Technical Report

Application Mapping In NoC-Based Many-Cores

Borislav Nikolic

Stefan M. Petters

HURRAY-TR-121201

Version:

Date: 12-03-2012

Application Mapping In NoC-Based Many-Cores

Borislav Nikolic, Stefan M. Petters

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

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

Abstract

Many-core platforms based on Network-on-Chip (NoC [Benini and De Micheli 2002]) present an emerging technology in the real-time embedded domain. Although the idea to group the applications previously executed on separated single-core devices, and accommodate them on an individual many-core chip offers various options for power savings, cost reductions and contributes to the overall system flexibility, its implementation is a non-trivial task. In this paper we address the issue of application mapping onto a NoC-based many-core platform when considering fundamentals and trends of current many-core operating systems, specifically, we elaborate on a limited migrative application model encompassing a message-passing paradigm as a communication primitive. As the main contribution, we formulate the problem of real-time application mapping, and propose a three-stage process to efficiently solve it. Through analysis it is assured that derived solutions guarantee the fulfilment of posed time constraints regarding worst-case communication latencies, and at the same time provide an environment to perform load balancing for e.g. thermal, energy, fault tolerance or performance reasons. We also propose several constraints regarding the topological structure of the application mapping, as well as the inter- and intra-application communication patterns, which efficiently solve the issues of pessimism and/or intractability when performing the analysis.

A

Application Mapping In NoC-Based Many-Cores

BORISLAV NIKOLIĆ, CISTER/INESC-TEC and ISEP, IPP
STEFAN M. PETTERS, CISTER/INESC-TEC and ISEP, IPP

Many-core platforms based on Network-on-Chip (NoC [Benini and De Micheli 2002]) present an emerging technology in the real-time embedded domain. Although the idea to group the applications previously executed on separated single-core devices, and accommodate them on an individual many-core chip offers various options for power savings, cost reductions and contributes to the overall system flexibility, its implementation is a non-trivial task. In this paper we address the issue of application mapping onto a NoC-based many-core platform when considering fundamentals and trends of current many-core operating systems, specifically, we elaborate on a limited migrative application model encompassing a message-passing paradigm as a communication primitive. As the main contribution, we formulate the problem of real-time application mapping, and propose a three-stage process to efficiently solve it. Through analysis it is assured that derived solutions guarantee the fulfilment of posed time constraints regarding worst-case communication latencies, and at the same time provide an environment to perform load balancing for e.g. thermal, energy, fault tolerance or performance reasons. We also propose several constraints regarding the topological structure of the application mapping, as well as the inter- and intra-application communication patterns, which efficiently solve the issues of pessimism and/or intractability when performing the analysis.

Categories and Subject Descriptors: C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms: Real-time, Many-core, Application-mapping

1. INTRODUCTION

Although the advancements in the semiconductor area made it feasible to accommodate an enormous amount of transistors on a single chip, in recent years the processing units have hit a limit in the clock speed. Among others, one reason is the inability to cope with the energy dissipated and thus increasing temperature at high frequencies. One solution to ever increasing requirements for more powerful processing devices is to integrate several cores into one system. Architectures that contain more than a dozen cores placed within a single chip are commonly known as many-cores and are emerging technologies that are becoming widely popular and used in many areas. For instance, the progression steps from single- to many-core devices were performed in the supercomputing area many years ago and, with a slight offset, a similar trend is noticeable in general purpose computing. Current real-time embedded systems are mostly single-cores, however the demands for functionality enhancements, money savings and power conservation are drivers for changes mirroring that of High-End-Computing area.

Although many-cores look suitable to address most of the aforementioned issues, the integration of said devices into real-time embedded domain is far from trivial. The most distinguishable problem is complex and/or pessimistic analysis. Since the efficiency of majority real-time devices highly depends on derived guarantees, providing an analysis with acceptable both complexity and pessimism presents a fundamental prerequisite. In this paper we present the application mapping procedure and the accompanying analysis which, with controllable complexity and negligible amount of pessimism, allows to plan and organise the workload execution on a many-core platform by assuring that no violations of time constraints occur for real-time applications.

The novelty of our work is reflected in the fact that we study both real-time and best-effort workload mapping, assuming a migrative execution model. Our approach encompasses application migrations to support multiple drivers, namely energy and thermal management, fault tolerance, performance, accommodation of dynamic soft- and non-real-time workload. The platform under consideration is a many-core system, with

both inter- and intra-application communication centered around a message-passing paradigm, utilising 2D-mesh NoC interconnect.

2. RELATED WORK

In recent years the problem of application mapping in many-cores gained in popularity, as many researchers recognised its importance. However, all current state-of-the-art approaches assume a static pre-assignment of the workload to individual intellectual properties (e.g. CPU cores, DSP cores) and consider the problem of their placement within the network without the possibility to migrate. Even this simplified problem is equivalent to the quadratic assignment problem, which is NP-Hard, hence no optimal solution can be found within a reasonable time even for small NoC platforms, e.g. 4×4 cores [Hu and Marculescu 2003a]. In the rest of the section we firstly give an overview of the existing approaches and then briefly summarise on how this work differs.

In order to solve the mapping problem, [Lei and Kumar 2003] assumed different processor types and developed a two-step genetic algorithm where the workload is firstly mapped to a specific processor type and in the second pass to a concrete processor. [Moein-darbari et al. 2009] use a modification of the aforementioned heuristic, called chaos-genetic algorithm. By employing the branch-and-bound technique [Hu and Marculescu 2003a] investigate mappings which minimise 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 for mapping evaluation. [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 introduce communication timing characterisation (communication dependence and ordering) and [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 found (called Pareto mappings) so as to provide a solutions to the multi-objective approach. [Kreutz et al. 2005] explore the same topic but consider the architectures with the interconnect topologies other than NoC.

While the previous works assume a static pre-defined routing policy, [Hu and Marculescu 2003b] exploit the routing flexibility so as to reduce the routing energy consumption and improve the performance. [Shi and Burns 2010] propose a priority assignment algorithm for task mapping and introduce a flow-level analysis for network latencies, assuming wormhole-switched, priority-aware NoCs. [Marcon et al. 2008] give an overview of several mapping algorithms targeting low energy consumption.

Although similar to the cited, our approach in application mapping differs in a way that it assumes a migrative model and as such brings additional complexity. Our motivation is similar to that of [Hung et al. 2004] (thermal management) and [Hu and Marculescu 2003a] (power management). We assume the per-message latency constraints like [Srinivasan and Chatha 2005] and take into account the message ordering similar to [Marcon et al. 2005], so as to accommodate the real-time traffic and reduce the pessimism of the analysis. The physical links of the interconnect have limited capacities reflected by the bandwidth constraints, as proposed by [Murali and De Micheli 2004]. Finally, data transfer over NoC employs a priority-aware wormhole routing technique, which is similar to the approach of [Shi and Burns 2010].

3. PROBLEM FORMULATION

3.1. System Model

In order to formulate the problem, we formally describe the platform and the application workload. Of interest is one NoC-based many-core system, comprised of $n \times n$ tiles (each tile contains a router and a core), and an application-set residing in that system.

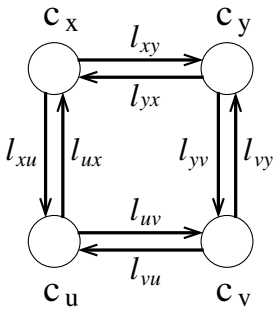


Fig. 1. RWG

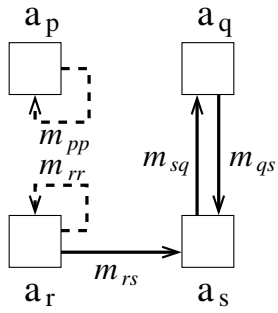


Fig. 2. CWG

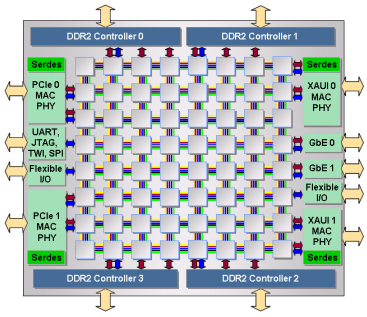


Fig. 3. TILE64 Many-core processor

Definition 1: The platform is described by a direct, resource-weighted graph $RWG = \langle \mathcal{C}, \mathcal{L} \rangle$, where vertices $\mathcal{C} = \{c_1, c_2, ..., c_{n^2-1}, c_{n^2}\}$ represent the cores in the system and the edges $\mathcal{L} = \bigcup l_{ij}$ denote the physical links of the interconnect. Every existing link $l_{ij} = (c_i, c_j, w_{ij}) | (c_i, c_j) \in \mathcal{C}$ between the cores c_i and c_j is described with w_{ij}, which stands for its physical characteristics (i.e. available bandwidth).

Applications are described by communication patterns. A traffic belongs to either inter- or intra-application communication. The former describes cross-application dependencies (data sharing, synchronisation), while the latter represents an execution of a per-application agreement protocol [Nikolić and Petters 2012], which purpose is to derive a decision regarding spacial and temporal characteristics of said application's future executions (i.e. will the migration occur and if so, which core is the destination).

Definition 2: The applications are described by a direct, communication-weighted graph $CWG = \langle \mathcal{A}, \mathcal{M} \rangle$, where vertices $\mathcal{A} = \{a_1, a_2, ..., a_{x-1}, a_x\}$ denote the application-set residing in the system, and the edges $\mathcal{M} = \bigcup m_{ij}$ symbolise the inter- or intra-application traffic. Every message $m_{ij} = (a_i, a_j, d_{ij}, p_{ij}, t_{ij}) | (a_i, a_j) \in \mathcal{A}$ between the applications a_i and a_j is characterised by the amount of data exchanged - d_{ij}, priority of the message - p_{ij} and optionally by a temporal (real-time) constraint - t_{ij}.

Figure 1 and Figure 2 illustrate RWG and CWG, respectively. The intra-application traffic (i.e. the one constituting the agreement protocols) has the same source and destination application and is represented in Figure 2 with dotted lines. Note, that the purpose of the agreement protocol is to elaborate on eventual migrations. Therefore, it exists only for applications which a system designer specifies as migrative (i.e. a_p and a_r in Figure 2). Also note, that we assume a limited migrative model, where every migrative application can execute only on a predefined set of cores, specified at design-time. These cores participate in the agreement protocol of a given application.

3.2. Problem Statement

The goal of this work is to develop a mapping procedure which, with tolerable computational complexity, finds a *feasible mapping* of a given application-set onto a particular many-core platform ($CWG \rightarrow RWG$). By feasible we assume a mapping that guarantees the fulfilment of all per-message temporal constraints and at the same time maximises the ability to perform load balancing via limited migrations motivated for example by thermal/energy management, fault tolerance, performance and service of additional dynamic non-real-time workload.

4. MODEL

4.1. Platform

The assumed platform is a generic NoC-based many-core system. The existing examples are experimental platforms such as Single-Chip-Cloud [Intel] and TILE64 family

of processors [Tilera] (Figure 3). The architecture we consider in this paper is comprised of $n \times n$ tiles, which are mutually connected via 2D-mesh interconnect. Each tile contains a router and a core. Data transfer on the mesh is performed by employing a deterministic, dimension-ordered XY routing policy. It is deadlock and livelock free [Hu and Marculescu 2003a] and is present in the aforementioned architectures. We assume a packet-based transfer technique called *wormhole routing* [Ni and McKinley 1993] which is widely applied for NoC-based many-cores, due to its good throughput and small buffering requirements. Finally, our approach encompasses wormhole traffic prioritisation and traffic preemptions, so as to accommodate the workload with different criticality levels. The implementation of said policy can be performed on present architectures by exploiting the existing virtual channels [Shi and Burns 2008].

4.2. Software

The execution workload is represented by the application-set. Each application is characterised by the individual priority, depending on the criticality of its execution. In this work applications are single-threaded, implemented as tasks and may communicate or synchronise with other applications. Multi-threaded workload can be modelled as several applications with inter-application dependencies. Data sharing is performed in the manner of message-passing paradigm - by an explicit exchange of messages. The rationale for this approach lies in the fact that current coherence mechanisms are inefficient in many-core domain due to scalability issues [Baumann et al. 2009].

4.2.1. Intra-application communication. Depending on their purpose, some applications may have the possibility to execute on multiple cores. As explained earlier, we assume a limited migrative model, where an application can execute only on a subset of cores. On each of the cores that can accommodate a particular application, its execution code exists, constituting an entity called *dispatcher*. Dispatchers of the same application communicate among themselves and decide temporal and spatial properties of future executions (i.e. will the migration occur, and if yes which is the destination core). Once the decision is made, the selected dispatcher will continue the execution on behalf of the application. This dispatcher is called *master dispatcher* and at any moment of time the application has only one. The other dispatchers are termed *slave dispatchers*.

Upon completing the execution, the master is responsible for initiating the communication with the slaves, so as to plan future executions. If the outcome of said communication is that the migration occurs - the master changes: i.e. a newly elected dispatcher becomes a master, while the old master becomes a slave. Perceived from application's perspective, containing dispatchers exchange one master token. We call this phenomenon *master volatility* and it has a number of implications which we elaborate on in subsequent sections.

The aforementioned intra-application communication, where dispatchers of the same application elect a new master, is called the *agreement protocol*. In previous work [Nikolić and Petters 2012] we have presented several agreement protocols and provided analytical steps on how to compute the worst-case duration of both, the entire agreement protocol and the individual messages constituting it. The actual policy of the employed agreement protocol is immaterial for the discussion in this paper, as our analysis only requires that the same is characterised by the exchanged messages and the individual temporal constraints on their latencies.

Our approach is motivated by, and similar to the concepts of a scalable many-core operating system called *Barrelfish* [Baumann et al. 2009]. In Figure 4 a graphical representation of an example application-set is given. Every application may execute only on cores where its dispatchers exist (connected with dotted arrows). Master dispatchers can be distinguished by a dot over their name.

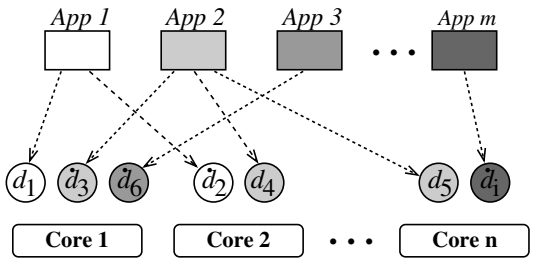


Fig. 4. Application model with limited migrations

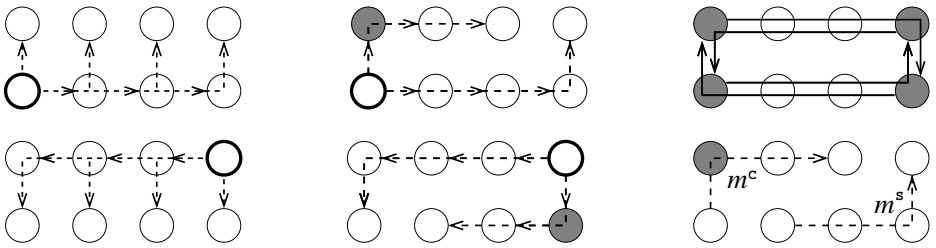
4.2.2. Inter-application communication. Besides the agreement protocol messages, our model also assumes the communication between different applications for various reasons, e.g. synchronisation and data sharing. The same is performed by the exchange of messages between current master dispatchers of communicating applications.

4.2.3. Traffic characterisation. Real-time applications have to fulfil posed time constraints, hence all intra- and inter-application traffic related to these applications has to traverse within a predefined time in order to avoid missed deadlines. In this paper we assume that the real-time traffic is characterised by a time interval t_{ij} in which it has to travel from the dispatcher of the source application a_i to the dispatcher of the destination application a_j. In case of agreement protocol messages, both interacting dispatchers belong to the same application, thus $i = j$. These per-message temporal constraints might be explicitly given, or can be extracted from applications. For instance, by specifying the employed agreement protocol and a time interval in which a migrative application has to complete it, individual time constraints for its agreement protocol messages can be derived using the findings of our previous work [Nikolić and Petters 2012]. In this paper we assume that the values of t_{ij} are specified for all real-time messages, while messages belonging to best-effort applications have $t_{ij} \to \infty$.

Besides temporal constraints, messages exchanged by interacting dispatchers of the applications a_i and a_j are characterised by the priority p_{ij}. For the agreement protocol traffic it holds that $i = j$ and it inherits the priority of the containing application a_i, while the inter-application traffic is characterised by the same of the lower priority application. In this paper we assume that applications and hence messages have already been characterised by their individual priorities prior to application mapping.

We classify the message m_{ij} as *feasible* if it is analytically proven that in the worst-case it can fulfil posed time constraint t_{ij}. In order to check the feasibility of a message, we firstly have to compute its worst-case latency when travelling from the source dispatcher belonging to the application a_i to the destination dispatcher of the application a_j, in the literature known as *worst-case traversal time* - $WCTT(m_{ij})$. A message is feasible when $WCTT(m_{ij}) \leq t_{ij}$.

$WCTT$ consists of several components. The first is the latency of the message traversal when there is no network interference - $L(m_{ij})$, in the literature known as *basic network latency*. The second is the direct interference caused by the higher priority traffic sharing some physical link with the analysed message - $D(m_{ij})$. Due to the wormhole routing technique, the message under analysis can additionally suffer the indirect interference caused by the higher priority traffic which does not share any links with it, but shares with the other traffic constituting either direct or indirect interference - $I(m_{ij})$. By allowing the preemptions on the bus, in every router on its path, the message of interest can be blocked by the lower priority traffic for at most one packet (flit) traversal time. Cumulative blocking delay on the whole path we denote by - $B(m_{ij})$. Finally, a potential re-routing of the message (see Section 5.1) causes additional overhead - $R(m_{ij})$. $WCTT$ is represented by Equation 1.



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

Fig. 5. Intra-application communication

$$WCTT(m_{ij}) \;=\; L(m_{ij}) + D(m_{ij}) + I(m_{ij}) + B(m_{ij}) + R(m_{ij}) \tag{1}$$

For obtaining $L(m_{ij})$, $D(m_{ij})$, $I(m_{ij})$ and $B(m_{ij})$ the reader is directed to previous works [Nikolić and Petters 2012], [Shi and Burns 2008] or [Ferrandiz et al. 2009]. The purpose of re-routing and the calculation of the incurred overhead cost $R(m_{ij})$ are elaborated in Section 5.1 and Section 5.3.

5. PROPOSED APPROACH

We augment the reasoning from the previous section and classify the solution as *feasible* if all messages constituting it are feasible - $\forall m_{ij} \in \mathcal{M} : WCTT(m_{ij}) \leq t_{ij}$. In our terminology, a solution presents a concrete placement of the application-set onto a given NoC-based many-core system.

The aim of this paper is to provide an application mapping procedure that will derive a feasible solution (when possible), which at the same time presents a suitable environment for load balancing. In other words, our approach is multi-objective with the primary concern of providing a feasible mapping, and the secondary objective being to place the applications in such a way that maximises their potential to efficiently migrate away from e.g. malfunctioning, overloaded, hot or hibernating core, when needed.

Performing an exact analysis on the limited migrative model by solving Equation 1 for each message and consequently checking its feasibility imposes significant overheads, causing either too pessimistic predictions or intractability, as is recognised and exhibited in the following subsections. To overcome this problem, we introduce several constraints regarding application topology and communication patterns.

5.1. Internal re-routing

As previously stated, in the assumed limited migrative model containing dispatchers discuss the properties of application's future executions via agreement protocols. The amount and the path of messages that constitute the agreement protocol depend on several factors: 1) the employed agreement protocol, 2) the number of dispatchers of an application 3) dispatcher placements, 4) the current master dispatcher.

The first aspect is addressed in our previous work [Nikolić and Petters 2012], and in this paper we assume that the agreement protocol is characterised by the generated messages. The second is specified by the system designer, while the third and the fourth aspect are elaborated in this paper.

Unlike dispatcher placement, which is determined at design-time, the actual master cannot be predicted due to the master volatility phenomenon. In the leftmost part of Figure 5 is given one example; the top and the bottom figure present generated agreement protocol messages of the same application, captured at different time instances (i.e. represented by emphasized circles different current masters communicate with all the slaves). The purpose of this example is to show that, depending on the master selection, two entirely different message-sets can be generated.

Now, let us elaborate on an exact feasibility analysis approach without the constraints introduced in this paper. As all intra-application messages comply with XY routing policy and exclude re-routings, it holds in Equation 1 that $R(m) = 0, \forall m \in \mathcal{M}$. Since the state-of-the-art works provide methods to obtain all the other terms constituting $WCTT$ in Equation 1, let us try to apply this approach to prove the feasibility of the intra-application traffic of an example application given in Figure 5. Each of 8 dispatchers might be a master, hence generating 7 different messages to communicate with all the slaves, in total $8 \times 7 = 56$ different messages exist. Proving the feasibility of intra-application traffic would require to prove the individual feasibility of each of these 56 messages. This holds only for the execution in isolation. In order to consider the interference caused by the traffic of other applications, two strategies are possible:

1) Assuming all possible messages of other applications and considering them as a potential interference will cause a significant pessimism build up due to the fact that not all intra-application messages may concurrently exist.

2) Specifying the master dispatcher for every interfering application and considering only the messages that may exist in that particular scenario would provide an exact worst-case, but only for that scenario. However, in order to capture a global worst-case, it is required to check all possible scenarios. Scenarios arise from the master volatility phenomenon and if there are, for example, x interfering applications, each with y dispatchers, then the number of possible scenarios is x^y. That is, every of the aforementioned 56 intra-application messages from Figure 5 would have to test its feasibility in all x^y different scenarios. It is easy to see that in practical cases an exponential complexity propagation very fast leads towards a combinatorial explosion.

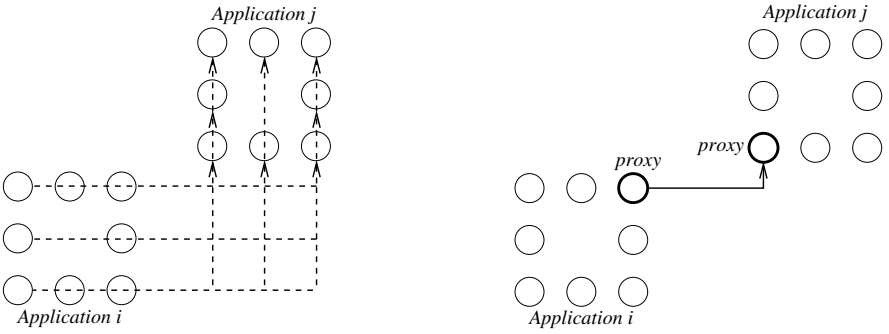
The approach we present solves this problem by enforcing the policies which make the intra-application traffic more deterministic and master-independent, namely **placement constraints** and **re-routing**:

Definition 3: Dispatchers of a migrative application can be placed only on the edges of a rectangular $a \times b$ structure, in such a way that no corner is left unoccupied and $a, b \in \mathbb{N}$. The special case is a line-like shape, which occurs when one or both dimensions of a shape are equal to "1".

Definition 4: Intra-application messages may travel only on the edge 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.

In the middle of Figure 5 the same example is elaborated, but with the routes generated by enforcing said constraints. Filled circles present the dispatchers where the potential re-routing occurs. Note, that enforced routes did not decrease the total number of messages, but made the part of the NoC infrastructure used by the intra-application traffic deterministic and master-independent. The impact of this approach on the performance is twofold; 1) re-routing causes additional overhead and 2) forced routes may be longer than those generated by the regular XY routing policy.

However, these rules allow generalisation of the intra-application traffic. In order to make the feasibility analysis possible, we introduce *supermessages*. A supermessage is a message which complies with XY routing policy and connects dispatchers placed in diagonal corners of the application shape. Due to possibility to assume a line- or rectangle-like shape, each application has 2 or 4 supermessages respectively. In the top right part of Figure 5 the same are provided for the elaborated example. If an inter-application message shares a part of the path with only one supermessage, we call it *simple message* - m^s. Otherwise, if it shares a part of the path with two supermessages, we call it *complex message* - m^c (see bottom right part of Figure 5). By transfering every simple message into its supermessage and every complex message into two corresponding supermessages the intra-application traffic can be described as a lin-



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

Fig. 6. Inter-application communication

ear combination of supermessages. This process dramatically decreases the number of different messages and hence makes the feasibility analysis tractable and possible, at the expense of produced pessimism. It arises from the fact that every simple message will be treated as a supermessage and every complex message as two supermessages, while some of them traverse only a small fraction of the path assumed by the analysis.

5.2. Proxy dispatchers

In this work the inter-application communication is modelled by the message exchange between the master dispatchers of the interacting applications. In situations where two non-migrative applications communicate, the path between the corresponding master dispatchers is predetermined and known at design time, once the mapping is derived. However, if one or both of the applications involved in the communication are considered as migrative, it induces an overhead to evaluate all possible combinations of master dispatchers in both the interacting applications.

Nevertheless, let us again elaborate on an exact feasibility analysis approach, without the constraints introduced in this paper. As all inter-application messages comply with XY routing policy and exclude re-routings, we can try to prove the inter-application traffic feasibility in a similar manner to that attempted for intra-application traffic - by explicitly checking the feasibility of every message. The graphical representation of this problem is given in the left part of Figure 6. In cases where the applications have x and y dispatchers respectively, the number of different combinations is equal to $x \times y$ (in this example $8 \times 8 = 64$). This holds for the communication between the two applications executed in isolation. However, when the interferences caused by the communication of the other applications are considered, similarly to the intra-application traffic, the analysis would be either too pessimistic or intractable, depending on the selected strategy for treating potentially interfering applications.

In order to solve this problem we introduce a *proxy dispatcher*. Its task is to perform the communication with the same of the other application as well as to transfer the data to/from its own master. The inter-application message is decomposed into 3 messages: 1) from the master sender to its proxy, 2) between the proxies of the interacting applications and 3) from the proxy to the master receiver. Note, that the communication between the proxies and their masters is an intra-application traffic which was covered in Section 5.1. The proxies are fixed, so two applications always communicate the data over the same dispatchers, thus the route is deterministic and master-independent. The proxies are selected during the mapping process in a strategic way, such that the distance between the applications is minimised (see right part of Figure 6). Additionally, proxy communication complies with XY routing policy, hence no re-routings occur. This approach of a three-step inter-application communication

allows to decompose an inter-application message into 3 different messages and study them individually, but at the expense of a re-routing that has to occur in the proxies and which may reduce the performance. Additionally, the proxies are dedicated, per-connection roles, so an application can have multiple proxies, each responsible for performing the communication with one or more applications.

5.3. Performing the feasibility analysis

The previous subsections showed that performing an exact feasibility analysis by employing one of the existing approaches to the assumed model causes either intractability or pessimism, whether analysing inter- and intra-application traffic. Nevertheless, let us again consider the example given in Figure 6. Applying the exact analysis without the constraints introduced in this paper, would require to consider $8 \times 7 = 56$ agreement protocol messages per application (explained in Subsection 5.1) and $8 \times 8 = 64$ inter-application messages (explained in Subsection 5.2). The total number of messages in this example is: $2 \times 56 + 64 = 176$, inferring that proving the feasibility for any practical scenario will lead towards either intractability or pessimism build up, which coincides with the conclusions of the previous subsections.

On the other hand, the proposed approach transforms the intra-application traffic into a linear combination of supermessages. Additionally, an inter-application message is decomposed into one proxy-to-proxy message and two intra-application messages, which are in turn also transferred into supermessages. Consequently, all inter- and intra-application traffic can be described by proxy-to-proxy messages m^p and supermessages $\widehat{m}$. Note, that both these types of messages comply with XY routing policy ($R(m^p) = R(\widehat{m}) = 0, \forall m^p \in \mathcal{M}, \forall \widehat{m} \in \mathcal{M}$), so it is possible to compute $WCTT$ of these messages by obtaining the terms of Equation 1. By applying this approach to example in Figure 6, the applications a_i and a_j have 4 supermessages (see Subsection 5.1) and have one proxy-to-proxy message (see Subsection 5.2) - in total $2 \times 4 + 1 = 9$ messages exist. Compared to the previous approach, a significant decrease in the number of messages is evident, which makes it possible to obtain the worst-case delays on messages for scenarios which reflect practical examples with hundreds of applications.

Once $WCTT$ values are obtained for supermessages and inter-proxy messages, said values are used to test the feasibility of inter- and intra-application traffic via reverse transformations. Every simple message m^s_{ii} of the application a_i shares a part of the path with only one supermessage $\widehat{m_{ii}}$ and thus covers the distance which is not longer than that of supermessage. Consequently, the WCTT of that supermessage can be interpreted as an upper-bound on the WCTT of a simple message - $\overline{WCTT(m^s_{ii})}$ (Equation 2). Thus, a simple message is feasible if its upper-bound is feasible (Equation 3).

$$WCTT(m^s_{ii}) \leq WCTT(\widehat{m_{ii}}) = \overline{WCTT(m^s_{ii})} \tag{2}$$

$$\overline{WCTT(m^s_{ii})} \leq t_{ii} \Rightarrow WCTT(m^s_{ii}) \leq t_{ii} \tag{3}$$

Furthermore, every complex message m^c_{ii} shares a part of the path with two supermessages - $\widehat{m_{ii}}^{\bullet}$ and $\widehat{m_{ii}}^{\circ}$. The distance it covers cannot be longer than the sum of the distances of its supermessages. Also, it suffers one re-routing on the core where the two supermessages intersect. The summation of the worst-case delays of the suppermessages augmented by the re-routing overhead is always greater or equal to the delay of the complex message and can be interpreted as its upper-bound - $\overline{WCTT(m^c_{ii})}$ (Equation 4). Thus, a complex message is feasible if its upper-bound is feasible (Equation 5).

$$WCTT(m^c_{ii}) \leq WCTT(\widehat{m_{ii}}^{\bullet}) + WCTT(\widehat{m_{ii}}^{\circ}) + R(m^c_{ii}) = \overline{WCTT(m^c_{ii})} \tag{4}$$

$$\overline{WCTT(m^c_{ii})} \leq t_{ii} \Rightarrow WCTT(m^c_{ii}) \leq t_{ii} \tag{5}$$

In order to obtain the delay a complex message suffers due to re-routing - $R(m^c_{ii})$, we firstly define a re-routing core c_R (Equation 6), such that it exists in the set of cores

$\mathcal{C}$ and contains the dispatcher d_i of the application a_i. Additionally, the message of interest and both the supermessages traverse that core.

$$c_R = c|(c \in \mathcal{C}) \wedge (c \ni d_i) \wedge (c \in path(\widehat{m_{ii}}^{\bullet}) \cap path(\widehat{m_{ii}}^{\circ}) \cap path(m_{ii}^c)) \quad (6)$$

We assume that the re-routing is implemented as a high priority OS operation serviced in a fifo, interrupt-handling manner, with a constant delay of d_R per single re-routing. Note, that a re-routing can occur only on dispatchers which are proxies and/or are positioned in the corners of their application shapes. The delay a message suffers on the core c_R is denoted by $r(c_R)$ in Equation 7. Since a complex message suffers only one re-routing, its total re-routing delay $R(m_{ii}^c)$ is equal to only one term (Equation 7). The worst-case re-routing delay is calculated by assuming a critical instant - the requests for all possible re-routings on the core of interest appeared just before the one belonging to the message under analysis, which is serviced last.

$$R(m_{ii}^c) \; = \; r(c_R) = \sum_{\forall d \in c_R | d \in \{corner, proxy\}} d_R \quad (7)$$

Finally, let us derive a feasibility constraint for the inter-application message m_{ij} between the applications a_i and a_j. Similar logic to that applied for complex messages can be used here; m_{ij} can be decomposed in inter-proxy communication m_{ij}^p and the intra-application traffic between the masters and their proxies on both sender and receiver side - m_{ii} and m_{jj}. To cover the worst-case, both these master-proxy messages are considered as complex ($m_{ii} = m_{ii}^c$, $m_{jj} = m_{jj}^c$). Note, that the analysis provides the worst-case delay of the proxy message $WCTT(m_{ij}^p)$, while Equation 4 gives an upper-bound on the latency of the complex messages - $\overline{WCTT(m^c)}$. Now, an upper-bound on the worst-case latency of the inter-application message can be formulated as the summation of the upper-bounds on the messages into which it is decomposed ($m_{ii}^c, m_{ij}^p, m_{jj}^c$) augmented by the re-routing delays occurring on the cores of both the proxies (Equation 8). The feasibility constraint is given by Equation 9.

$$WCTT(m_{ij}) \leq \overline{WCTT(m_{ii}^c)} + WCTT(m_{ij}^p) + \overline{WCTT(m_{jj}^c)} + R(m_{ij}^p) = \overline{WCTT(m_{ij})} \quad (8)$$

$$\overline{WCTT(m_{ij})} \leq t_{ij} \Rightarrow WCTT(m_{ij}) \leq t_{ij} \quad (9)$$

The only term still not defined is the re-routing overhead of the inter-proxy message - $R(m_{ij}^p)$. Even though the message m_{ij}^p itself does not experience re-routing, the same happens inside both the proxies. The re-routing delay is described by Equation 10, where c_{PS} and c_{PR} stand for the cores of proxy sender and proxy receiver, respectively.

$$R(m_{ij}^p) = r(c_{PS}) + r(c_{PR}) = \sum_{\forall d \in c_{PS} | d \in \{corner, proxy\}} d_R + \sum_{\forall d \in c_{PR} | d \in \{corner, proxy\}} d_R \quad (10)$$

In the worst-case, an inter-application message might suffer 4 re-routings. The first may happen between the master and the proxy of a_i, considered in the $\overline{WCTT(m_{ii}^c)}$ of Equation 8. Two may occur within the proxies, which account for $R(m_{ij}^p)$. The last might happen between the proxy and the master of a_j, considered in the $\overline{WCTT(m_{jj}^c)}$.

Due to re-routing and forced paths, this approach sacrifices actual performance in order to get deterministic routes. Additionally, it induces pessimism via message transformations, so as to reduce the total number of different messages and avoid the combinatorial complexity. Nevertheless, we base our reasoning on the fact that these overheads can be considered as justifiable and we employ this approach in order to derive a tight and computationally tractable analysis.

5.4. Allowed application geometries

One of the prerequisites of our approach is to position the dispatchers in such a way so as not to violate the placement constraints introduced by Definition 3. The corners

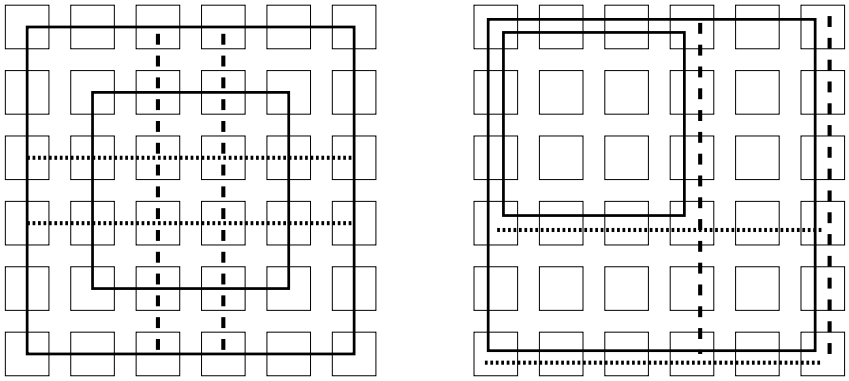


Fig. 7. Allowed application geometries

of the shape cannot be left unoccupied due to re-routing purposes. Figure 7 illustrates some possible application placements (one forming 6×6, one 4×4, two 1×6 and two 6×1). Different line styles represent different applications. As already stated, if one of the dimensions is equal to "1" the application assumes a line-like shape. For the sake of clarity, we omitted to pinpoint the exact dispatchers and decided to graphically represent the applications only with the shapes.

The example in Figure 7 emphasizes the importance of the placement process. Left and right part present two different placements of the same application-set. In the left part the applications are placed in such a way that there is no interference between any two, while in the right part every application interferes with at least one. The former placement is better, since fewer or none potential interferences lead towards a less pessimistic analysis and better performance. We employ this reasoning during mapping process, and apart from deriving a feasible solution with maximised migration abilities, one of the objectives is to minimise the potential traffic contentions.

5.5. The quality of the placement

The multi-objective nature of the proposed approach is reflected in 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 load balancing, i.e. to migrate. In this subsection we address this secondary objective. 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 it is claiming and **the distribution of the dispatchers** on that shape. The approach we apply when evaluating the placement of the application can be summarised with the following two statements:

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

2) Assuming a fixed shape, the more equalised 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, thermal and energy management and are resistant 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.

Although it looks that this objective contradicts feasibility requirements, it is not entirely true. If perceived as an optimisation problem, migrative abilities of applications are the objective function which has to be maximised, and assuring the feasibility is a constraint. As a result, the solution is found such that the dimensions of the shape it is claiming are as big as possible, the dispatcher distances are as equal as possible, and

all the messages are feasible. In order to qualitatively evaluate different mappings, a proper metric has to be established. We say that a quality of a placement of the application a_i, noted down by q_i (Equation 11), is equal to the product of the distances between its consecutive dispatchers, multiplied by its migration coefficient $m(a_i)$.

$$q_i = m(a_i) \prod_{\forall d_j \in a_i} dist(d_j, d_{i+j}) \tag{11}$$

m is a parameter specified by the system designer and reflects the importance of the distribution of a particular application. The more the system would benefit from one application's wide placement, the greater this parameter is. It has no significance in absolute terms, it has only relative meaning when all the applications are considered. The purpose of this parameter is further explained in the following subsections.

The product between the dispatcher distances is used because it is a computationally cheap operation. 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, it is directly proportional to the shape size, i.e. greater shape size means greater distances between the dispatchers and hence greater product of its distances. Finally, when assuming that a shape has been decided, the product of the dispatcher distances reaches the maximum when all the distances are as equal as possible. For details see Theorem 5.1.

Note that the distances between the dispatchers are natural numbers, which is a subset of real numbers. In cases when $\frac{C}{n} = p \in \mathbb{N}$, where C stands for the circumference of the application shape and n stands for the number of the dispatchers, the results of Theorem 5.1 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). In cases when $\frac{C}{n} = p \notin \mathbb{N}$ the maxima are different. The maximum on a discontinuous natural-number domain is the point geometrically the closest to the continuous maximum, due to the fact that the quality function represented by Equation 11 has a unique maximum on a continuous real-number domain, as proven by Theorem 5.1. The existence of a single maximum suggests that the function is monotonically increasing from the boundary to the extremum, with respect to each variable, when treating the other variables as constants. In fact, there are multiple discontinuous solutions with the same distance from the continuous maximum, and hence multiple maxima. These are the solutions where the distances between all the dispatchers are either $\lceil \frac{C}{n} \rceil$ or $\lfloor \frac{C}{n} \rfloor$, which coincides with our reasoning (see the right example in Figure 11 which is in fact the maximum for given $C = 16, n = 6$ and where the distances between the dispatchers are either 2 or 3 because $\frac{C}{n} = \frac{16}{6} = 2.67$).

THEOREM 5.1. *Let $X = \{x_1, x_2, ..., x_n\} \in \mathbb{R}$ be a set of real number variables, such that the following holds:*

$$\sum_{x_i = x_1}^{x_n} x_i = C \tag{12}$$

$$x_i \geq 0, \forall x_i \in X \tag{13}$$

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

PROOF. This is a constrained optimisation problem, with 1 constraint expressed by the equality (Equation 12), and n constraints expressed by the inequalities (Inequality 13) . If we exclude the inequalities from consideration, the extreme values of the function $f(X)$, subject to equality constraint, can be found by the *Lagrange Multipliers Method*.

$$f(X) = \prod_{x_i = x_1}^{x_n} x_i, \qquad g(X) = \sum_{x_i = x_1}^{x_n} x_i = C \qquad \Rightarrow \qquad \mathcal{L} = \prod_{x_i = x_1}^{x_n} x_i + \lambda \left(C - \sum_{x_i = x_1}^{x_n} x_i \right) \tag{14}$$

A new variable λ is called Lagrange multiplier. According to the first derivative test, the necessary condition for 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 Equations 15-17)

$$\frac{\partial \mathcal{L}}{\partial x_1} = \frac{\partial f(X)}{\partial x_1} - \lambda \frac{\partial g(X)}{\partial x_1} = 0 \quad \Rightarrow \quad \prod_{x_i = x_2}^{x_n} x_i = \lambda \tag{15}$$

$$\vdots$$

$$\frac{\partial \mathcal{L}}{\partial x_n} = \frac{\partial f(X)}{\partial x_n} - \lambda \frac{\partial g(X)}{\partial x_n} = 0 \quad \Rightarrow \quad \prod_{x_i = x_1}^{x_{n-1}} x_i = \lambda \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \quad \Rightarrow \quad \sum_{x_i = x_1}^{x_n} x_i = C \tag{17}$$

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

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

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

Additionally, due to inequality constraints ($x_i \geq 0, \forall x_i \in X$), it is necessary to check the boundaries as well. The boundaries are represented with the solutions where only one of the variables is equal to 0, (i.e. $\exists x_i \in X \wedge \nexists x_j \in X | x_i = x_j = 0 \wedge i \neq j$. Those are called boundary points and there exists no test to prove their properties, they have to be checked explicitly. In this concrete case it is easy; it is obvious that those points represent 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 *Bordered Hessian*. It consists of finding the first partial derivatives of $g(X)$ with respect to X, then finding the second partial derivatives of $f(X)$ also with respect to X and finally putting them into the matrix called Bordered Hessian. The general form of Bordered Hessian is represented by Figure 8.

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

Fig. 8. General form of Bordered Hessian for n variables and 1 constraint

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

Fig. 9. Bordered Hessian for $\lambda = 0$

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

Fig. 10. Bordered Hessian for $\lambda \neq 0$

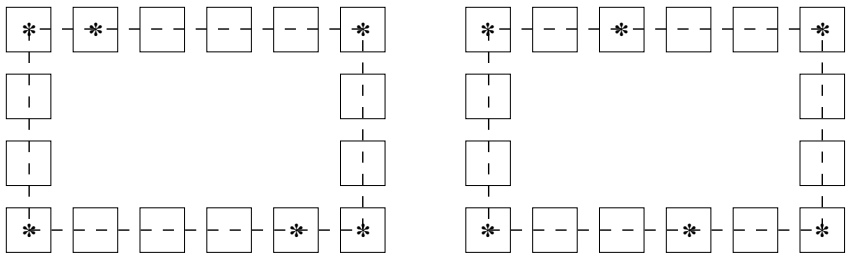


Fig. 11. Comparison of mappings

1) $\underline{\lambda = 0}$: Figure 9 presents Bordered Hessian for the case where $\lambda = 0$. The variable z_{ij} stands for the second partial derivative with respect to variables x_i and x_j and is described by Equation 18. 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. Sufficient condition for local maximum is that Bordered Hessian is negative definite, i.e. the determinants of its principal minors alternatively change sign (Equation 19). In both cases, there exists some $|H_i| = 0$, thus making this test inconclusive. In such cases, each of the points should be investigated individually. However, in this example it is obvious that both these cases represent the minima on the domain, since $f(X) = 0$ for all of them.

$$z_{ij} = z_{ji} = \prod_{\substack{k=1 \\ k \neq i \\ k \neq j}}^{n} x_k \tag{18}$$

$$|H_1| < 0, |H_2| > 0, ... \qquad \Rightarrow \qquad sign(|H_i|) = (-1)^i, \forall i \in n \tag{19}$$

2) $\underline{\lambda \neq 0}$: Bordered Hessian for this case is represented by Figure 10. For z_{ij} also holds Equation 18, 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 (Equation 20).

$$z_{ij} = z_{ji} = z = \left(\tfrac{C}{n}\right)^{(n-2)}, \forall i \forall j \in n | i \neq j \tag{20}$$

When computed, the determinants of the principal minors are $|H_1| = 2z, |H_2| = -3z^2, |H_3| = 4z^3, ..., |H_n| = (-1)^n n z^n$. 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 hence 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 $x_1 = x_2 = ... = x_n = \frac{C}{n}$ and the value is $max(f(X)) = \left(\frac{C}{n}\right)^n$. □

Note, that a zero-distance between the dispatchers presents a special case where the same are located on a common core. In the assumed model, such a placement is meaningless and the results of Equation 11, which evaluates the quality of the placement, also coincide with this reasoning, hence returning $q_i = 0$ for every such placement.

For a given shape without any constraints, choosing and applying an optimal dispatcher distribution would be trivial: if possible - make all the inter-dispatcher distances equal, otherwise make all the distances either $\lceil \frac{C}{n} \rceil$ or $\lfloor \frac{C}{n} \rfloor$. However, the corners of the shape pose implicit constraints regarding the inter-dispatcher distances and, in some cases, prevent optimal solutions. Also, not all the cores on the path of the application shape might be available due to schedulability or malfunctioning reasons, thus further preventing optimal solutions. The purpose of the aforementioned evaluation metric is to provide a qualitative comparison and selection between different possible suboptimal dispatcher distributions in cases when optimal ones are not possible.

The example in Figure 11 illustrates how different mappings are evaluated. Two possibilities are presented, in both of them the application claims the same shape,

but the placement of inner dispatchers differs. All dispatchers are represented with a star sign. A placement is considered as good if dispatcher distances are equalised as much as possible. From that perspective the right placement is better. By obtaining the values of Equation 11, 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.

5.6. Mapping procedure

The importance of application mapping in many-cores is reflected in the fact that many works already addressed this problem (see Section 2). The state-of-the-art approaches assume a static pre-assignment of the workload to the cores and investigated their placement on the chip. This problem is recognised as a quadratic assignment problem [Hu and Marculescu 2003a], which is in the most of the practical cases not solvable within reasonable time. Our work additionally encompasses migrations (via dispatchers), which additionally increases the complexity. We believe that obtaining an optimal result through exhaustive search for practical cases with hundreds of applications, each represented by several to dozen of dispatchers, is prohibitively expensive and requires an unjustifiably ample amount of time.

In this paper we present a heuristic-based application mapping method which, with tolerable complexity and negligible amount of pessimism, provides a placement plan for an application-set such that all real-time traffic fulfils posed constraints and at the same time allows workload migrations. In order to be considered, a solution has to be feasible, i.e. by the analysis presented in Section 5.3 every message has to be proven feasible. Feasible solution is then evaluated by its quality Q, presented by Equation 21, where q_i stands for the placement quality of an individual application (Equation 11). Our aim is to provide a feasible solution, such that its quality Q is maximised.

$$Q = \sum_{\forall a_i \in \mathcal{A}} q_i \tag{21}$$

The proposed method is a three-step process, with three application mapping stages: *Initial Phase* ($\mathcal{IP}$), *Feasibility Phase* ($\mathcal{FP}$) and *Optimisation Phase* ($\mathcal{OP}$).

5.6.1. Initial Phase ($\mathcal{IP}$). The applications are separated into two groups, $\mathcal{H}$ if the priority is higher or equal to P, and $\mathcal{L}$ if the priority is lower than P. During $\mathcal{IP}$, only the applications from $\mathcal{H}$ are mapped. The parameter P is arbitrarily chosen by the system designer. Any change in dispatcher placements and/or assumed shape of one application requires a new feasibility check not only for all the messages of that application, but also for all the lower priority traffic. The recalculation triggered by a high priority application might be computationally very expensive, hence as a means to control the complexity we have introduced a parameter P. All applications that are above the threshold, i.e. ones belonging to $\mathcal{H}$, will assume their final placement during $\mathcal{IP}$ and will not be subject to any changes during later mapping stages.

The applications are sorted non-increasingly by their priorities. By performing the assignment in this order, a currently assigning application doesn't have an impact on the feasibility of previously mapped applications. Therefore, in order to maintain the feasibility of the mapping, it is sufficient to check the feasibility only of the messages belonging to the currently assigning application. This reduces the complexity of $\mathcal{IP}$ from $O({n_h}^2)$ to $O(n_h)$, where n_h stands for the amount of applications in $\mathcal{H}$.

When mapping an application, we differentiate between several types of placements. We define *narrow placement* to be a shape of an application with a minimum surface, i.e. a placement where all dispatchers of an application occupy consecutive cores. For instance, narrow placements for a 4-dispatcher application are: $1 \times 4, 4 \times 1, 2 \times 2$. Conversely, in a *wide placement* an application assumes a shape with the maximum sur-

ALGORITHM 1: $\mathcal{IP}$(A, P, G) The first mapping phase $\mathcal{IP}$ (Initial Phase)

```
   input : A, P, G
 1 foreach (a in A : p(a) ≥ P) do
 2 |   add(a, H); // Select all the higher priority applications
 3 end
 4 H.sort(P↓); // Sort by non-increasing priority
 5 foreach (a in H) do
 6 |   a.calculate(S^min, S^max);
 7 |   S = a.S^min + G × (a.S^max - a.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 non-increasing surface
12 |   while (a.placed() != true && Allowed != empty) do
13 |   |   shape = Allowed.get(); // Get the widest existing shape
14 |   |   if (try(a, shape) == true) then
15 |   |   |   a.place(shape); a.map(dispatchers);
16 |   |   |   a.proxies(dispatchers); // Select proxies
17 |   |   else
18 |   |   |   Allowed.remove(shape);
19 |   |   end
20 |   end
21 |   if (a.placed == false) then Mapping.Failed(a); // Declare mapping failure of IP
22 end
23 Mapping.Success(); // Declare mapping success of IP
```

face - in most of cases the boundaries of the chip. We refer to the surfaces of the narrow and the wide placement as to S^{min} and S^{max}, respectively.

Since the applications from $\mathcal{H}$ undergo a final placement during $\mathcal{IP}$, it is essential to dedicate a proper shape to each of these applications. If during $\mathcal{IP}$ most of the applications assume wide placement, that will create significant high priority traffic on the network, which might cause lower priority applications to be unable to reach the feasibility of their messages in subsequent mapping phases. Oppositely, forcing the applications from $\mathcal{H}$ to claim mostly narrow placements might unnecessarily preserve the network underutilised as there might be very few applications left for mapping in subsequent phases. Therefore, it is important to wisely limit the network consumption by the high priority applications, and hence leave enough resources to the workload that has to be considered later. For that purpose, we introduce the parameter G. It controls the "greediness" of applications from $\mathcal{H}$. G presents an upper-bound on the allowed application shapes. For instance, if $G = 0$ only the shapes which fulfil $S = S^{min}$ are allowed. Similarly, if $G = 1$, shapes which fulfil $S \leq S^{max}$ are allowed, which includes all possible shapes. If $0 < G < 1$, allowed shapes are calculated by Equation 22. 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 consideration.

$$S \leq S^{min} + G(S^{max} - S^{min}) \tag{22}$$

Algorithm 1 illustrates how the $\mathcal{IP}$ stage is performed. Firstly, high priority applications are selected and sorted by non-increasing priority (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 or equal to the calculated threshold S are selected and sorted non-increasingly by the shape surface (lines 8-11). The placement is attempted with the selected application and the selected shape (line 14) on the entire grid. Note, that this process involves 1) the placement of a shape at the particular location on the grid, 2) the generation of su-

permessages, 3) the assignment of individual proxy roles, 4) the assignment of proxy roles to other applications communicating with the one under analysis, 5) the generation of inter-application messages assuming elected proxies, 6) feasibility checks for all application's inter- and intra-application messages.

If the application can be placed with the selected shape on multiple places of the grid, the smallest sum of the worst-case latencies of the application messages is used as a tie breaker criteria, where the sum includes the latencies of both inter- and intra-application messages. In this way the algorithm searches for a position on the grid where an application suffers the least interference from the other traffic. As mentioned in Section 5.4, this is the secondary objective of the mapping process and it causes interacting applications to be placed close to each other. A special case occurs when proxies of two interacting applications share the same core.

Furthermore, if there are several positions on the grid where the application can be placed and for which the sum of the message latencies is minimised, the approach selects the one for which the sum of the dispatcher distances from the center of the grid is minimised. By making sure that an application is placed as close to the center of the grid as possible, any lower priority application which is yet to be placed, and which performs the communication with said application, will have the possibility to evaluate more placement options so as to minimise the communication penalty, while that would not be the case if the higher priority application is placed on the border of the grid. Note, that due to clarity purposes these two shape selection criteria are not explicitly mentioned in Algorithm 1, but we assume the aforementioned logic is encapsulated within the function located in the line 14.

Once the shape and the grid position are selected, the corner dispatchers are placed (line 15). Subsequently, the rest of the dispatchers (if any) are positioned on the edges of the shape, such that the quality of the application q_i is maximised (line 15). Once proxy roles are assigned (line 16), the application is considered as mapped.

If an application cannot be placed on the grid with the current shape, the same is excluded (line 18), and the placement is attempted with the next shape from the collection of allowed shapes. The process repeats until proper shape and position are found such that all feasibility constraints hold. If an application cannot claim the feasibility with any of the shapes, the mapping process declares failure (line 21). Conversely, when all the applications from $\mathcal{H}$ are mapped, $\mathcal{IP}$ declares a success (line 23).

5.6.2. Feasibility Phase ($\mathcal{FP}$). This phase only performs the mapping of lower priority applications, with the primary objective to derive a feasible solution with the entire application-set mapped. Therefore, during $\mathcal{FP}$ every application may assume only a narrow placement. $\mathcal{FP}$ is described by Algorithm 2. Firstly, all lower priority applications are grouped in $\mathcal{L}$ and sorted by priority, non-increasingly (lines 1-4). The applications are treated sequentially; a surface of a narrow placement S^{min} is found (line 6), and subsequently it is used to find all possible shapes which correspond to the narrow placement of the application (lines 7-9). The mapping is attempted with the first shape from the list, without any preference, as all selected shapes are considered as a narrow placement (lines 11, 12). The same logic as used in $\mathcal{IP}$ applies here: if multiple grid positions are possible for the same shape, one with the minimised sum of the message latencies is selected, and if multiple grid positions also have the same sum of the latencies, the one closer to the center of the grid has the precedence. These steps are assumed to exist within the line 12 and are omitted for better clarity of the algorithm. Once the shape is claimed, the dispatchers are mapped on every core of the shape, since it is a narrow placement (line 13). Finally, proxy roles are assigned (line 14).

If an application cannot be mapped with a selected shape, the same is removed and the mapping is attempted with the next from the list of allowed shapes (line 16). The

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

```
   input : A, P
1  foreach (a in A : p(a) < P) do
2  |  add(a, L); // Select all the lower priority applications
3  end
4  L.sort(P↓); // Sort by non-increasing priority
5  foreach (a in L) do
6  |  a.calculate(S^min);
7  |  foreach (shape : shape.S() == S^min) do
8  |  |  add(shape, Allowed); // Select all the narrow-placement shapes
9  |  end
10 |  while (a.placed() != true && Allowed != empty) do
11 |  |  shape = Allowed.get(); // Get the first of the allowed shapes
12 |  |  if (try(a, shape) == true) then
13 |  |  |  a.place(shape); a.map(dispatchers);
14 |  |  |  a.proxies(dispatchers); // Select proxies
15 |  |  else
16 |  |  |  Allowed.remove(shape);
17 |  |  end
18 |  end
19 |  if (a.placed == false) then Mapping.Failed(a); // Declare mapping failure of FP
20 end
21 Mapping.Success(); // Declare mapping success of FP
```

process repeats until a shape is found such that all the messages are feasible. If none of shapes satisfies posed timing constraints, the mapping process declares failure (line 19). Oppositely, once all the applications are mapped, $\mathcal{FP}$ declares a success (line 21). The computational complexity of $\mathcal{FP}$ is $O(n_l)$, where n_l denotes the amount of applications in $\mathcal{L}$. The output of $\mathcal{FP}$ is the first feasible solution of an entire application-set.

5.6.3. Optimisation Phase ($\mathcal{OP}$). The objective of this phase is to improve on the solution received from $\mathcal{FP}$. Said process is performed by attempting to extend narrow shapes that lower priority applications claimed during $\mathcal{FP}$. It is performed until no further extensions are possible. That marks the end of the mapping process and reached solution presents the output of the entire mapping process. $\mathcal{OP}$ is depicted by Algorithm 3.

Similarly, applications from $\mathcal{L}$ are selected and sorted non-increasingly (lines 1-4), but during $\mathcal{OP}$ by the parameter m. As said, it represents the significance of the wider placement of an application, i.e. the more the system would benefit from a wider placement of an application, the greater its parameter m is. The positive side is that the applications which distribution matters the most 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, in order an expansion of one application to be accepted, it is necessary to perform a re-check of the feasibility of all the applications with a priority lower than the expanding one. Therefore, the computational complexity of $\mathcal{OP}$ is $O({n_l}^2)$, where n_l represents the amount of applications in $\mathcal{L}$.

The applications are sorted and treated sequentially; the expanded shape is found and the placement attempted (lines 8, 9). The process consists of an attempt to push the corner dispatchers of the application in all possible directions. If an attempted shape claims feasibility (line 9), for every application with the priority lower than the stretching one a feasibility re-check is performed (lines 10-13). If any of re-evaluated applications is unable to declare feasibility, an attempted shape is removed from the consideration. The application is expanded until any further stretches will cause infeasibility of either itself or any of lower priority applications. Every expansion is followed by the rearrangement of the dispatchers of the expanding applications, so as

ALGORITHM 3: $\mathcal{OP}(\mathcal{A}, \mathrm{P})$ The third mapping phase $\mathcal{OP}$ (Optimisation Phase)

```
   input : A, P
1  foreach (a in A : p(a) < P) do
2  |  add(a, L); // Select all the lower priority applications
3  end
4  L.sort(m↓); // Sort by non-increasing distribution priority
5  foreach (a in L) do
6  |  feasible = true;
7  |  while (feasible == true) do
8  |  |  new_shape = expand(a.shape); // Find expanded shape to attempt
9  |  |  if try(a, new_shape == true) then
10 |  |  |  foreach (a' in L : a'.P() < a.P()) // Feasibility re-check for lower priority apps
11 |  |  |  do
12 |  |  |  |  if (try(a', a'.shape) == false) then  feasible = false; break;
13 |  |  |  end
14 |  |  |  if (feasible == true) then
15 |  |  |  |  a.shape = new_shape; // Claim expanded shape
16 |  |  |  |  a.rearrange(dispatchers); // Equalise inter-dispatcher distances
17 |  |  |  |  a.proxies(dispatchers); // Reassign proxy roles
18 |  |  |  end
19 |  |  else
20 |  |  |  feasible = false;
21 |  |  end
22 |  end
23 end
24 Mapping.Success(); // Declare mapping success of OP and entire process
```

to equalise the inter-dispatcher distances as much as possible (line 16). Then, proxy roles are reassigned for the application under analysis and all applications interacting with it (line 17). Once this process is performed for all applications from $\mathcal{L}$, the entire mapping process concludes (line 24) and returns the current solution as a final output.

5.6.4. A word on schedulability. The schedulability of a solution on particular cores is out of scope of this paper. Our assumption is that there exists a schedulability test which verifies that for every generated solution the following holds: at any time instance there will be enough resources to allow all real-time applications to fulfil their execution requirements and that the logic of agreement protocols will assure the schedulability, i.e. at any time instance at least one of dispatchers will be able to accommodate the execution of the application on its core and the agreement protocol will recognise it. This aspect is the main objective of the currently ongoing work (see Section 7).

Apart from these implicit assumptions, additional constraints addressing schedulability issues can be explicitly placed, e.g. 1) the maximum number of dispatchers per core, 2) the per-core maximum allowed cumulative utilisation, where dispatchers carry a fraction of a belonging application's utilisation. In the evaluation section we employ these two constraints by allowing no more than 10 dispatchers per core and by limiting the cumulative on-core utilisation to the capacity of individual cores.

6. EVALUATIONS

This section presents several case studies addressing different aspects of the proposed approach.

6.1. Case study 1: Application shapes

In many situations, several shapes have similar or identical characteristics, especially during $\mathcal{FP}$ when narrow placements are considered. In order to asses the differences between shapes and reason about their applicability, we analyse the characteristics

which we consider relevant, namely a traversal distance of the average and the longest message. For easier calculation, in all cases narrow placements are assumed.

Equation 23 calculates the average traversal distance of a message, assuming a n-dispatcher application with 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 Equation 23). 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). In similar way, the value is calculated for a rectangular shape with the forced routes (Equation 24), while obtaining the maximum message distance for both shapes is trivial (Equation 25 and Equation 26).

$$\text{Line-AVG} = \frac{\sum_{i=1}^{n}\left(\sum_{k=1}^{i-1} k + \sum_{j=1}^{n-i} j\right)}{n(n-1)} = \frac{n+1}{3} \quad (23)$$

$$\text{Rect-AVG} = \frac{\sum_{i=1}^{\lceil \frac{n-1}{2} \rceil} i + \sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} i}{n-1} = \frac{n^2}{4(n-1)} \quad (24)$$

$$\text{Line-MAX} = n-1 \quad (25)$$

$$\text{Rect-MAX} = \left\lceil \frac{n-1}{2} \right\rceil \quad (26)$$

These values are plotted in Figure 12 to illustrate how the amount of dispatchers influences the distance traversed by the average and the longest message of these two possible shapes. The rectangular shape shows better performance than the line one, for both the average and the longest message, but at the expense of a potential re-routing, which is not needed for the latter. The decision regarding shape precedence in concrete cases can be made by the system designer, or left to mapping process to decide.

Additionally, in order to estimate the penalty of the forced re-routing, traversal distances of the average and the longest message are computed for the rectangular shape with free point-to-point communication between dispatchers (Equation 27 and Equation 28 respectively) and consequently plotted in Figure 12. Equation 27 is derived by using an intermediate results of Equation 23, where n is substituted by $\frac{n}{2}$, because a rectangular shape has 2 rows of dispatchers.

$$\text{Free-AVG} = \frac{2\sum_{i=1}^{\frac{n}{2}}\left(\sum_{k=1}^{i-1} k + \sum_{j=1}^{\frac{n}{2}-i} j\right) + \frac{n^2}{4}}{\frac{n}{2}(n-1)} = \frac{n^2+3n-4}{6(n-1)} \quad (27)$$

$$\text{Free-MAX} = \left\lceil \frac{n-1}{2} \right\rceil \quad (28)$$

It is clear from Figure 12 that the removal of constraints improves the performance for the average message, but keeps the same longest message and, as recognised in the previous sections, significantly complicates the analysis. Unlike Figure 12, which visualises the overhead of the forced paths expressed in absolute values (Equation 29), Figure 13 depicts the same overhead expressed relatively (Equation 30). It shows how much the traversal distance of the average message under forced paths is longer than the same without said restrictions, expressed in the percentages of the later. In practical terms it means that for a 10-dispatcher application, the traversal distance of the average messages is 19% longer.

6.2. Case Study 2: Scalability

The objective of this case study is to test the scalability potential of the proposed approach. We performed the mapping process, assuming a workload synthetically generated by using the parameters from the following table:

Application utilisation	**(0 - 1]***	Number of dispatchers (migrative application)	**[2 - 10]***
Application priority - $p(a)$	**[0 - 20]***	Application distribution importance - $m(a)$	**[0 - 20]***
Agreement message size	**64 bytes**	Inter-application message size	**[0 - 1024]* bytes**
Migrative applications	**50%**	Probability of inter-application communication	**5%**
Router switch + transfer time	**1 + 3 cycles**	2D mesh width	**16 bytes**

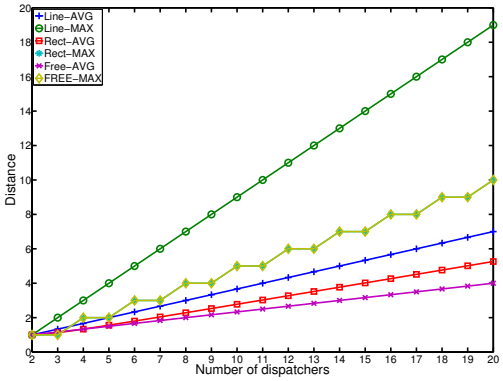


Fig. 12. Comparison of different shapes

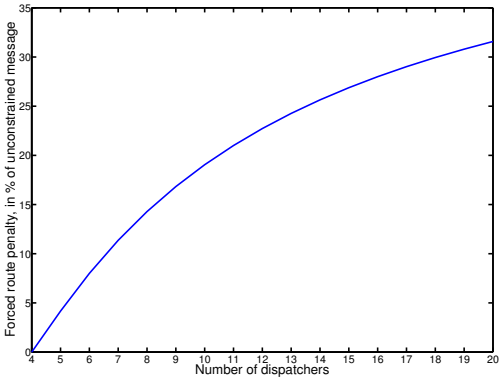


Fig. 13. Estimation of penalty due to forced routes

$$O_A = \text{Rect-AVG} - \text{Free-AVG} = \frac{n^2 - 6n + 8}{12(n-1)} \quad (29)$$

$$O_R = \frac{\text{Rect-AVG} - \text{Free-AVG}}{\text{Free-AVG}} = \frac{n^2 - 6n + 8}{2n^2 + 6n - 8} \quad (30)$$

An asterisk sign denotes a uniformly distributed random value. Intra-application message latencies were generated by multiplying their respective *basic latencies* (latencies assuming a narrow placement in isolation) by an additional factor. We made this factor oppositely proportional to priorities, resulting with high priority applications having time constraints close to their basic latencies, while lower priority applications have less tight constraints, or don't have them at all (best-effort workload). We believe that this assumption reasonably represents realistic scenarios. Each inter-application message assumes a timing constraint which is equal to the maximum of all the constraints posed on the intra-application messages belonging to the lower priority application. All these parameters hold for this and subsequent case studies.

The scalability potential of the proposed approach was tested by varying the number of applications in the range [2 - 200]. For each given value of the application-set size we generated and mapped 1000 application-sets on a 8×8 grid. The other mapping-related parameters were kept constant, namely $G = 0.5$, $P = 10$. We performed the timing analysis by capturing the duration of the mapping process of each run.

Figure 14 presents the results. The horizontal axis stands for the application-set size. For clarity purposes we present a reciprocal cumulative graph, where the vertical axis stands for the quantity of the runs that still did not complete the mapping process within a given time interval (depth axis). The left part of Figure 14 shows that almost all the runs for small application-sets complete very fast, while 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 ($0 - 3$ seconds), almost all the smaller sets finished, while very few of the largest ones report mapping completion.

The second period ($3 - 40$ seconds) is presented in the right part of Figure 14. The 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. The observation covers a larger time period, hence 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 - Figure 15. 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 25^{th} and 75^{th} percentile, which is

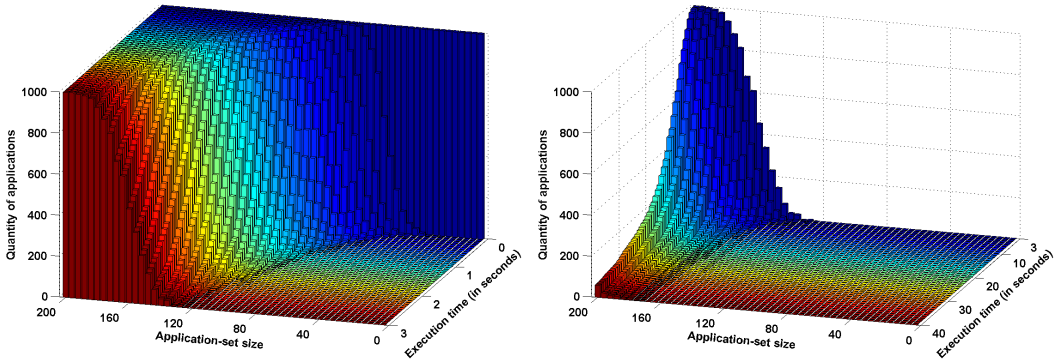


Fig. 14. The influence of the application-set size on the execution time

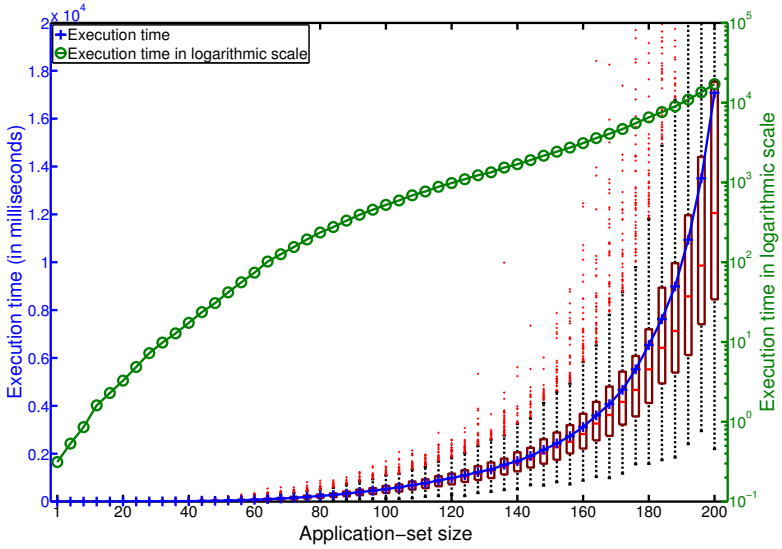


Fig. 15. The influence of the application-set size on the execution time

99.3% coverage when assuming a normal distribution, it is visible that a non-negligible amount of runs falls outside the aforementioned area, suggesting that the duration of the mapping process may hugely vary when compared to the average case and highly depends on a concrete application-set.

Overall, the mapping duration exponentially increases with the number of applications. However, it is averaging at 12 seconds for application-sets consisting of 200 applications and we thus believe that the proposed approach meets the demands and 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. 1000 sets were generated and mapped. The right part of Figure 16 shows how the variation of the grid size (horizontal axis) influences the execution time (depth axis). A cumulative reciprocal representation is used, where vertical axis stands for the amount of applications which mapping didn't complete within a given time interval.

It is visible that within 6 seconds in almost all cases the mapping completed for smaller grids, while larger grids are more time consuming and report fewer completions in the observed period. Conversely to application-set size variations, the grid size variations cause a linear increase in the duration of the mapping process, visible in the left part of Figure 16. Note, that these results suggest that on average, the mapping

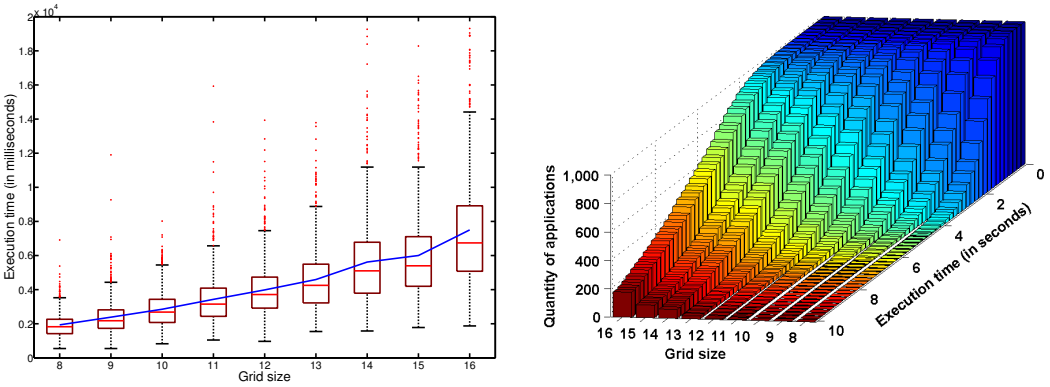


Fig. 16. The influence of the grid size on the execution time

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 that although the larger grids have to inspect more cores, at the same time it gives the opportunity to map the applications in such a way to avoid mutual contention, which decreases the duration of message feasibility checks, since fewer contentions occur.

The number of outliers again confirms huge variations in duration times of the mapping process, suggesting that for some specific application-sets the mapping can take significantly longer than the average time needed for that particular platform size.

6.3. Case Study 3: Parameter P

The parameter P controls the amount of applications which will be grouped in $\mathcal{H}$ and hence mapped during $\mathcal{IP}$, while the rest of the applications will undergo a two-stage mapping process in $\mathcal{FP}$ and $\mathcal{OP}$. Note, that $\mathcal{OP}$ was recognised as computationally the most intensive phase, so mapping more applications during $\mathcal{FP} + \mathcal{OP}$ 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 placements. On the other hand, mapping the most of the applications during $\mathcal{IP}$ may save on computation time, but makes the efficiency of the mapping process dependant on the right selection of the parameter G, as is shown in the next experiment. Assuming a constant G (in this experiment $G = 0.5$), an intuitive assumption is that the selection of the parameter P creates a trade-off between the execution time and the solution quality.

To test that, we performed an experiment where we varied the parameter P in the range $0-21$ (horizontal axis of Figure 17). Border cases present scenarios where all and none of the applications undergo the mapping during $\mathcal{IP}$, respectively. The application-set size is 150, and for each incremental step of P, 1000 sets were created and mapped on a 8×8 platform. We performed a timing analysis (right vertical axis) but also collected the qualities of generated solutions (left vertical axis), since the objective of this case study is to observe the effect of P on both execution times and solution qualities.

As expected, the left part of Figure 17 shows that the increase on the parameter P causes the duration time of the mapping process to grow exponentially. As P increases, more applications undergo more computationally extensive placement process which results in longer execution times. Until $P = 10$ the results report a logarithmic growth of the solution quality, but surprisingly, further increase of P does not necessarily cause an increase in the solution quality. It stays almost constant until $P = 17$, when it starts to decay and maintains that trend until the end of the observed interval.

These counter-intuitive results suggest that putting more applications in $\mathcal{L}$ and placing them during $\mathcal{FP}$ and $\mathcal{OP}$ might not produce a better quality solution. The explanation is that the final mapping phase - $\mathcal{OP}$, sorts the applications non-increasingly by

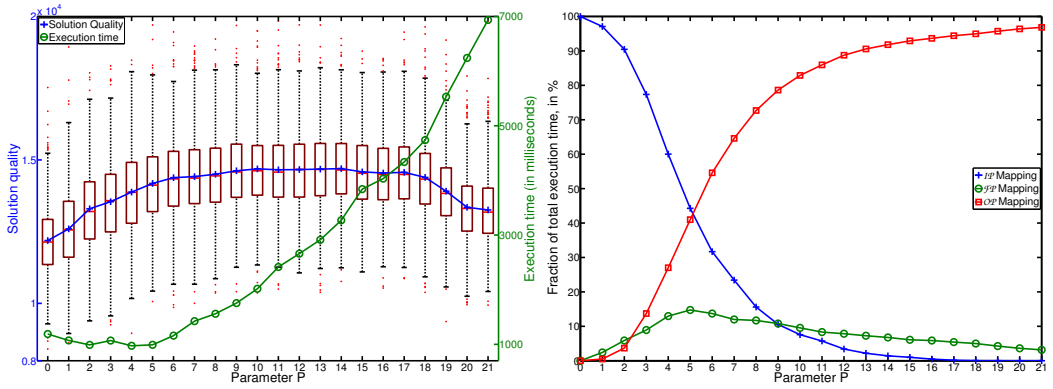


Fig. 17. The influence of the parameter P on the mapping process

the parameter m and tries to optimise their placements in that order. Since $\mathcal{OP}$ does not have a greediness control mechanism, every application will assume the widest possible placement such that its feasibility is preserved. Hence, it might happen that some application a with high $m(a)$ has very low priority. Since its placement is optimised early, every other application with a higher priority but a smaller distribution significance parameter - $a' : p(a') > p(a) \wedge m(a') < m(a)$ will be considered later than a during $\mathcal{OP}$. Any potentially better shape of the application a' which geometrically interferes with already optimised shape of a will be rejected, since the shape of a is already maximally stretched and introducing any interference from the higher priority traffic will cause its infeasibility. Therefore, even though a' has a potential to optimise its shape, it will have limited optimising possibilities, since its stretching might jeopardise the feasibility of already optimised lower priority applications. This fact prevents an expected increase in the solution quality for moderate values of P, while for higher P the effects become even more severe, causing a significant drops in solution qualities.

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. The right part of Figure 17 shows the fraction of the total execution time each phase consumes and compares them in relative terms. For small values of P, almost all applications are placed during $\mathcal{IP}$, hence leaving less workload for the subsequent phases. Until $P = 5$, $\mathcal{IP}$ witnesses an exponential decrease of the execution time, while the other two phases report a linear increase. Since $\mathcal{OP}$ has the highest computational complexity, any increase in its workload, significantly increases its execution time and hence causes, further exponential decrease of the execution time of $\mathcal{IP}$. On the other hand, $\mathcal{FP}$ experiences a steady increase in the absolute execution time, but dominated by the complexity of $\mathcal{OP}$ reports a linear decrease in relative terms and also converges towards 0. Finally, for greater values of P, the execution time is almost entirely spent in $\mathcal{OP}$ as is reported by a logarithmic growth at the end of the observed domain.

6.4. Case Study 4: Parameter G

A parameter G controls the level of greediness given to higher priority applications. Intuitive reasoning suggests that greater values of G cause greater amount of traffic as a consequence of spread placements of applications from $\mathcal{H}$. Consequently, lower priority applications, which get placed during $\mathcal{FP}$ and $\mathcal{OP}$, will have less possibilities to assume well distributed placements, since the network resources were greedily consumed by the applications from $\mathcal{H}$. In some cases this may cause applications from $\mathcal{L}$ to be unable to claim feasibility even with narrow placements, resulting with a mapping process failure. Oppositely, smaller G may limit the distribution of high priority work-

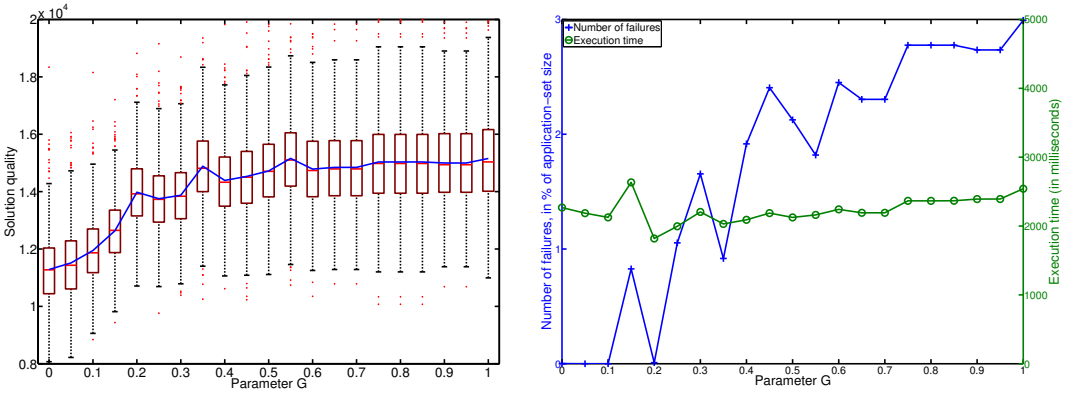


Fig. 18. The influence of the parameter G on the mapping process

load and unnecessary preserve the network for applications from $\mathcal{L}$, while there might be only few of them. This can significantly impact the quality of a derived solution.

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 constant ($P = 10$). We performed a timing analysis, but also collected the information regarding qualities of derived solutions and mapping failures, so as to study the effect of G on 1) the duration of mapping process, 2) the solution quality and 3) the amount of failed mappings. The parameter setup is identical to the one of the previous experiment. G is varied in the range $[0-1]$.

The left part of Figure 18 shows the effect of G on the solution quality. As G increases, high priority applications gain more freedom in assuming placements wider than narrow ones, resulting with a constant increase of the solution quality. This effect is noticeable until the middle of the domain. For higher values of G, we expected the solution quality to start decreasing, since high G allows greedy placements of high priority applications and hence limits the optimisation of lower priority workload during $\mathcal{OP}$. Surprisingly, the results show even a slight increase as G approaches 1.

The explanation for this finding is that for high values of G, applications from $\mathcal{H}$ try to assume very wide shapes and the mapping starts from the edges of the grid inside. If the assumed shape violates time constraints for an application, a smaller one is attempted. By performing the mapping in this way, higher priority applications are placed in such a way that the paths of their messages overlap and many common routes appear at the edges of the grid. At the end of $\mathcal{IP}$, the most of higher priority applications are distributed wide, thus leaving inner (central) cores free for lower priority applications. This implicit spacial partialisation of the workload proves to be good option is some cases. On the other hand, for smaller values of G the applications have more options to evaluate their placements and hence always choose the position which assumes less traffic interference. This allows high priority applications to avoid mutual interference and equally distribute themselves in the grid, with as less mutual interference as possible, which might be a more favourable environment in other cases.

The right part of Figure 18 shows that the duration of the mapping process is constant and does not depend on the parameter G (right vertical axis). Additionally, we observed the influence of G on the number of the applications which were unable to claim feasibility even with narrow placements (left vertical axis). As expected, higher values of G induce higher amount of traffic, which in some cases causes the mapping process to fail in deriving a solution. Note, that the results are in percentages of the application-set size, inferring that the results were very mild and not nearly severe

as we expected, which is the direct consequence of the efficient overlapped mapping of high priority workload and spacial partialisation between priorities.

6.5. 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 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 recognise 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 a lot in terms of efficiency when applied to different cases.

For instance, sets with significant differences in latency constraints of applications will benefit the most from a spacial partialisation approach invoked by low or moderate values of P and high values of G. Conversely, for sets where applications have similar temporal constraints the best strategy is to invoke balanced network utilisation by keeping P low and G moderate. In cases when applications have very tight temporal constraints two options are available for limiting the network utilisation: 1) by keeping P very high, 2) by keeping P low or moderate and setting G to a low value.

We argue that recognising the right approach in particular cases should be the responsibility of the system designer and we perceive the mapping process as an adaptive activity where different strategies are tested and consecutively fine-tuned through small variations of parameters, until a solution with desired quality is found.

7. CONCLUSIONS AND FUTURE WORK

NoC-based many-core platforms are the next frontier technology in the real-time embedded domain. This design not only allows cost reductions and power savings, but also contributes to overall system flexibility.

In this paper we addressed the issue of application mapping onto one such platform, considering the concepts of current many-core operating systems, such as Barrelfish [Baumann et al. 2009]. We gave a formal definition of the problem and proposed an efficient three-stage multi-objective mapping process which derives a placement plan for an application-set. Additionally, we proposed the analysis through which the mapping process validates derived solutions and assures that all the constraints regarding the message traversal latencies of the real-time traffic are fulfilled. We derived the metric for the qualitative assessment of the solution's potential to allow the applications to efficiently perform load balancing for various reasons (e.g. energy savings, temperature regulation, performance enhancements, dynamic accommodation of additional best-effort and non-real-time workload, fault tolerance). Finally, we proposed several constraints regarding the placement topology and the communication patterns, which efficiently solve the problems of pessimism and intractability of the analysis.

As the directions for the future work, we recognise several aspects. We plan to perform the assessment of the mapping on a concrete example and observe the behaviour, i.e. to compare the measured worst-case traversal delays against the imposed constraints, so as to test the pessimism of our approach. Furthermore, we plan to compare the performance of the mapping with and without topology and communication constraints, in order to quantitatively express the penalty suffered due to enforcing the aforementioned restrictions. We plan to extend our approach to encompass the issues regarding the schedulability of derived mappings. Finally, assuming that the mapping process provides a placement plan, and assuming that the schedulability of a derived solution is assured, we intend to estimate the overheads of agreement protocols when considering the application placements of that particular solution. The plan is to in-

corporate those overheads into the schedulability analysis, so as to provide a more realistic platform for many-core scheduling research.

REFERENCES

ASCIA, G., CATANIA, V., AND PALESI, M. 2004. Multi-objective mapping for mesh-based noc architectures. In *2nd Int. Conf. Hardware/Softw. Codesign & Syst. Synthesis*. 182–187.

BAUMANN, A., BARHAM, P., DAGAND, P.-E., HARRIS, T., ISAACS, R., PETER, S., ROSCOE, T., SCHÜPBACH, A., AND SINGHANIA, A. 2009. The multikernel: A new os architecture for scalable multicore systems. In *SOSP*.

BENINI, L. AND DE MICHELI, G. 2002. Networks on chips: a new soc paradigm. *The Comp. J. 35,* 1, 70 –78.

FERRANDIZ, T., FRANCES, F., AND FRABOUL, C. 2009. A method of computation for worst-case delay analysis on spacewire networks. In *IEEE Int. Symp. Industrial Emb. Syst.* 19–27.

HU, J. AND MARCULESCU, R. 2003a. Energy-aware mapping for tile-based noc architectures under performance constraints. In *8th ASPDAC*. 233–239.

HU, J. AND MARCULESCU, R. 2003b. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In *6th DATEE*. 10688–.

HUNG, W., ADDO-QUAYE, C., THEOCHARIDES, T., XIE, Y., VIJAYKRISHNAN, N., AND IRWIN, M. J. 2004. Thermal-aware ip virtualization and placement for networks-on-chip architecture. In *Int. Conf. Comp. Design*. 430–437.

INTEL. *Single-Chip-Cloud Computer*. `www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html`.

KREUTZ, M., MARCON, C., CARRO, L., CALAZANS, N., AND SUSIN, A. 2005. Energy and latency evaluation of noc topologies. In *Int. Symp. Circ. & Syst.* 5866 – 5869 Vol. 6.

LEI, T. AND KUMAR, S. 2003. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Euromicro Symp. Digital Syst. Design*. 180–.

MARCON, C., BORIN, A., SUSIN, A., CARRO, L., AND WAGNER, F. 2005. Time and energy efficient mapping of embedded applications onto nocs. In *10th ASPDAC*. 33 – 38 Vol. 1.

MARCON, C., MORENO, E., CALAZANS, N., AND MORAES, F. 2008. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *Digital Techn. J. 2,* 6, 471–482.

MOEIN-DARBARI, F., KHADEMZADE, A., AND GHAROONI-FARD, G. 2009. Cgmap: a new approach to network-on-chip mapping problem. *IEICE Electron. Express 6,* 1, 27–34.

MURALI, S. AND DE MICHELI, G. 2004. Bandwidth-constrained mapping of cores onto noc architectures. In *7th DATEE*. 20896–.

NI, L. M. AND MCKINLEY, P. K. 1993. A survey of wormhole routing techniques in direct networks. *The Comp. J. 26,* 2, 62–76.

NIKOLIĆ, B. AND PETTERS, S. M. 2012. Towards network-on-chip agreement protocols. In *12th EMSOFT*. IEEE, Tampere, Finland.

SHI, Z. AND BURNS, A. 2008. Real-time communication analysis for on-chip networks with wormhole switching. In *Int. Symp. Netw.-on-Chip*.

SHI, Z. AND BURNS, A. 2010. Schedulability analysis and task mapping for real-time on-chip communication. *Real-Time Syst. J. 46,* 3, 360–385.

SRINIVASAN, K. AND CHATHA, K. 2005. A technique for low energy mapping and routing in network-on-chip architectures. In *ISLPED*. 387 – 392.

TILERA. *TILEPro64 Processor*. `www.tilera.com/products/processors/TILEPRO64`.