Isolation-Aware Timing Analysis and Design Space Exploration for Predictable and Composable Many-Core Systems

Behnaz Pourmohseni
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
behnaz.pourmohseni@fau.de

Fedor Smirnov
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
fedor.smirnov@fau.de

Stefan Wildermann
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
stefan.wildermann@fau.de

Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
juergen.teich@fau.de

Abstract

Composable many-core systems enable the independent development and analysis of applications which will be executed on a shared platform where the mix of concurrently executed applications may change dynamically at run time. For each individual application, an off-line Design Space Exploration (DSE) is performed to compute several mapping alternatives on the platform, offering Pareto-optimal trade-offs in terms of real-time guarantees, resource usage, etc. At run time, one mapping is then chosen to launch the application on demand. In this context, to enable an independent analysis of each individual application at design time, so-called *inter-application isolation schemes* are applied which specify temporal/spatial isolation policies between applications. State-of-the-art composable many-core systems are developed based on a *fixed* isolation scheme that is exclusively applied to every resource in every mapping of every application and use a timing analysis *tailored* to that isolation scheme to derive timing guarantees for each mapping. A *fixed* isolation scheme, however, heavily restricts the explored space of solutions and can, therefore, lead to suboptimality. Lifting this restriction necessitates a timing analysis that is applicable to mappings with an arbitrary mix of isolation schemes on different resources. To address this issue, in this paper, we (a) present an *isolation-aware timing analysis* that – unlike existing analyses – can handle multiple isolation schemes in combination within one mapping and delivers safe yet tight timing bounds by identifying and excluding interference scenarios that can never happen under the given combination of isolation schemes. Based on the timing analysis, we (b) present a DSE which explores the choices of isolation scheme per resource within each mapping and uses the proposed timing analysis for timing verification. Experimental results demonstrate that, for a variety of real-time applications and many-core platforms, the proposed approach achieves an improvement of up to 67% in the quality of delivered mappings compared to approaches based on a fixed isolation scheme.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Embedded and cyber-physical systems; Computer systems organization → Multicore architectures

Keywords and phrases Many-core systems, timing analysis, design space exploration (DSE), isolation scheme, predictability, composability.

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2019.12

Acknowledgements This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project Number 146371743 - TRR 89 Invasive Computing.

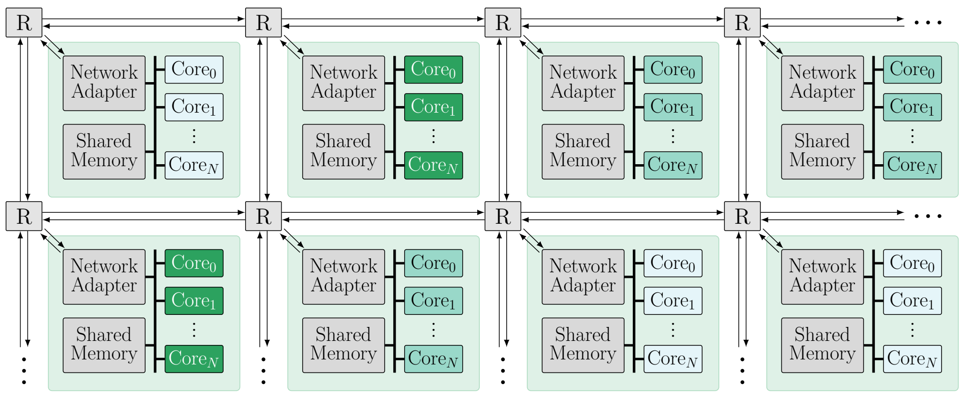

Figure 1 An example of a tiled many-core architecture composed of heterogeneous compute tiles.

1 Introduction

The ever-growing number of applications hosted in modern embedded systems introduces a high compute power demand which has given rise to the recent shift towards many-core architectures, e.g., Tilera TILE-Gx [4], Kalray MPPA-256 [9], and Intel SCC [23]. For scalability, a many-core architecture is often organized as a set of compute tiles interconnected via a Network-on-Chip (NoC), see Figure 1. Each compute tile comprises a Network Adapter (NA), a set of processing cores and peripherals such as memories which are interconnected via a set of buses. From a designer's perspective, the large number and diversity of applications and their non-functional requirements, e.g., real-time constraints, in modern embedded systems introduces an immense system design complexity. This renders the *integrated* system design approach, in which the whole system is designed at once, impractical. Over the past decade, *composable* systems, e.g. [21], have emerged to address this issue. In a composable system, applications are temporally and/or spatially isolated from each other, enabling an *incremental* system design approach where each application is first developed and analyzed individually and then added to the system on demand [1].

Composability is particularly crucial for the development of dynamic many-core systems with variable workloads and hard real-time requirements. In such systems, several independent applications are executed simultaneously, each being launched and terminated on demand and independently from others, resulting in a dynamic mix of active applications and, thus, a dynamic availability of platform resources. Each running application may be exposed to a variable workload which corresponds to a dynamic compute power demand to, e.g., meet real-time constraints. Moreover, unforeseeable conditions, e.g., thermal hot spots and hardware faults, affect the execution of applications that are currently running on the affected regions of the platform. Reserving resources according to the worst-possible workload scenario often leads to an immense underutilization of system resources which is typically not acceptable due to cost concerns. To cope with such dynamics in both resource availability of the system and resource demand of applications, *Hybrid Application Mapping (HAM)* strategies have emerged recently [45, 50]. In HAM, each application is developed and analyzed individually using an off-line Design Space Exploration (DSE) which computes several deployment options, so-called *mappings*, of the application on the platform. The computed mappings are ensured to offer diverse resource demand and performance guarantees to address various run-time resource-availability and workload scenarios, respectively. The mappings computed for each application are then provided to a so-called *run-time platform manager* which launches each application on demand using a precomputed mapping that satisfies the on-line performance requirements of the application and the resource constraints of the platform.

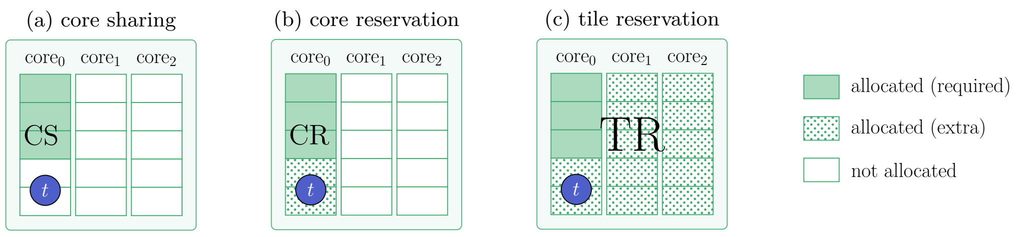

Figure 2 An example of a task t mapped to core$_0$ of a 3-core tile under (a) core-sharing, (b) core-reservation, and (c) tile-reservation isolation schemes. For a timely completion, t requires a core budget of 3 (shaded cells). Dotted cells are allocated in addition due to the isolation scheme in use.

Moreover, if the mapping in use by a running application fails, e.g., due to thermal hot spots, resource faults, or a drastic workload change, the run-time manager switches the application to another precomputed mapping which conforms to the new conditions [37].

The off-line DSE used in HAM strategies employs compute-intensive optimization and verification techniques to find high-quality mappings that offer Pareto-optimal trade-offs w.r.t. multiple – oftentimes non-linear and conflicting – design objectives, e.g., resource usage, latency, energy, etc. To achieve *predictability*, the worst-case timing properties of each mapping are bounded based on the choice of allocated resources by deriving the worst-case timing interferences that may be imposed by other (concurrent) applications which share resources with the mapping under analysis [48]. During the DSE, however, an application is developed individually, and the characteristics of the potential concurrent applications are not available to be considered in the timing analysis. In order to regulate the maximum degree of timing interferences that may be imposed by other applications, temporal and spatial isolation techniques are employed which apply certain restrictions on the accessibility of resources used by the mapping under analysis to other applications that run concurrently. These restrictions are referred to as so-called *inter-application isolation schemes*. Figure 2 illustrates three major isolation schemes that are predominantly used in many-core systems to establish composability. In all three cases, an exemplary task t is mapped to core$_0$ of a 3-core tile. For the sake of brevity, only cores are depicted in the illustration of the tile. The compute power of each core is divided into 5 budgets of equal size. For a timely completion, t requires a budget of 3 on core$_0$. In the following, each isolation scheme is explained.

- **Core Sharing (CS).** In the isolation scheme referred to as core sharing, illustrated in Figure 2a, only the 3 core budgets required for t are allocated for the mapping. Hence, concurrent applications are *temporally isolated* from the current mapping on core$_0$ and can use the 2 not-allocated budgets of core$_0$, the whole budget of core$_1$ and core$_2$, and any other on-tile resource, e.g., memories and the NA. Core sharing is the least restrictive isolation scheme which merely depends on temporal isolation on all resources.
- **Core Reservation (CR).** In the isolation scheme referred to as core reservation, illustrated in Figure 2b, core$_0$ is allocated as a whole, regardless of the budget demand of t. This realizes a *spatial isolation* from other applications on core$_0$, eliminating interferences from them on core$_0$ and resulting in an alleviated worst-case latency compared to core sharing. Note that interferences may still arise on other on-tile resources, e.g., memory buses used by t, as applications are temporally isolated from each other on those resources.
- **Tile Reservation (TR).** In the isolation scheme referred to as tile reservation, illustrated in Figure 2c, the compute tile is allocated as a whole, eliminating any interference from concurrent applications on any on-tile resource. This is the most restrictive isolation scheme which realizes a *spatial isolation* from concurrent applications at tile level and enables the largest reduction in the worst-case timing interferences imposed on t.

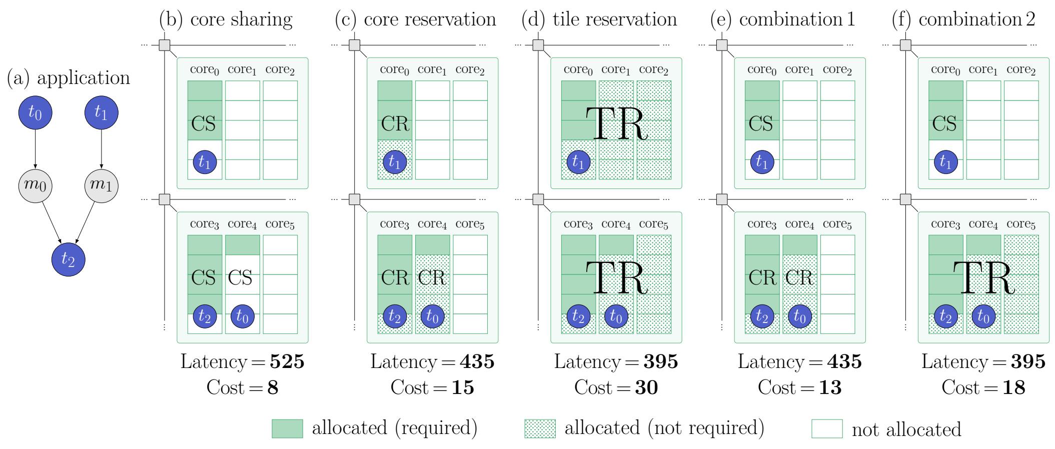

Figure 3 (a) an example of an application. A mapping of the application on two adjacent tiles is illustrated for the following isolation scenarios: (b) exclusive core sharing, (c) exclusive core reservation, (d) exclusive tile reservation, (e) core sharing on core$_0$ and core reservation on core$_3$ and core$_4$, (f) core sharing on core$_0$ and tile reservation on the lower tile. The core budget required for each task is shaded on the respective core. For each scenario, the worst-case latency of the mapping (derived based on Table 1a–b) and its allocation cost are given below the respective sub-figure.

Noteworthy, sharing the NoC which interconnects the tiles can hardly be avoided [34]. Hence, applications are always *temporally isolated* from each other on the NoC. State-of-the-art composable many-core systems are designed based on a *fixed isolation scheme* that is uniformly applied to *every* core/tile of *every* mapping of *every* application in the system and is coupled with a timing analysis *tailored* to that specific isolation scheme to derive tight timing guarantees. The choice of isolation scheme highly impacts the resource usage and worst-case timing characteristics of the mappings. As an example, consider the application shown in Figure 3a and a mapping of it on two adjacent 3-core tiles illustrated in Figure 3b–f. For the sake of this motivational example, assume that the timing analysis of the tasks yields the qualitative Worst-Case Response Time (WCRT) values given in Table 1a under different isolation schemes. Likewise, the qualitative Worst-Case Traversal Time (WCTT) of message m_1 (communicated between the two tiles, from t_1 to t_2) is given in Table 1b for various combinations of isolation scheme on m_1's source and destination cores/tiles. Message m_0 is implicitly communicated between t_0 and t_2 via the tile's shared memories. In this example, applying core sharing exclusively, as illustrated in Figure 3b, offers the minimum allocation cost, i.e., 8 core budgets, at the expense of considerably high worst-case

Table 1 Qualitative timing properties assumed for tasks and messages in the example illustrated in Figure 3: (a) WCRT of tasks t_0, t_1, and t_2 under different isolation schemes, (b) WCTT of message m_1 for different combinations of isolation scheme on m_1's source and destination cores/tiles.

(a) Worst-Case Response Time assumed for t_0–t_2

isolation scheme	WCRT(t_0)	WCRT(t_1)	WCRT(t_2)
TR	290	75	105
CR	315	115	120
CS	380	165	145

(b) Worst-Case Traversal Time assumed for m_1

source isolation scheme	destination isolation scheme	WCTT(m_1)
TR	TR	12
TR	CR/CS	15
CR/CS	TR	17
CR/CS	CR/CS	20

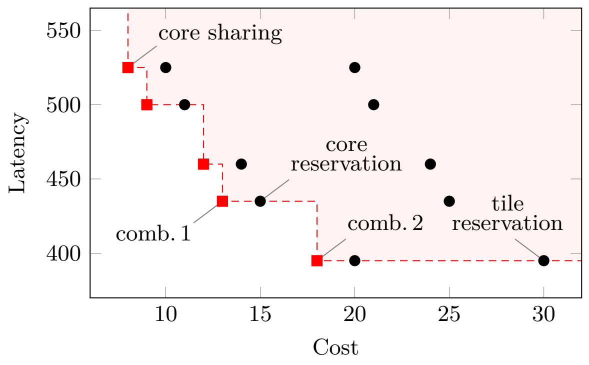

Figure 4 The latency-cost trade-offs offered by the 15 possible isolation scheme scenarios for the application mapping from Figure 3. Pareto-optimal trade-offs are designated by red squares, dominating the highlighted space. Labeled trade-offs correspond to the scenarios illustrated in Figure 3.

interferences on cores and other on-tile resources, resulting in an end-to-end application latency of 525 time units in the worst case. Applying core reservation exclusively, as depicted in Figure 3c, results in an elevated allocation cost of 15 budgets. It, however, eliminates core interferences which alleviates the worst-case application latency to 435 time units. Applying tile reservation exclusively, as depicted in Figure 3d, minimizes the worst-case application latency to 395 time units at the expense of a maximized allocation cost of 30 budgets.

Motivation. A *fixed isolation scheme* that is always applied exclusively – which is the common practice in state-of-the-art composable many-core systems – heavily restricts the space of explored mappings and excludes numerous promising solutions where multiple isolation schemes are applied *in combination*. For instance, in the example above, considering arbitrary combinations of isolation schemes within one mapping enables 12 additional isolation scenarios, two of which are depicted in Figure 3e–f. In combination 1, core sharing is applied on core$_0$ and core reservation is applied on core$_3$ and core$_4$, offering the same latency as the exclusive core-reservation scenario from Figure 3c, but at a lower cost. Likewise, combination 2 applies core sharing on core$_0$ and tile reservation on the lower tile, outperforming the exclusive tile-reservation scenario from Figure 3d by offering the same latency at a considerably lower cost. Having evaluated all possible combinations of isolation scenarios, Figure 4 illustrates the obtained cost-latency trade-offs where Pareto-optimal trade-offs are designated by red squares, dominating the highlighted space above the red dashed line. The trade-offs corresponding to the scenarios in Figure 3 are labeled accordingly in Figure 4. As shown, nearly always, Pareto-optimal trade-offs are obtained only when multiple isolation schemes are applied in combination within one mapping. In Section 6, we experimentally verify this observation for realistic use cases as well.

Contribution. The example above demonstrates that using multiple isolation schemes in combination yields mappings of a better quality trade-off. Deriving safe bounds on the worst-case timing characteristics of such mappings, however, requires an *isolation-aware timing analysis* that can capture the impact of the isolation scheme of each core/tile on the worst-case timing behavior of tasks/messages affected by it and, hence, on the worst-case timing behavior of the application. On the one hand, existing timing analyses applicable to composable many-core systems are *tailored* to one fixed isolation scheme and cannot handle multiple isolation schemes in combination within one mapping. On the other hand, from an architectural point of view, many-core platforms are undergoing a transition from single-core tiles, e.g., in [4], to multi-core tiles, e.g., in [9]. This transition introduces new sources of timing interference within each tile, e.g., on shared memory buses, which must be accounted for to obtain safe timing guarantees. In the context of composable systems, there exist no timing analysis known to us which accounts for all sources of timing interference that may arise in a many-core platform with multi-core tiles (without applying restrictive spatial isolation schemes that eliminate on-tile inter-application interferences altogether). Provided with an isolation-aware timing analysis, the off-line DSE in HAM strategies can be extended to also explore the choices of isolation scheme for each core/tile within each investigated mapping to obtain solutions with a better quality trade-off. In this line, the paper at hand makes the following contributions in the context of composable many-core systems:

1. We present an *isolation-aware timing analysis* which can handle arbitrary combinations of isolation schemes within one mapping and accounts for all timing interferences that may arise within a shared multi-core tile and on the NoC interconnecting tiles. It derives tight timing bounds by identifying and excluding interference scenarios that can never happen under the current combination of isolation schemes.
2. We present an *isolation-aware DSE* which explores the choices of isolation scheme for each core/tile in each investigated mapping of the application under analysis and uses the proposed timing analysis for the scheduling and timing verification of the mappings.

For a variety of hard real-time applications and many-core platforms, we experimentally verify the advantage of the proposed isolation-aware exploration and timing analysis approach over existing fixed-isolation-scheme approaches in terms of the quality of obtained mappings.

Organization. The remainder of this paper is organized as follows. In Section 2, related work on timing analysis and DSE of multi-/many-core systems is reviewed. Section 3 presents the preliminaries and the system model for this work. The proposed isolation-aware DSE and timing analysis are presented in Sections 4 and 5, respectively. Experimental results are discussed in Section 6 before the paper is concluded in Section 7.

2 Related Work

Worst-case timing analysis of applications in multi-/many-core systems has long been conducted in two steps: First, a context-independent analysis is applied to bound the Worst-Case Execution Time (WCET) of each task in isolation, i.e., in absence of interferences. An overview of tools and methods for context-independent WCET analysis is provided in [51]. Following the context-independent analysis, an interference analysis is performed to bound the additional latencies that may be imposed on shared resources due to external interferences.

Multi-/many-core timing analyses predominantly focus on worst-case interference analysis based on the context-independent characteristics of each task and message. For instance, the analyses presented in [2, 6, 8, 15, 17, 40, 41, 46] bound the WCRT of tasks in a multi-core setup where several cores are connected to one or more memories over shared buses. In [2, 8], a framework for multi-core response time analysis is presented for a preemptive Fixed-Priority (FP) core scheduling policy coupled with multiple memory bus arbitration policies, e.g., Time-Division Multiplexing (TDM), Round-Robin (RR), and FP. Targeting mixed-criticality systems, [15, 17] analyze WCRT under a non-preemptive FP core scheduling policy and a RR memory bus arbitration. The authors of [40, 46] present response time analyses for non-preemptive FP core scheduling policy and a multi-level bus arbitration scheme which combines RR and FP policies. In [41], memory bus interference is bounded using an ILP-based timing analysis under a RR bus arbitration policy. The framework presented in [6], analyzes memory bus interference under a variety of arbitration policies, e.g., TDM and FP. Concerning the NoC, in [43] and [54] NoC transfer delays are analyzed under FP and RR link arbitration policies, respectively.

The works listed above consider – to some extent – non-preemptive and/or *contention-oriented* arbitration/scheduling policies, e.g., priority-based schemes, for which safe timing guarantees can be derived only if the whole set of applications accessing shared resources is known at the time of analysis. In a composable system, however, the mix of concurrently running applications that share some resources may not be known at design time. Hence, contention-oriented policies can be applied in a composable system only if each and every resource with such an arbitration scheme is exclusively used by one application. To realize this on state-of-the-art many-core platforms, each mapping computed for each application must (a) have a tile-reservation isolation scheme only, (b) be restricted within one tile only, and (c) not rely on inter-tile communications, e.g., I/O transfers. In practice, however, the small number of cores comprised within one tile [4, 23], on the one hand, and the high compute power demand of each application, on the other hand, necessitate the deployment of some applications over several compute tiles to meet their real-time requirements. For such applications, NoC links could – in theory – be exclusively reserved per application to enable the timing analysis of multi-tile mappings using NoC delay analyses, e.g. [54, 43]. In practice, however, sharing NoC links among applications can hardly be avoided [34].

Contention-free arbitration policies based on time slicing, e.g., TDM and Weighted Round-Robin (WRR), offer a practical approach for predictable inter-application resource sharing without violating composability. For instance, the authors of [48, 49] consider a NoC [22] with WRR link arbitration policy and analyze worst-case NoC delays based on the reserved link budget for each communication, independent of the other communication flows that may share the same links. For WCRT analysis, [48] considers a preemptive RR core scheduling policy which, however, restricts its scope of coverage to core interferences originating from within the application under analysis only. In [49], on the other hand, a WRR core scheduling policy is considered, and based on that, a response time analysis is presented which accounts also for core interferences that may be imposed by other (currently unknown) applications. Both [48, 49], however, consider single-core tiles and, hence, cannot capture NA and memory bus interferences that may arise in a *multi-core tile*, e.g., in [9, 23]. Contrarily, we present a timing analysis that captures also the NA and memory bus interferences in multi-core tiles.

From an isolation scheme point of view, existing multi-/many-core timing analyses are tailored to a fixed isolation scheme. For instance, the analyzes in [2, 6, 8, 15, 17, 40, 41, 46] are applicable only under tile-reservation isolation scheme. The analysis presented in [48] is applicable to systems with single-core tiles only and assumes tile-reservation isolation scheme. Authors in [49] consider single-core tiles and core-sharing isolation scheme. The timing analysis presented in this paper is the first timing analysis known to us which can handle arbitrary mixes of isolation schemes in combination within one mapping.

Application mapping in multi-/many-core systems is typically viewed as a multi-objective optimization problem and is known to be NP-hard [14]. Due to its immensely large space of possible mapping solutions, using exact optimization approaches, e.g., enumeration or (integer-)linear programming, to solve this NP-hard problem demands an extremely high computational effort and is prohibitively time-consuming, except for very small/simple problems. In this context, *meta-heuristic optimization* approaches, e.g., evolutionary/genetic algorithms [7, 13, 12], simulated annealing [27], and particle swarm optimization [25], enable a scalable optimization approach that can deliver high-quality solutions at a reasonable time- and computational effort. As a result, they have become the de facto standard approach for multi-objective application mapping optimization in multi-/many-core systems. For instance, in [33, 36, 48, 49], evolutionary/genetic algorithms are used for mapping optimization. In [16, 17], mapping optimization is realized using simulated annealing algorithms, while [42] adopts particle swarm optimization. The majority of existing works on mapping optimization in HAM methodologies employ *population-based* meta-heuristics, e.g., particle swarm optimization or evolutionary/genetic algorithms, to collect a so-called *population* of best mappings. During the Design Space Exploration (DSE), the optimizer follows the course of several iterations to generate new mappings and evaluate them w.r.t. the given set of design objectives, e.g., resource cost, timing, or energy. It collects a population of explored mappings which is updated per iteration to retain the best solutions found so far and is used for the generation of mappings in the next iteration. Like [33, 36, 48, 49], we employ a multi-objective evolutionary algorithm for mapping optimization.

To the best of our knowledge, existing DSE proposals on mapping optimization in the context of HAM strategies, e.g., [26, 33, 36, 47, 44, 48, 49, 53], consider a fixed isolation scheme that is applied exclusively to *every* resource (core or tile) of *every* explored mapping of *every* application. For instance, the DSE approaches in [36, 33, 53] assume a core-reservation isolation scheme. In [26, 44, 48], a tile-reservation isolation scheme is assumed. In [49], a core-sharing isolation scheme is assumed. The DSE proposed in this paper does not assume a globally-fixed isolation scheme. Instead, it explores the choices of isolation scheme per allocated resource within each explored mapping. This provides a fine-grained control over the degree of admitted inter-application interferences (which we analyze using the proposed timing analysis) and renders many mappings with promising quality trade-offs reachable to the DSE which are not reachable under a fixed isolation scheme.

3 Preliminaries

3.1 Mapping Optimization Problem Specification

Similar to any other optimization problem, the multi-/many-core mapping optimization problem requires a problem model which describes the space of possible solutions and the conditions that must be satisfied by a mapping to be regarded a valid solution. In this paper, we use the graph-based system model from [3]. This model represents the mapping optimization problem by a so-called *specification* which describes the entire design space and is used to generate different mapping solutions. The specification and each mapping generated based on that consist of an *application graph*, an *architecture graph*, and *mapping edges* connecting them, which will be introduced in Sections 3.1.1 to 3.1.3.

3.1.1 Application Model

We consider periodic hard real-time applications. Each application is specified by an application graph $G_P(T \cup M, E)$ where T denotes the set of (processing) tasks, M denotes the set of messages exchanged between tasks, and edges $e \in E$ define data dependencies among tasks and messages, e.g., see Figure 5a. For each task $t \in T$, the execution period $\mathrm{PRD}(t)$, the context-independent worst-case execution time $\mathrm{WCET}(t, c)$ on each mappable core c, and the memory demand $\mathrm{MD}(t)$ (maximum number of memory accesses) per execution iteration

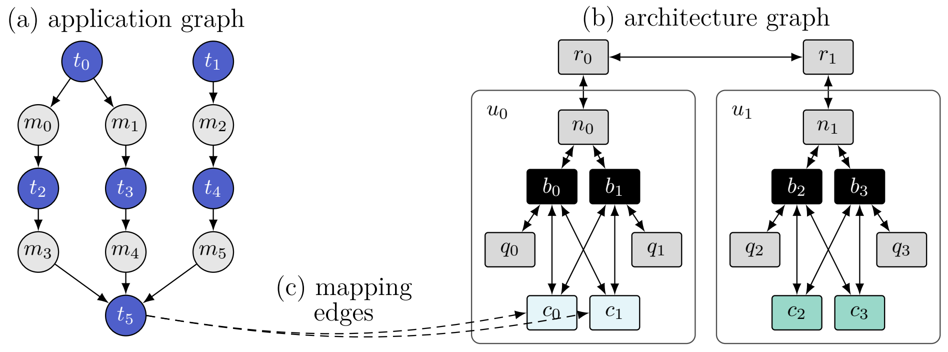

Figure 5 Specification of an exemplary mapping optimization problem, composed of (a) application graph, (b) architecture graph, and (c) mapping edges connecting them (depicted only for task t_5).

are known. For each message $m \in M$, the transfer period $\mathrm{PRD}(m)$ and the maximum payload size $\mathrm{PLD}(m)$ together with its corresponding memory demand $\mathrm{MD}(m)$ (number of memory accesses for reading/writing m from/to memory) are given.

3.1.2 Architecture Model

We consider heterogeneous tiled many-core architectures composed of multi-core tiles interconnected by a NoC. The platform architecture is specified by an architecture graph $G_A = (R \cup N \cup B \cup C \cup Q \cup U, L)$, e.g., see Figure 5b. Here, $r \in R$ denotes a NoC router, $n \in N$ a NA, $b \in B$ a memory bus, $c \in C$ a processing core, and $q \in Q$ a shared memory (or memory bank) with an own memory bus, respectively. Edges $l \in L$ represent bidirectional connections between these resources. Each $u \in U$ represents a compute tile which comprises a set of cores and memories and a NA, thus, $u \subseteq N \cup B \cup C \cup Q$. Each memory (or memory bank) is accessible to cores and the NA on the same tile via a shared bus. Each NA consists of separate Transmitter (TX) and Receiver (RX) units with own ports to each memory bus. We assume many-core architectures without timing anomalies [39]. This enables a compositional analysis of execution, communication, and memory latency contributions which can be combined to derive globally safe timing guarantees.

Memory Model. We consider a No Remote Memory Access (NORMA) scheme which is commonly practiced in many-core systems to achieve scalability [32]. Under a NORMA scheme, the memories located on one tile are not accessible to resources located on other tiles, thus, data exchanges among different tiles are realized exclusively by means of explicit message passing between them. To achieve storage composability, (a) the on-tile memory space is partitioned, such that each core is given a dedicated storage space. Likewise, (b) shared caches are partitioned or disabled. In the same line, (c) the dedicated storage space of each core is dynamically partitioned among the tasks hosted by it. Finally, (d) each message is provided with a dedicated memory space in each tile where it is produced and/or consumed. To perform a memory access, the requestor must first attain the ownership of the shared bus associated with the target memory. We assume blocking and indivisible memory accesses, so that the processing progress on the requestor side is stalled during the memory operation. Each memory access is a single-word operation with a known maximum service time, eliminating burst/block operations. A requestor may perform several consecutive but non-overlapping single-word memory operations during its bus ownership interval.

NoC Model. We consider wormhole-switched NoCs with a credit-based virtual-channel flow control, for instance [22], to allow per-link bandwidth reservation and, thereby, to enable link sharing without violating composability. Under a *wormhole-switched* flow control, data packets are decomposed into control flow digits (flits) of fixed size which are then routed over the NoC independent from each other in a pipeline fashion [35]. The *virtual-channel* flow control [5] provides multiple buffers per physical link to enable transmission preemption and composable link sharing. The transmission progress of flits in each buffer of a router is controlled using so-called *credits* which reflect the availability of buffer space at the next router. This realizes a backpressure mechanism inherently.

Communication Model. Intra-tile communications, i.e., data exchanges *within* one tile, are realized through a dedicated space in the tile's shared memories. Inter-tile communications, i.e., data exchanges *between* tiles, are realized by explicit message passing over the NoC. For the latter, after the data is written by the sender into the dedicated memory space, the TX reads the data, decomposes it into flits, and injects the flits into the NoC. The flits are then routed over the NoC toward the destination tile where the RX reconstructs the data from the flits and writes it into a dedicated memory space to be read by the receiver thereafter.

Resource Arbitration. Any hardware resource, e.g., cores, memory buses or the NoC, that can be shared among multiple applications is assumed to have an arbitration policy that is both predictable and composable. While (a) *predictability* enables formal worst-case timing analysis of each application, (b) *composability* ensures that worst-case timing bounds can be derived for each application merely based on the resource budgets reserved for it, without any knowledge about other applications that may run concurrently to it and share resources with it. This definition necessitates a *contention-free* arbitration policy for each and every resource that may be shared among applications. Time-Division Multiplexing (TDM) and Weighted Round-Robin (WRR) are well-established predictable and composable arbitration policies which serve as primary candidates for composable many-core systems [19, 21, 22]. Both TDM and WRR establish temporal isolation using time-triggered preemption, dividing the access to a resource into time slots of equal length that are periodically assigned to requestors which have reserved one or more slots on that resource. TDM is not work-conserving[1] which leads to a poor average-case performance, making it an unattractive candidate for, e.g., systems hosting both real-time and best-effort applications [24]. Contrarily, WRR provides predictability and composability in a work-conserving fashion by skipping over idle slots. This enables a notable average-case performance improvement in favor of best-effort applications while allowing worst-case timing guarantees to be derived for real-time applications. We assume each shared resource has a preemptive time-triggered arbitration policy that follows similar principles as TDM and WRR.

3.1.3 Mapping Edges

In the specification of the mapping optimization problem, the application graph and the architecture graph are connected by mapping edges $V \subseteq T \times C$, e.g., see Figure 5c. Each mapping edge $v = (t, c) \in V$ indicates that task $t \in T$ can be executed on core $c \in C$.

[1] In a work-conserving arbitration policy, time slots are assigned only to requestors with pending requests while idle slots, i.e., those without an access request, are skipped. This scheme results in a varying arbitration period and varying position of assigned slots for each requestor within one arbitration period.

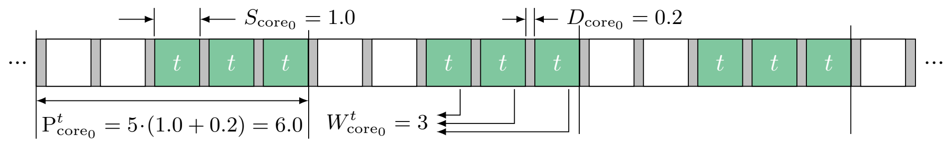
$S_{\text{core}_0} = 1.0$
$D_{\text{core}_0} = 0.2$
t
$\mathrm{P}^t_{\text{core}_0} = 5\cdot(1.0+0.2) = 6.0$
$W^t_{\text{core}_0} = 3$

Figure 6 Arbitration tuple of task t on core$_0$ from Figure 2a, calculated as $(1.0, 3, 6.0)$.

3.2 Arbitration Tuple

In this work, all parameters relevant for the worst-case timing analysis of a requestor on a resource, i.e., the budget reserved for it and the worst-case interference that can be imposed by other requestors, are compactly reflected by a so-called *arbitration tuple*: For each requestor x of resource r, the arbitration tuple is represented as (S_r, W_r^x, P_r^x) where S_r denotes the length of one arbitration slot of r, W_r^x denotes the number of periodic arbitration slots (weight) reserved for x on r, and P_r^x denotes the worst-case arbitration period perceived by x on r. Given the arbitration tuple, one can deduce that x has a periodic reserved time budget of $(W_r^x \cdot S_r)$ on r and, thus, experiences a worst-case wait time of $(P_r^x - W_r^x \cdot S_r)$ per arbitration period P_r^x. We exemplify the calculation of the arbitration tuple for task t in Figure 2a under an arbitration policy with a slot length of $S_{\text{core}_0} = 1.0$, an arbitration delay of $D_{\text{core}_0} = 0.2$ between consecutive slots[2], and an arbitration capacity of $K_{\text{core}_0} = 5$ slots per period. Under such an arbitration policy, task t with 3 reserved periodic slots perceives an arbitration weight of $W^t_{\text{core}_0} = 3$ and a worst-case arbitration period of $P^t_{\text{core}_0} = 5 \times (1.0 + 0.2) = 6.0$ based on which the arbitration tuple is created as $(1.0, 3, 6.0)$. This calculation is also illustrated in Figure 6. Note that, the arbiter delay is reflected in the calculation of arbitration period, rendering the arbitration tuple expressive of realistic resource arbiters in practice.

It may happen that the isolation scheme of a resource leads to one or more arbitration slots to be allocated by the mapping under analysis but not utilized. Given a work-conserving arbitration policy, e.g. WRR, in such cases we reduce the arbitration capacity of that resource to exclude those slots that are never utilized and, hence, are always skipped by the arbiter. For instance, in Figure 2b, core$_0$ is allocated exclusively where only 3 slots are utilized by t while the remaining 2 slots are never utilized. Thus, given a work-conserving arbitration policy, the arbitration period is guaranteed not to exceed $P^t_{\text{core}_0} = (5-2) \times (1.0 + 0.2) = 3.6$, resulting in an adapted arbitration tuple of $(1.0, 3, 3.6)$ for t on core$_0$.

The proposed timing analysis presented in Section 5 takes the arbitration tuples as input and derives safe bounds on the worst-case timing characteristics of each task and message under the combination of applied isolation schemes. For the calculation of the arbitration tuples, the arbitration policy of each resource r and its parameters, namely, arbitration slot length S_r, arbitration delay D_r, and arbitration capacity K_r, are provided by the architecture, while both the isolation scheme of each resource r and the number W_r^x of slots reserved for each analyzed requestor x are decided by the mapping optimizer during the DSE. We calculate arbitration tuples for the following requestors:

- For each task $t \in T$, the arbitration tuple on core $c \in C$ executing t is calculated and denoted as (S_c, W_c^t, P_c^t).
- For each inter-tile message $m \in M$, the arbitration tuple (a) on the TX tx injecting m into the NoC, (b) on the NoC route ρ (sequence of links) over which m is routed, and (c) on the RX rx receiving m from the NoC are calculated and denoted as $(S_{tx}, W^m_{tx}, P^m_{tx})$, $(S_\rho, W^m_\rho, P^m_\rho)$, and $(S_{rx}, W^m_{rx}, P^m_{rx})$, respectively.

[2] The arbitration delay denotes the latency of the arbiter for switching between consecutive arbitration slots. For instance, on a core, it corresponds to the context-switch overhead of the operating system.

- For each core $c \in C$ that has at least one task mapped to it, the arbitration tuple on each on-tile memory bus $b \in B$ is calculated and denoted as (S_b, W_b^c, P_b^c).
- For each TX tx that routes at least one outbound message out of the tile, the arbitration tuple on each on-tile memory bus $b \in B$ is calculated and denoted as $(S_b, W_b^{tx}, P_b^{tx})$.
- For each RX rx that routes at least one incoming message into the tile, the arbitration tuple on each on-tile memory bus $b \in B$ is calculated and denoted as $(S_b, W_b^{rx}, P_b^{rx})$.

4 Isolation-Aware Design Space Exploration (DSE)

This section presents our isolation-aware DSE approach, illustrated in Figure 7. The DSE takes as input the specification of the mapping optimization problem comprising the application graph, the architecture graph, and mapping edges, and delivers a set of mappings that offer Pareto-optimal trade-offs w.r.t. a given set of design objectives, e.g., latency, throughput, energy, and resource usage. The DSE employs a *mapping optimizer* to explore the space of possible mappings of the application on the target architecture. The optimizer creates each mapping by conducting a series of binding, isolation, routing, allocation, and scheduling design decisions, elaborated later in Section 4.1. Once a mapping is generated, it is provided to a set of *evaluators* to assess the quality of the mapping w.r.t. the given design objectives. The proposed isolation-aware timing analysis, which will be presented in Section 5, is used here as an evaluator to derive safe bounds for timing-related design objectives, namely, latency and throughput. The optimizer uses an evolutionary algorithm which conducts several iterations to generate new mappings and collect a set of Pareto-optimal mappings.

The Pareto-optimal mappings will be used at run time to launch the application on demand by selecting a mapping that complies with the current real-time constraints of the application and resource availability of the system (restricted due to, e.g., other running applications). Since our timing analysis accounts for the worst-case interferences that may be imposed by any mix of (statically unknown) concurrent applications, the worst-case timing characteristics it provides for each mapping are guaranteed to hold regardless of the mix and deployment of other applications at run time. This allows the DSE to be performed for each application *individually* without any knowledge about the other applications in the system.

4.1 Mapping Creation

The creation of each mapping starts by making the *binding* and the *isolation* design decisions:

- **Binding.** In this step, each task of the application is bound to a core on the architecture. We use the SAT-Decoding approach [29] to explore the binding of tasks to cores. By encoding the constraints from [31], we ensure that each task is bound exactly to one core.
- **Isolation.** In this step, an isolation scheme is selected for each core and each tile. To this end, each core and each tile is decided to be either *shared* or *reserved*. We implement the exploration of isolation schemes using the SAT-Decoding approach [29], see also [18].

The creation of the mapping is completed by deriving the implications of binding and isolation decisions on message *routing*, resource *allocation*, and *scheduling* of tasks and message:

- **Routing.** In this step, for each inter-tile message (i.e., a message communicated between two tasks which, according to the binding decisions, are bound to different tiles), a NoC route (sequence of links) is determined over which the message is transferred between the two tiles. Without loss of generality, we use the XY-routing algorithm [35].
- **Allocation.** In this step, the processor budget required for the mapping is allocated according to the previously made binding and isolation decisions: If a task is bound onto any core of a reserved tile, the whole tile is allocated completely and, thus, none of the

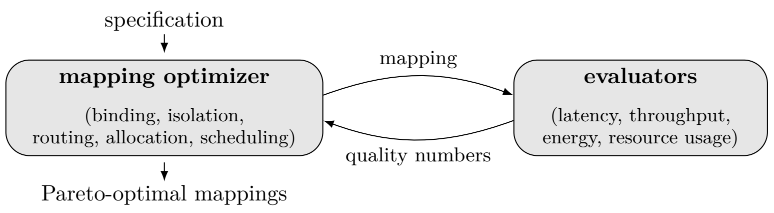
specification
mapping optimizer
(binding, isolation, routing, allocation, scheduling)
Pareto-optimal mappings
mapping
quality numbers
evaluators
(latency, throughput, energy, resource usage)

Figure 7 Overview of the proposed isolation-aware design space exploration (DSE) approach.

resources on that tile can be used by other applications (tile reservation). If a task is bound to a reserved core on a shared tile, that particular core is allocated completely while other resources on the tile can be used by other applications (core reservation). Finally, if a task is bound to a shared core on a shared tile, only the minimum budget required by that task will be allocated on that core while the remaining core budget can be used by other applications (core sharing). Any core and tile that does not correspond to one of the cases above is not allocated and, therefore, can be used by other applications. We also allocate the minimum budget required for each inter-tile message on NoC links that are part of the message's route specified in the routing step.

- **Scheduling.** We elaborate on the scheduling step in Section 4.2.

4.2 Isolation-Aware Scheduling

In the scheduling step, the arbitration tuples listed in Section 3.2 which are required for the timing analysis of the mapping are calculated. In what follows, we first calculate in Section 4.2.1 the resource budgets (arbitration weight) required for tasks and messages. Based on these weights, the arbitration tuples are then calculated in Section 4.2.2.

4.2.1 Resource Budget Calculation

To calculate the arbitration weight of each task/message $x \in T \cup M$ on each respective resource r, first the worst-case arbitration period P_r^x for x on r is calculated (as presented in Section 3.2) based on r's arbitration capacity K_r and arbitration slot length S_r which are provided by the architecture. Then, for each task t bound to core c, we start with an arbitration weight of $W_c^t = 1$ and iteratively (I) construct the arbitration tuple (S_c, W_c^t, P_c^t) and (II) use Equations (1) to (4) from Section 5 to determine t's WCRT for the current arbitration weight W_c^t. If the WCRT of t exceeds its deadline given by the application, i.e., $\text{WCRT}(t) > \text{PRD}(t)$, we (III) increment W_c^t by one and go back to step (I). Otherwise, the iterations are terminated and the current arbitration weight is considered for t. Likewise, for each inter-tile message m routed via TX unit tx, RX unit rx, and NoC route ρ, we use Equations (5) to (8) from Section 5 to calculate the minimum arbitration weight $W_{tx}^m = W_{rx}^m = W_\rho^m$ such that m's WCTT does not exceed its production period, i.e., $\text{WCTT}(m) \le \text{PRD}(m)$.

After the weights are derived, we evaluate the overall weight demanded on each resource. If the weight demanded by all tasks mapped to a core exceed the core's arbitration capacity, the mapping is considered as infeasible and is discarded. Likewise, if the weight demanded on a NoC link, a TX or a RX by messages routed over it exceeds its arbitration capacity, the mapping is considered as infeasible and discarded.

4.2.2 Arbitration Tuple Calculation

Given the arbitration weights of tasks and messages on their respective resources, calculated in Section 4.2.1, and the isolation scheme of each core/tile, the arbitration tuples for tasks and messages are calculated as follows. For each exclusively allocated core c, first the arbitration capacity is reduced as $K_c = \sum_{t \in T} W_c^t$ to discard scheduling slots that are not utilized by the tasks mapped to it, and the arbitration period P_c^t of each task $t \in T$ mapped to it is refined accordingly as presented in Section 3.2. Then, the arbitration tuple (S_c, W_c^t, P_c^t) is created for each task $t \in T$ mapped to c where the scheduling slot length S_c is provided by the architecture. Likewise, for each exclusively allocated tile, first the arbitration capacity of the TX unit tx is reduced as $K_{tx} = \sum_{m \in M} W_{tx}^m$ to reflect only the arbitration slots reserved for the outbound messages $m \in M$ of that tile within the current mapping. Then, the arbitration period P_{tx}^m of each outbound message m on tx is refined, and the arbitration tuple $(S_{tx}, W_{tx}^m, P_{tx}^m)$ is constructed. A similar procedure is followed for the incoming messages of the tile. Since the NoC is assumed to be shared, no reduction will be applied to the capacity of NoC links, and thus, no refinement is required for the calculation of the arbitration tuple $(S_\rho, W_\rho^m, P_\rho^m)$ of an inter-tile message $m \in M$ on the links of its NoC route ρ.

For each core $c \in C$, the arbitration tuple (S_b, W_b^c, P_b^c) on each memory bus $b \in B$ connected to it is calculated as follows. The bus arbitration slot length S_b, the bus arbitration capacity K_b, and the core arbitration weight W_b^c on the bus are provided by the architecture. The arbitration period P_b^c perceived by c on bus b is calculated according to the decided isolation scheme: If a tile-reservation isolation scheme is selected, the bus arbitration capacity K_b is reduced to exclude the arbitration slots corresponding to idle cores (cores that do not host any tasks). Accordingly, the refined arbitration period P_b^c is calculated as presented in Section 3.2. For each TX and RX, the arbitration tuples $(S_b, W_b^{tx}, P_b^{tx})$ and $(S_b, W_b^{rx}, P_b^{rx})$ on each memory bus $b \in B$ connected to them is calculated similarly. Note that all arbitration capacity reductions discussed above are applied only in case of a work-conserving arbitration policy, e.g., WRR. Otherwise, the arbitration capacities remain unaffected.

5 Isolation-Aware Timing Analysis

This section presents the proposed timing analysis which formally bounds the WCRT of each task $t \in T$ and the WCTT of each inter-tile message $m \in M$ using the arbitration tuples calculated in the scheduling step. The timing analysis is performed compositionally [20], so that the worst-case timing behavior of each task/message is decomposed into several timing contributions on different resources which are analyzed separately and, then, are combined to obtain globally safe timing guarantees. The obtained task/message latency bounds are then used to bound the worst-case latency (makespan) and throughput of the mapping.

5.1 Worst-Case Response Time

The WCRT of a task denotes the worst-case time interval between its start time and completion time, subject to the worst-case timing interferences that may arise on shared resources due to the presence of interfering requests. In each iteration of its execution, a task reads its required data and input messages from the memory, performs its processing, and writes its output messages into the memory. In general, two sources of interference may occur here: (a) bus interferences when accessing memory and (b) preemption delays on the cores. Therefore, the WCRT of each task $t \in T$ can be bounded as:

$$\mathrm{WCRT}(t, c, q, b) = \mathrm{WCET}(t, c) + \mathrm{MD}(t) \cdot \mathrm{ST}(q, b) + I^{\mathrm{bus}}(t, c, q, b) + I^{\mathrm{core}}(t, c, q, b) \quad (1)$$

where $\mathrm{WCET}(t, c)$ denotes the worst-case execution time of t on core c in isolation, assuming a zero delay for bus and memory. $\mathrm{MD}(t)$ denotes the memory demand (number of single-word memory accesses) of t, and $\mathrm{ST}(q, b)$ denotes the service time of memory q over bus b, i.e., the turnaround delay of a single-word memory access in absence of bus interferences. $I^{\mathrm{bus}}(t, c, q, b)$ denotes the worst-case delay introduced due to interferences on bus b over which memory q is accessed. $I^{\mathrm{core}}(t, c, q, b)$ denotes the worst-case preemption delay imposed on t's execution on c. Here, $\mathrm{WCET}(t, c)$ and $\mathrm{MD}(t)$ are provided by the application, $\mathrm{ST}(q, b)$ is provided by the architecture, and $I^{\mathrm{bus}}(t, c, q, b)$ and $I^{\mathrm{core}}(t, c, q, b)$ are derived as follows.

5.1.1 Memory Bus Interference

Given that task t executed on core c accesses memory q over bus b, we bound the worst-case memory bus interference $I^{\mathrm{bus}}(t, c, q, b)$ based on the memory demand $\mathrm{MD}(t)$ of t, the service time $\mathrm{ST}(q, b)$ of memory q accessed over bus b, and the arbitration tuple (S_b, W_b^c, P_b^c) for core c on bus b. To this end, we first derive the maximum number of dedicated bus arbitration slots, denoted by $N(t, c, q, b)$, that t may require for performing $\mathrm{MD}(t)$ memory operations: On the one hand, $N(t, c, q, b)$ can never exceed $\mathrm{MD}(t)$, since each bus arbitration slot is necessarily long enough to allow performing at least one single-word memory access, i.e., $S_b \geq \mathrm{ST}(q, b)$. On the other hand, $N(t, c, q, b)$ can never exceed the maximum number of bus arbitration slots that may pass during t's execution when performing its memory accesses in absence of bus interferences. Thus, $N(t, c, q, b)$ can be bounded as:

$$N(t, c, q, b) = \min \left\{ \mathrm{MD}(t), \left\lceil \frac{\mathrm{WCET}(t, c) + \mathrm{MD}(t) \cdot \mathrm{ST}(q, b)}{S_b} \right\rceil \right\} \tag{2}$$

In the worst case, the memory accesses of t are distributed such that each of the $N(t, c, q, b)$ required bus slots falls into a separate bus arbitration period. In each bus arbitration period, a worst-case wait time of $(P_b^c - W_b^c \cdot S_b)$ may be imposed on t's execution before t acquires the bus ownership. Therefore, the overall bus interference can be bounded as:

$$I^{\mathrm{bus}}(t, c, q, b) = N(t, c, q, b) \cdot (P_b^c - W_b^c \cdot S_b) \tag{3}$$

In some cases, a memory access may be initiated so late that it cannot be completed before the end of the respective bus arbitration slot. To compensate for this misalignment, the arbitration delay D_b of each bus $b \in B$ is extended by the service time $\mathrm{ST}(q, b)$ of its memory q. This practice (a) allows late memory accesses to be completed before the next bus arbitration slot is assigned and, thereby, (b) eliminates memory interferences due to overlapping memory accesses from different bus masters in consecutive bus slots. Note that, if t accesses multiple memories, the analysis above must be applied for each memory bus b_i over which t performs $\mathrm{MD}_i(t)$ accesses to memory q_i with a service time of $\mathrm{ST}(q_i, b_i)$.

5.1.2 Core Preemption Delay

We bound the worst-case preemption delay of task t on core c based on its arbitration tuple on c, i.e., (S_c, W_c^t, P_c^t), and the arbitration tuple of c on the bus b over which c accesses memory q, i.e., (S_b, W_b^c, P_b^c). In each iteration of its execution, t requires an overall dedicated processor time of $(\mathrm{WCET}(t, c) + \mathrm{MD}(t) \cdot \mathrm{ST}(q) + I^{\mathrm{bus}}(t, c, q, b))$ on core c to complete its processing and perform its memory accesses over the shared bus. In each scheduling period of c, an overall processor time of $(W_c^t \cdot S_c)$ is dedicated to t and, hence, a worst-case periodic wait time of $(P_c^t - W_c^t \cdot S_c)$ is imposed on t's execution due to preemption. Thus, the overall preemption delay imposed on t's execution can be bounded as:

$$I^{\text{core}}(t, c, q, b) = \left\lceil \frac{\text{WCET}(t, c) + \text{MD}(t) \cdot \text{ST}(q) + I^{\text{bus}}(t, c, q, b)}{W_c^t \cdot S_c} \right\rceil \cdot \left(P_c^t - W_c^t \cdot S_c\right) \tag{4}$$

where the first factor derives the maximum number of scheduling periods over which the execution of t may span, and the second factor reflects the worst-case wait time imposed per scheduling period. Note that, also here, a memory access may be initiated so late that it cannot be completed before the end of the scheduling slot. To compensate for this misalignment, the context-switch latency D_c of each core $c \in C$ is extended by the maximum service time among all memories accessible to c to allow an initiated memory access to be completed before the following context switch on c.

5.2 Worst-Case Traversal Time

The WCTT of an inter-tile message m denotes its worst-case transfer latency from the memory in which m is stored on the source tile to the respective target memory on m's destination tile. This transfer is realized in three steps: (I) the TX tx on the source tile reads m from memory q over bus b, decomposes it into flits, and injects the flits into the NoC. (II) the flits are routed over m's NoC route ρ to the destination tile. Once arrived at the destination tile, (III) the RX rx reconstructs m and writes it into m's dedicated space in memory $\hat{q}$ over bus $\hat{b}$. Therefore, the WCTT of message m can be bounded as:

$$\text{WCTT}(m, tx, q, b, \rho, rx, \hat{q}, \hat{b}) = D^{\text{tx}}(m, tx, q, b) + D^{\text{noc}}(m, \rho) + D^{\text{rx}}(m, rx, \hat{q}, \hat{b}) \tag{5}$$

where $D^{\text{tx}}(m, tx, q, b)$, $D^{\text{noc}}(m, \rho)$, and $D^{\text{rx}}(m, rx, \hat{q}, \hat{b})$ respectively denote the worst-case latency of the transfer steps (I)–(III) above which we analyze in the following.

5.2.1 TX/RX Latency

Reading message m from memory q over bus b for injection into the NoC is subject to two sources of interference: (a) TX interference (as multiple outbound messages may be transmitted concurrently from the tile) and (b) bus interference (when TX accesses memory for reading m). We assume that, in each bus arbitration period, bus slots dedicated to TX tx are used for reading one message only, resulting in an equivalent tx slot length of one bus period, i.e., $S_{tx} = P_b^{tx}$. To bound the TX latency $D^{\text{tx}}(m, tx, q, b)$, we first derive the maximum number of bus slots, denoted as $N(m, q, b)$, that can be required in the worst case for reading m from memory q: In each bus arbitration slot, a total of $\lceil S_b/\text{ST}(q, b) \rceil$ consecutive memory accesses can be initiated and completed. Hence, given m's memory demand MD(m) (number of memory accesses required for reading m), $N(m, q, b)$ is calculated as:

$$N(m, q, b) = \left\lceil \text{MD}(m) \cdot \left\lceil \frac{S_b}{\text{ST}(q, b)} \right\rceil^{-1} \right\rceil \tag{6}$$

Given $N(m, q, b)$, we use Equation (7) to bound the TX latency based on the arbitration tuple of tx on bus b, i.e., $(S_b, W_b^{tx}, P_b^{tx})$, and the arbitration tuple of m on tx, i.e., $(S_{tx}, W_{tx}^m, P_{tx}^m)$ where $S_{tx} = P_b^{tx}$ as discussed above. The first summand in Equation (7) bounds the overall memory service time for the MD(m) memory accesses required for reading m from memory in absence of bus and TX interferences. The second summand calculates the worst-case bus interference imposed when reading m: Here, the first factor gives the number of bus arbitration periods that may pass before acquiring all $N(m,q,b)$ bus slots required for reading m, and the second factor gives the worst-case wait time imposed per bus period. Finally, the third summand in Equation (7) calculates the worst-case TX interference: Here, the first factor calculates the number of tx arbitration periods involved in the transfer of m, and the second factor gives the worst-case wait time per tx period. We use the same analysis to bound the RX latency, i.e., $D^{\text{rx}}(m, rx, \hat{q}, \hat{b})$ in Equation (5).

$$\begin{aligned} D^{\text{tx}}(m,tx,q,b) = \text{MD}(m)\cdot\text{ST}(q,b) \;+\; \left\lceil \frac{N(m,q,b)}{W_b^{tx}} \right\rceil \cdot \left(P_b^{tx} - W_b^{tx}\cdot S_b\right) \\ +\left\lceil \left\lceil \frac{N(m,q,b)}{W_b^{tx}} \right\rceil \cdot \frac{1}{W_{tx}^{m}} \right\rceil \cdot \left(P_{tx}^{m} - W_{tx}^{m}\cdot S_{tx}\right) \end{aligned} \tag{7}$$

5.2.2 NoC Latency

We bound the worst-case NoC routing latency of message m over route ρ based on the arbitration tuple of m on the links in ρ, i.e., $(S_\rho, W_\rho^m, P_\rho^m)$, using Equation (8) adopted from [48]. Here, f_m denotes the number of m's flits which is calculated based on its payload size PLD(m), $|\rho|$ gives the length of route ρ in number of hops, τ^{noc} denotes the length of one NoC clock cycle which also gives the length of one link arbitration slot ($S_\rho = \tau^{\text{noc}}$), and D^{router} gives the latency of a NoC router in clock cycles. The first summand in Equation (8) derives the transfer latency of f_m flits in absence of interferences on ρ. The second summand bounds the worst-case interferences on ρ: Here, the first factor gives the maximum number of link arbitration periods in which m's flits may be stalled due to interfering flows, and the second factor gives the worst-case wait time per link arbitration period, see [48].

$$D^{\text{noc}}(m,\rho) = (f_m - 1 + |\rho|\cdot D^{\text{router}})\cdot\tau^{\text{noc}} + \left(\left\lceil \frac{f_m}{W_\rho^m} \right\rceil - 1 + |\rho| \right)\cdot\left(P_\rho^m - W_\rho^m\cdot\tau^{\text{noc}}\right) \tag{8}$$

5.3 Worst-Case Throughput and Latency

Given the WCRT of each task $t \in T$, in short, WCRT(t), and the WCTT of each inter-tile message $m \in M$, in short, WCTT(m), the worst-case throughput of the mapping is calculated using Equation (9). Likewise, the worst-case application latency (makespan) is calculated using Equation (10) where Π denotes the set of all end-to-end paths in the application graph.

$$\mathcal{TH} = \max\left\{ \max_{t\in T}\{\text{WCRT}(t)\}, \max_{m\in M}\{\text{WCTT}(m)\} \right\}^{-1} \tag{9}$$

$$\mathcal{L} = \max_{\pi\in\Pi}\left\{ \sum\nolimits_{t\in(T\cap\pi)} \text{WCRT}(t) + \sum\nolimits_{m\in(M\cap\pi)} \text{WCTT}(m) \right\} \tag{10}$$

6 Experimental Results

This section presents the results of a series of experiments for a variety of applications and architectures to compare the performance of the proposed isolation-aware approach with existing fixed-isolation-scheme approaches w.r.t. the quality of delivered mappings.

6.1 Experiment Setup

Applications and Architectures. We use four real-time applications from the domains of networking (7 tasks, 9 messages), consumer (11 tasks, 12 messages), telecommunication (14 tasks, 20 messages), and automotive (18 tasks, 21 messages), provided by the Embedded

System Synthesis Benchmarks Suite (E3S) [11]. For target platform, we use three heterogeneous many-core architectures with 4×4, 5×5, and 6×6 tiles, respectively. Each architecture is composed of three tile types. Each tile comprises four homogeneous cores, a shared memory, a NA with separate TX and RX units, and a shared memory bus. Every shared resource has a WRR arbitration policy configured as follows: For each core, an arbitration capacity of $K = 10$, a slot length of $S = 50\,us$, and a context switch overhead of $D = 10\,us$ is considered. On each bus, each bus master (TX, RX, and four cores) has an arbitration weight of $W = 1$, resulting in a bus arbitration capacity of $K = 6$. The length of each bus slot S is set equal to the memory service time of 7 clock cycles. For NoC links, we consider an arbitration capacity of $K = 10$ and a slot length $S = \tau^{\text{noc}} = 10\,ns$. For each TX/RX, an arbitration capacity of $K = 10$ and a slot length equal to the bus arbitration period is considered, cf. Section 5.2.1.

Design Objectives. We use three design objectives which are commonly considered in the DSE of predictable and composable many-core systems: (I) *worst-case latency* derived using the proposed timing analysis, (II) *resource usage* calculated (in number of cores) as the sum of time slots reserved on each core divided by core arbitration capacity $K = 10$, and (III) *energy consumption*, calculated based on the processor power parameters provided by [11] for each investigated benchmark application and the NoC/bus energy model from [52], assuming a link length of $2\,mm$ and a bus length of $5\,mm$.

Design Space Exploration. To perform the DSE, we use the OpenDSE framework [38] and the NSGA-II [10] multi-objective evolutionary algorithm provided by the optimization framework Opt4J [30]. Each run of the DSE features 4,000 iterations with 25 mappings generated per iteration and a population size of 100 mappings. The results reported in this section are an average over 20 runs of the DSE for each application on each architecture.

Investigated Approaches. We compare the proposed isolation-aware DSE approach with the three major fixed-isolation-scheme DSE approaches, namely, core sharing, core reservation, and tile reservation. For each approach, the DSE delivers a set of mappings with Pareto-optimal trade-offs in the space of the three design objectives above. We compare the DSE approaches in terms of the quality of mappings they deliver.

Quality Metric. To compare the quality of mappings obtained by the investigated DSE approaches, we use the well-established *ϵ-dominance* metric [28] from the domain of multi-objective optimization, defined as follows: Let $F \subseteq \mathbb{R}^{+^N}$ represent a set of Pareto-optimal mappings $f \in F$ obtained by an optimization approach in the space of design objectives $o_1, ..., o_N$ to be minimized. Let $S \subseteq \mathbb{R}^{+^N}$ represent a reference set containing the true Pareto-optimal mappings for the optimization problem in question. To assess the quality of mappings in F w.r.t. S, ϵ-dominance provides a unary indicator $\epsilon_F \in [0, 1)$ calculated as:

$$\epsilon_F = \min\{0 \leq \epsilon < 1 \mid \forall s \in S : \exists f \in F \text{ s.t. } (1-\epsilon) \cdot f_{o_n} \leq s_{o_n} \; \forall n = 1, ..., N\} \tag{11}$$

where f_{o_n} and s_{o_n} denote the quality of mappings $f \in F$ and $s \in S$, respectively, w.r.t. design objective o_n. A *smaller* value for ϵ_F indicates a smaller distance between the mappings in F and those in the reference set S w.r.t. the design objectives which, in turn, denotes a *higher quality* of mappings in F. For our following experiments, the reference set S is constructed by collecting the mappings obtained by the four approaches under analysis in one set.

6.2 Result Discussion

Figure 8 illustrates, for each benchmark application on each many-core architecture, the ϵ-dominance indicator for the proposed approach and the three fixed-isolation-scheme approaches across their exploration run time. The plots in each row correspond to the same application, and the plots in each column correspond to the same architecture. In general, the quality of mappings collected by each approach improves as the exploration progresses, resulting in a decrease in the ϵ indicator of each approach throughout the course of its optimization iterations. Also, the exploration time of each approaches grows with the application size (compare the plots in one column) and architecture size (compare the plots in one row). The results illustrated in Figure 8 uniformly verify that (a) the proposed approach always outperforms the approaches with a fixed isolation scheme, irrespective of the choice of application and architecture, which is indicated by its lower ϵ-dominance indicator at the end of its exploration. Moreover, (b) the proposed approach exhibits an exploration run time within the same range as the other approaches and, hence, does not impact the scalability of the optimization approach adversely. Among the other approaches, core reservation generally performs better than core sharing and tile reservation for most of cases. Core sharing exhibits the worst quality of solutions for all applications and architectures. Considering all 12 combinations of applications and architectures, the proposed isolation-aware approach achieves an ϵ-dominance improvement of up to 67% (compared with the core-sharing approach) with an average improvement of 26% over the three fixed-isolation-scheme approaches.

To reason about these quality differences, we investigate the distribution of the mappings delivered by each DSE approach in the 3D objective space. Figure 9 illustrates this using two 2D projections of the objective space for two exemplary applications on the 6×6 architecture. Similar distributions are obtained for other applications and architectures. For each application (column), the x-axes in both projections denote resource usage while the y-axes denote worst-case latency (top) and energy consumption (bottom). As illustrated, solutions delivered by the core-sharing approach exhibit low resource usage, as this approach allocates a *minimal* resource budget, just enough to meet the deadlines of tasks/messages. This, however, implies a high worst-case inter-application interference and, thus, considerably high worst-case latency and energy consumption. The tile-reservation approach, on the other hand, allocates compute tiles exclusively (thus, higher resource usage) which eliminates all on-tile inter-application interferences (thus, lower latency and energy). When considering all objectives together, both of these approaches need a large scaling factor ϵ to compensate for their inferior objective values, explaining their poor (high) ϵ-dominance indices in Figure 8. A compromise between these approaches is achieved by core reservation which restricts the exclusive allocation of resources to cores, leading to a moderate trade-off between resource usage, latency, and energy consumption and, thus, a better (lower) ϵ-dominance index.

The fixed isolation scheme of the approaches above excludes large parts of the actual solution space and, thus, restricts their coverage of the objective space to a considerably smaller sub-region. Contrary to them, the proposed approach covers and extends beyond the solution space of these approaches as it explores the isolation schemes in combination within each mapping, also reflected by the wide spread of its solutions in Figure 9. As a result, it can find solutions of higher quality and, thus, outperforms the other approaches as confirmed by its lower ϵ-dominance index in Figure 8 for all investigated applications and architectures.

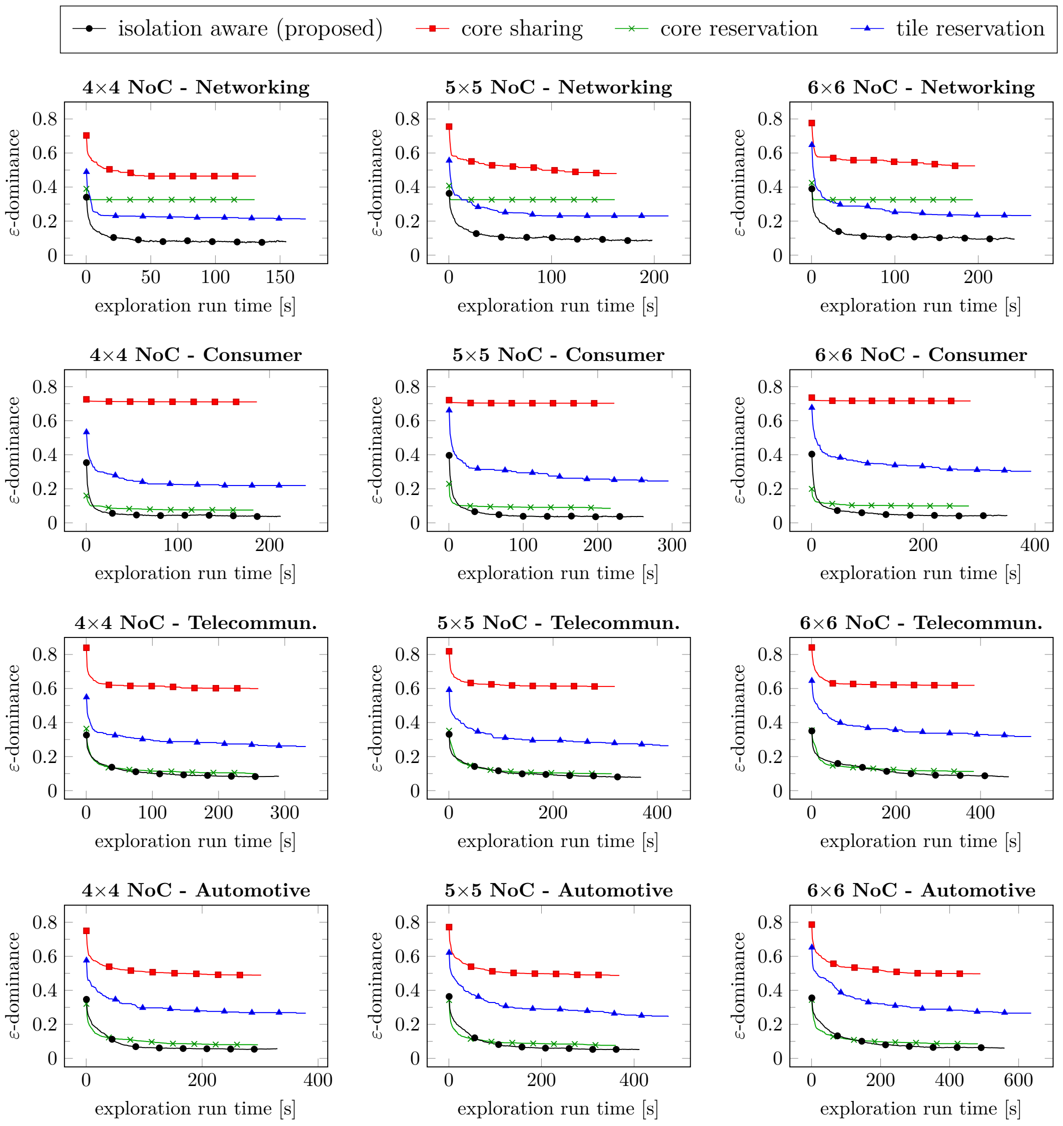

Figure 8 ϵ-dominance of the investigated DSE approaches versus their exploration run time. Plots in each column (row) correspond to the same many-core architecture (application). The proposed isolation-aware approach outperforms existing fixed-isolation-scheme approaches for all applications and architectures, denoted by its lower ϵ-dominance indicator.

7 Conclusion

Applications in composable many-core systems are typically developed with the assumption of a fixed inter-application isolation scheme which restricts the resource allocation policy of the applications and, therefore, their quality trade-off w.r.t. resource usage and worst-case timing. To lift this restriction, we have proposed (a) an *isolation-aware Design Space Exploration (DSE)* which explores the isolation schemes per allocated core/tile within each mapping and (b) an *isolation-aware timing analysis* to formally bound the worst-case timing properties of each explored mapping. For a variety of hard real-time applications and many-core architectures, we have experimentally demonstrated the advantage of the proposed approach over existing fixed-isolation-scheme approaches w.r.t. the quality of the delivered solutions in terms of resource usage, worst-case latency, and energy consumption.

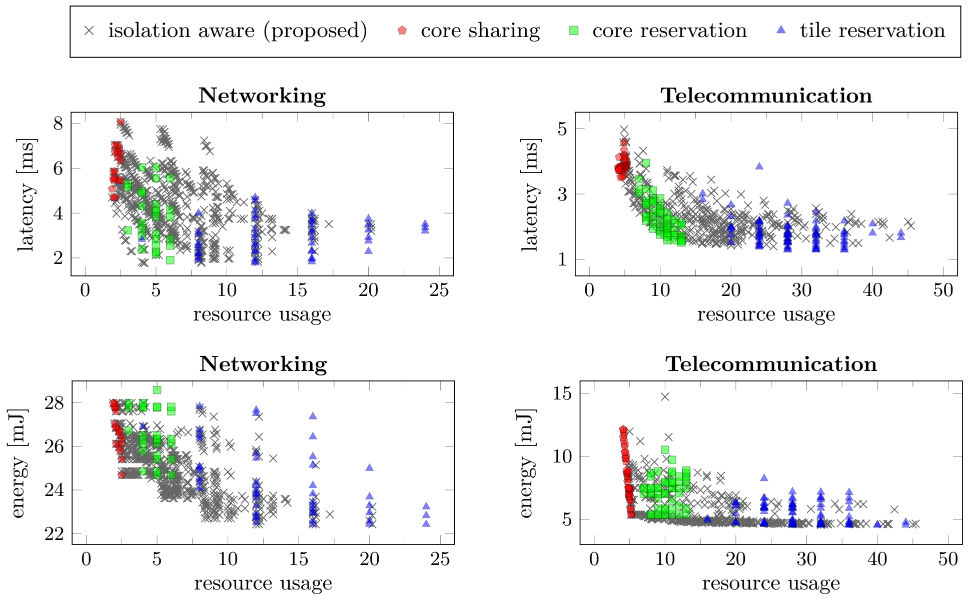

Figure 9 2D projections of the 3D objective space showing the spread of solutions delivered by each DSE approach for two exemplary applications (columns) on the 6×6 many-core architecture.

References

1 Benny Akesson, Anca Molnos, Andreas Hansson, Jude Ambrose Angelo, and Kees Goossens. Composability and predictability for independent application development, verification, and execution. In *Multiprocessor System-on-Chip*, pages 25–56. Springer, 2011.

2 Sebastian Altmeyer, Robert I Davis, Leandro Indrusiak, Claire Maiza, Vincent Nelis, and Jan Reineke. A generic and compositional framework for multicore response time analysis. In *Proceedings of the Conference on Real Time Networks and Systems (RTNS)*, pages 129–138. ACM, 2015.

3 Tobias Blickle, Jürgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *Design Automation for Embedded Systems*, 3(1):23–58, 1998.

4 Tilera Corporation. Tile Processor Architecture Overview for the TILE-Gx Series, 2012.

5 William J Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed systems*, 3(2):194–205, 1992.

6 Dakshina Dasari, Vincent Nelis, and Benny Akesson. A framework for memory contention analysis in multi-core platforms. *Real-Time Systems*, 52(3):272–322, 2016.

7 Lawrence Davis. *Handbook of genetic algorithms.* VNR computer library. Van Nostrand Reinhold, 1991.

8 Robert I Davis, Sebastian Altmeyer, Leandro S Indrusiak, Claire Maiza, Vincent Nelis, and Jan Reineke. An extensible framework for multicore response time analysis. *Real-Time Systems*, pages 1–55, 2017.

9 Benoît Dupont de Dinechin, Renaud Ayrignac, Pierre-Edouard Beaucamps, Patrice Couvert, Benoit Ganne, Pierre Guironnet de Massas, François Jacquet, Samuel Jones, Nicolas Morey Chaisemartin, Frédéric Riss, et al. A clustered manycore processor architecture for embedded and accelerated applications. In *Proceedings of High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2013.

10 Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

11 Robert Dick. Embedded system synthesis benchmarks suite (E3S), 2010. URL: `http://ziyang.eecs.umich.edu/~dickrp/e3sdd/`.

12 Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Proceedings of the International Conference on Genetic Algorithms*, pages 416–423. Morgan Kaufmann Publishers Inc., 1993.

13 Carlos M Fonseca and Peter J Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1):1–16, 1995.

14 Michael R Garey and David S Johnson. A Guide to the Theory of NP-Completeness. *Computers and Intractability*, pages 37–79, 1990.

15 Georgia Giannopoulou, Kai Lampka, Nikolay Stoimenov, and Lothar Thiele. Timed model checking with abstractions: Towards worst-case response time analysis in resource-sharing manycore systems. In *Proceedings of the International Conference on Embedded Software*, pages 63–72. ACM, 2012.

16 Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, and Lothar Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Proceedings of the International Conference on Embedded Software*, page 17. ACM, 2013.

17 Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, Lothar Thiele, and Benoît Dupont de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, 52(4):399–449, 2016.

18 Michael Glaß, Jürgen Teich, Martin Lukasiewycz, and Felix Reimann. Hybrid optimization techniques for system-level design space exploration. In *Handbook of Hardware/Software Codesign*, volume 1, pages 217–246. Springer, 2017.

19 Kees Goossens, Arnaldo Azevedo, Karthik Chandrasekar, Manil Dev Gomony, Sven Goossens, Martijn Koedam, Yonghui Li, Davit Mirzoyan, Anca Molnos, Ashkan Beyranvand Nejad, et al. Virtual execution platforms for mixed-time-criticality systems: the CompSOC architecture and design flow. *ACM SIGBED Review*, 10(3):23–34, 2013.

20 Sebastian Hahn, Jan Reineke, and Reinhard Wilhelm. Towards compositionality in execution time analysis: definition and challenges. *ACM SIGBED Review*, 12(1):28–36, 2015.

21 Andreas Hansson, Kees Goossens, Marco Bekooij, and Jos Huisken. CoMPSoC: A template for composable and predictable multi-processor system on chips. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1):2, 2009.

22 Jan Heisswolf, Ralf König, Martin Kupper, and Jürgen Becker. Providing multiple hard latency and throughput guarantees for packet switching networks on chip. *Computers & Electrical Engineering*, 39(8):2603–2622, 2013.

23 Jason Howard, Saurabh Dighe, Yatin Hoskote, Sriram Vangal, David Finan, Gregory Ruhl, David Jenkins, Howard Wilson, Nitin Borkar, Gerhard Schrom, et al. A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 108–109. IEEE, 2010.

24 Timon Kelter, Tim Harde, Peter Marwedel, and Heiko Falk. Evaluation of resource arbitration methods for multi-core real-time systems. In *Proceedings of the workshop on Worst-Case Execution Time Analysis (WCET)*. Schloss Dagstuhl-Leibniz-Zentrum fr̈ Informatik, 2013.

25 James Kennedy. Particle swarm optimization. *Encyclopedia of Machine Learning*, pages 760–766, 2010.

26 Pham Nam Khanh, Amit Kumar Singh, Akash Kumar, and Khin Mi Mi Aung. Incorporating energy and throughput awareness in design space exploration and run-time mapping for heterogeneous MPSoCs. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 513–521. IEEE, 2013.

27 Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

28 Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

29 Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich. SAT-decoding in evolutionary algorithms for discrete constrained optimization problems. In *Proceedings of the IEEE Congress onEvolutionary Computation*, pages 935–942. IEEE, 2007.

30 Martin Lukasiewycz, Michael Glaß, Felix Reimann, and Jürgen Teich. Opt4J: a modular framework for meta-heuristic optimization. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1723–1730. ACM, 2011.

31 Martin Lukasiewycz, Shanker Shreejith, and Suhaib A Fahmy. System simulation and optimization using reconfigurable hardware. In *International Symposium on Integrated Circuits (ISIC)*, pages 468–471, 2014.

32 Guilherme Madalozzo, Liana Duenha, Rodolfo Azevedo, and Fernando G Moraes. Scalability evaluation in many-core systems due to the memory organization. In *Proceedings of the International Conference on Electronics, Circuits and Systems (ICECS)*, pages 396–399. IEEE, 2016.

33 Giovanni Mariani, Prabhat Avasare, Geert Vanmeerbeeck, Chantal Ykman-Couvreur, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 196–201, 2010.

34 Tulika Mitra, Jürgen Teich, and Lothar Thiele. Time-Critical Systems Design: A Survey. *IEEE Design & Test*, 35(2):8–26, 2018.

35 Lionel M Ni and Philip K McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

36 Roberta Piscitelli and Andy D Pimentel. Design space pruning through hybrid analysis in system-level design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 781–786. IEEE, 2012.

37 Behnaz Pourmohseni, Stefan Wildermann, Michael Glaß, and Jürgen Teich. Hard real-time application mapping reconfiguration for NoC-based many-core systems. *Real-Time Systems*, pages 1–37, 2019.

38 Felix Reimann, Martin Lukasiewycz, Michael Glaß, and Fedor Smirnov. OpenDSE – open design space exploration framework, 2018. URL: `http://opendse.sourceforge.net/`.

39 Jan Reineke, Björn Wachter, Stefan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A definition and classification of timing anomalies. In *Proceedings of the International Workshop on Worst-Case Execution Time Analysis (WCET)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

40 Hamza Rihani, Matthieu Moy, Claire Maiza, Robert I Davis, and Sebastian Altmeyer. Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor. In *Proceedings of the conference on Real-Time Networks and Systems (RTNS)*, pages 67–76. ACM, 2016.

41 Benjamin Rouxel, Steven Derrien, and Isabelle Puaut. Tightening contention delays while scheduling parallel applications on multi-core architectures. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):164, 2017.

42 Pradip Kumar Sahu, Tapan Shah, Kanchan Manna, and Santanu Chattopadhyay. Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2):300–312, 2014.

43 Z. Shi and A. Burns. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *International Symposium on Networks-on-Chip (NOCS)*, pages 161–170, 2008.

44 Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs. *ACM Transaction on Design Automation of Electronic Systems (TODAES)*, 18(1):9:1–9:29, 2013.

45 Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–10, 2013.

46 Stefanos Skalistis and Alena Simalatsar. Worst-case execution time analysis for many-core architectures with NoC. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 211–227. Springer, 2016.

47 Stefanos Skalistis and Alena Simalatsar. Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 752–757. IEEE, 2017.

48 Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, and Jürgen Teich. DAARM: Design-time application analysis and run-time mapping for predictable execution in many-core systems. In *Proceedings of the International Conference on Hardware/-Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, 2014.

49 Andreas Weichslgartner, Stefan Wildermann, Deepak Gangadharan, Michael Glaß, and Jürgen Teich. A design-time/run-time application mapping methodology for predictable execution time in MPSoCs. *ACM Transactions on Embedded Computing Systems (TECS)*, 2018.

50 Andreas Weichslgartner, Stefan Wildermann, Michael Glaß, and Jürgen Teich. *Invasive Computing for mapping parallel programs to many-core architectures*. Springer, 2018.

51 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. The worst-case execution-time problem-overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008.

52 Pascal T Wolkotte, Gerard JM Smit, Nikolay Kavaldjiev, Jens E Becker, and Jürgen Becker. Energy model of networks-on-chip and a bus. In *Proceedings of the International Symposium on System-on-Chip (SoC)*, pages 82–85, 2005.

53 Chantal Ykman-Couvreur, Prabhat Avasare, Giovanni Mariani, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *IET Computers and Digital Techniques*, 5(2):123–135, 2011.

54 Jia Zhan, Nikolay Stoimenov, Jin Ouyang, Lothar Thiele, Vijaykrishnan Narayanan, and Yuan Xie. Designing energy-efficient NoC for real-time embedded systems through slack optimization. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6. IEEE, 2013.