectures. In: Proceedings of the international conference on computer design

Chou CL, Ogras U, Marculescu R (2008) Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels. IEEE Trans Comput Aided DesIntegr Circuits Syst 27(10):1866–1879

Dally W (1992) Virtual-channel flow control. IEEE Trans Parallel Distrib Syst 3(2):194–205

Dally W, Seitz C (1987) Deadlock-free message routing in multiprocessor interconnection networks. IEEE Trans Comput 36:547–553

Duato J, Yalamanchili S, Lionel N (2002) Interconnection networks: an engineering approach. M.K. Publishers

Hu J, Marculescu R (2003) Energy-aware mapping for tile-based noc architectures under performance constraints. In: Proceedings of the 8th Asia and South Pacific design automation conference

Hu J, Marculescu R (2003) Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In: Proceedings of the 6th conference on design automation and test in Europe

Hung W, Addo-quaye C, Theocharides T, Xie Y, Vijaykrishnan N, Irwin M (2004) Thermal-aware ip virtualization and placement for networks-on-chip architecture. In: Proceedings of the international conference on computer design

Intel: Single-Chip-Cloud Computer. www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-article.pdf. Accessed September 2014

Kato S, Yamasaki N, Ishikawa Y (2009) Semi-partitioned scheduling of sporadic task systems on multiprocessors. In: Proceedings of the 21st Euromicro conference on real-time systems

Kavaldjiev NK, Smit GJM (2003) A survey of efficient on-chip communications for soc. In: Proceedings of the 4th symposium on embedded systems

Kreutz M, Marcon C, Carro L, Calazans N, Susin A (2005) Energy and latency evaluation of noc topologies. In: Proceedings of the international symposium on circuits and systems, Vol. 6, pp. 5866–5869

Kumar R, Mattson T, Pokam G, van der Wijngaart R (2011) The case for message passing on many-core chips. In: Multiprocessor system-on-chip. Springer, New York

Lei T, Kumar S (2003) A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In: Proceedings of the Euromicro symposium on digital systems design, p. 180

Liu C, Layland J (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61

Lundstrom M (2003) Moore's law forever? Science 299:210–211

Marcon C, Borin A, Susin A, Carro L, Wagner F (2005) Time and energy efficient mapping of embedded applications onto nocs. In: Proceedings of the 10th Asia and South Pacific design automation conference, vol. 1, pp. 33–38

Mesidis P, Indrusiak L (2011) Genetic mapping of hard real-time applications onto noc-based mpsocs—a first approach. In: 6th International workshop on reconfigurable communication-centric systems-on-chip

Moein-darbari F, Khademzade A, Gharooni-fard G (2009) Cgmap: a new approach to network-on-chip mapping problem. IEICE Electron. Express 6(1):27–34

Murali S, De Micheli G (2004) Bandwidth-constrained mapping of cores onto noc architectures. In: Proceedings of the 7th conference on design automation and test in Europe, p. 20896

Nikolic´ B, Yomsi PM, Petters SM (2014) Worst-case communication delay analysis for many-cores using a limited migrative model. In: Proceedings of the 20th IEEE conference on embedded and real-time computing and applications

Ni LM, Mckinley PK (1993) A survey of wormhole routing techniques in direct networks. Comput J 26:62–76

Nikolic´ B, Ali HI, Petters SM, Pinho LM (2013) Are virtual channels the bottleneck of priority-aware wormhole-switched noc-based many-cores? In: Proceedings of the 21th international conference on real-time networks and systems

Nikolic´ B, Petters SM (2012) Towards network-on-chip agreement protocols. In: Proceedings of the 12th international conference on embedded software

Racu A, Indrusiak L (2012) Using genetic algorithms to map hard real-time on noc-based systems. In: 7th International workshop on reconfigurable communicationcentric systems-on-chip

Sahu PK, Chattopadhyay S (2013) A survey on application mapping strategies for network-on-chip design. J Syst Arch 59:60–76

Shi Z, Burns A (2008) Real-time communication analysis for on-chip networks with wormhole switching. In: International symposium on networks-on- chip

Shi Z, Burns A (2009) Real-time communication analysis with a priority share policy in on-chip networks. In: Proceedings of the 21st Euromicro conference on real-time systems

Shi Z, Burns A (2010) Schedulability analysis and task mapping for real-time on-chip communication. Real-Time Syst J 46(3):360–385

Song H, Kwon B, Yoon H (1997) Throttle and preempt: a new flow control for real-time communications in wormhole networks. In: Proceedings of the 1997 international conference on parallel processing

Srinivasan K, Chatha K (2005) A technique for low energy mapping and routing in network-on-chip architectures. In: Proceedings of the international symposium on low power electronics and design

Tilera: TILE64™ Processor. www.tilera.com/products/processors/TILEPro_Family. Accessed September 2014

Wentzlaff D, Agarwal A (2009) Factored operating systems (fos): the case for a scalable operating system for multicores. SIGOPS Oper Syst Rev 42(3):76–85

Li Y, Danish M, West R (2014) Quest-v: a virtualized multikernel for high-confidence systems. Technical report http://www.cs.bu.edu/~richwest/quest.html. Accessed September 2014

Zimmer C, Mueller F (2012) Low contention mapping of real-time tasks onto tilepro 64 core processors. In: Proceedings of the 18th IEEE real-time technology and applicationssymposium