Journal of
Low Power Electronics and Applications

Article

Hybrid Application Mapping for Composable Many-Core Systems: Overview and Future Perspective

Behnaz Pourmohseni [1,*], Michael Glaß [2], Jörg Henkel [3], Heba Khdr [3], Martin Rapp [3], Valentina Richthammer [2], Tobias Schwarzer [1], Fedor Smirnov [4], Jan Spieck [1], Jürgen Teich [1], Andreas Weichslgartner [5] and Stefan Wildermann [1]

1. Chair for Hardware/Software Co-Design, Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 91058 Erlangen, Germany; tobias.schwarzer@fau.de (T.S.); jan.spieck@fau.de (J.S.); juergen.teich@fau.de (J.T.); stefan.wildermann@fau.de (S.W.)
2. Institute of Embedded Systems/Real-Time Systems, Faculty of Engineering, Computer Science and Psychology, Ulm University, 89081 Ulm, Germany; michael.glass@uni-ulm.de (M.G.); valentina.richthammer@uni-ulm.de (V.R.)
3. Chair for Embedded Systems, Department of Computer Science, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany; henkel@kit.edu (J.H.); heba.khdr@kit.edu (H.K.); martin.rapp@kit.edu (M.R.)
4. Distributed and Parallel Systems Group, Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria; fedor@dps.uibk.ac.at
5. AUDI AG, 85045 Ingolstadt, Germany; andreas.weichslgartner@audi.de

* Correspondence: behnaz.pourmohseni@fau.de

Received: 25 September 2020; Accepted: 13 November 2020; Published: 17 November 2020

Abstract: Many-core platforms are rapidly expanding in various embedded areas as they provide the scalable computational power required to meet the ever-growing performance demands of embedded applications and systems. However, the huge design space of possible task mappings, the unpredictable workload dynamism, and the numerous non-functional requirements of applications in terms of timing, reliability, safety, and so forth. impose significant challenges when designing many-core systems. Hybrid Application Mapping (HAM) is an emerging class of design methodologies for many-core systems which address these challenges via an incremental (per-application) mapping scheme: The mapping process is divided into (i) a design-time Design Space Exploration (DSE) step per application to obtain a set of high-quality mapping options and (ii) a run-time system management step in which applications are launched dynamically (on demand) using the precomputed mappings. This paper provides an overview of HAM and the design methodologies developed in line with it. We introduce the basics of HAM and elaborate on the way it addresses the major challenges of application mapping in many-core systems. We provide an overview of the main challenges encountered when employing HAM and survey a collection of state-of-the-art techniques and methodologies proposed to address these challenges. We finally present an overview of open topics and challenges in HAM, provide a summary of emerging trends for addressing them particularly using machine learning, and outline possible future directions. While there exists a large body of HAM methodologies, the techniques studied in this paper are developed, to a large extent, within the scope of invasive computing. Invasive computing introduces resource awareness into applications and employs explicit resource reservation to enable incremental application mapping and dynamic system management.

Keywords: Hybrid Application Mapping (HAM); many-core systems; embedded systems; composability; design space exploration (DSE); resource management; Network-on-Chip (NoC); real-time guarantees; predictability; machine learning

1. Introduction

The ever-increasing computational power requirements of embedded applications have substantially changed the design process of embedded systems over the past decade. To address the performance demands of emerging applications, embedded domains have undergone a paradigm shift from single-core platforms to many-core platforms. Many-core platforms such as Tilera TILE-Gx [1], Kalray MPPA-256 [2], Intel SCC (Single-chip Cloud Computer) [3], the KiloCore [4], or the upcoming SiPearl Rhea processor family [5] integrate tens, hundreds, or even thousands of processing cores on a single chip with a highly scalable communication scheme. This enables them to deliver a scalable computational power which is required to meet the progressively growing performance demands of emerging embedded applications and systems. Along the same line, modern platforms also incorporate heterogeneous processing resources to cater to the specific functional and non-functional requirements of applications from different domains of computing, see, for example, Reference [6]. In addition to the current practice of integrating various types of general-purpose cores on a chip, many-core platforms are also on the verge of incorporating domain/application-specific processing resources, for example, Digital Signal Processor (DSP) cores for signal/image processing [7], Graphics Processing Units (GPUs) for graphics processing and AI acceleration in deep learning [8,9], and Coarse-Grained Reconfigurable Arrays (CGRA) for the acceleration of (nested-)loops [10,11]. Moreover, Field Programmable Gate Arrays (FPGAs) have also been incorporated to provide a reconfigurable fabric for hardware acceleration [9,12]. While a large number of (possibly heterogeneous) processing resources increases the available computational power dramatically, it also introduces a significant number of additional design decisions to be made during the phases of system design as well as application mapping. In the following, an overview of the major challenges of many-core application mapping is provided.

1.1. Many-Core Application Mapping Challenges

1.1.1. Task Mapping Complexity

To exploit the massive computational power of many-core platforms, parallel computation models have been increasingly adopted in the development of embedded applications [13]. In such models, each application is partitioned into several (processing) tasks that can be executed concurrently on different cores and exchange data with each other. In this context, the optimization of the application mapping, that is, finding a suitable assignment of an application's tasks to platform resources, becomes a particularly challenging effort. This is due to the number of possible mappings for an application growing exponentially with the application size (number of application's tasks) and the platform size (number of cores). In fact, the many-core application mapping problem is known to be an *NP-hard* [14,15] optimization problem which renders an enumeration of all possible mappings for realistic problem sizes impractical, if at all feasible. As a consequence, system designers resort to meta-heuristic optimization algorithms, for example, evolutionary algorithms [16–18] and particle swarm optimization [19], for the automated Design Space Exploration (DSE) of possible application mappings. Meta-heuristic optimization algorithms have proven effective in finding satisfactory mappings at an affordable computational effort.

1.1.2. Complexity of Evaluation and Verification Techniques

In addition to functional correctness, embedded applications typically also need to satisfy a set of non-functional requirements, often provided in the form of upper/lower bound constraints on timing, reliability, security, safety, and other qualities [20,21]. For each mapping of an application, the satisfaction of the application's non-functional requirements must be verified, for example, by means of measurement, simulation, or formal analysis. Subject to the characteristics of the application and the platform resources, the choice of non-functional requirements, and the strictness of their constraints, the verification process may become fairly complex and/or demand a considerable amount of computational effort and time.

1.1.3. Workload Dynamism

Another factor contributing significantly to the design complexity of modern embedded systems is the growing dynamism of workload. In these systems, a mix of applications—each with its own set of non-functional requirements—must typically be executed concurrently. Recent years give evidence of a rapid increase in the number of concurrent applications in embedded systems with different requirements. In these systems, the system's *workload scenario*, that is, the mix of concurrently executed applications (also known as the system's *use-case* [22]), tends to change over time such that, at each point in time, only a fraction of all applications in the system are active. These workload variations, including the activation and the termination of applications, often happen in reaction to external events whose arrival pattern cannot be predicted, for example, user requests or changes in the environment with which the system is interacting. In general, the number of system workload scenarios, that is, possible mixes of concurrently active applications, grows exponentially with the number of applications in the system [23]. The increasing trends in (i) the number of applications in the system and in (ii) the dynamic workload of the applications each contribute exponentially to the complexity of the process of finding optimal mappings of the applications to system resources [24]. To alleviate the design complexity w.r.t. the increased number of applications, the *integrated design* approach, where the mappings of all applications are considered at the same time, has been gradually replaced by an *incremental (constructive) design* approach in which the mapping process is partitioned into a phase with a per-application mapping optimization step followed by a system integration step.

1.2. Hybrid Application Mapping

Application mapping methodologies for multi/many-core systems are generally classified into two categories, namely, design-time (static) and run-time (dynamic) approaches, see Reference [25]. In this paper, in the context of application mapping, the terms *static*, *offline*, and *design-time* are used interchangeably to denote that the operation in question is performed at design time. Likewise, the terms *dynamic*, *online*, and *run-time* are used interchangeably to denote that the operation in question takes place at run time.

In *design-time (static) mapping approaches*, all mapping decisions are conducted statically at design time (offline). These approaches employ compute-intensive optimization and verification techniques to find an optimal mapping for each application which is also verified to satisfy the application's requirements. Since each system design generated by static approaches is tailored to a single scenario, these approaches either cannot at all be used for the design of dynamic systems or have to resort to single solutions compromising between different expected run-time scenarios.

In the second class of mapping approaches, namely, *run-time (dynamic) mapping approaches*, all mapping decisions are made dynamically at run time (online) when an application must be launched. These approaches take into account the current system workload in their mapping decisions. This offers an adaptive solution for the design of dynamic systems and eliminates the need to statically compromise between different workload scenarios. This advantage, however, comes at the expense of increased time pressure, since the time overhead of the application mapping process has a direct impact on the system's performance. Due to this time pressure, run-time mapping methodologies cannot afford powerful mapping optimization and non-functional verification procedures. Instead, they are limited to lightweight (incremental and/or iterative) mapping heuristics to find an acceptable mapping at a low computational effort, see, for example, Reference [26]. Consequently, they mostly yield sub-optimal mappings and will often not strictly provide non-functional guarantees which require compute-intensive verification processes, for example, reliability analysis or worst-case timing verification.

Hybrid Application Mapping (HAM) is a new class of mapping approaches which addresses the aforementioned many-core application mapping challenges (discussed in Section 1.1) by combining static and dynamic mapping approaches to exploit the individual strengths of each, see Reference [25]. In HAM, a set of Pareto-optimal mappings is computed for each application at design time where compute-intensive mapping optimization and non-functional verification techniques are affordable. These mappings are then used at run time to launch the application on demand by selecting one of the precomputed mappings which fits best to the current system workload state and resource availability. Combining (Pareto-)optimal mappings with guaranteed non-functional properties while coping with the workload dynamism, HAM is regarded as a promising paradigm for the design of future embedded systems. In this paper, we provide an overview of HAM and the design methodologies developed in line with it.

Since the introduction of many-core platforms in embedded domains, numerous proposals for application mapping on these platforms have been registered, addressing a broad range of application mapping challenges. In the same line, new programming paradigms have emerged to enable a systematic design approach for the incremental mapping of applications to embedded many-core systems. *Invasive computing* [6] is an emerging many-core programming paradigm in which *resource awareness* is introduced into application programming, and dynamic per-application *resource reservation* policies are employed to achieve not only a high utilization of resources but also providing isolation between resources and applications on demand in order to create predictability in terms of timing, safety, or security [27,28]. This setup is particularly promising for HAM and has served as the base for a large number of works in the scope of HAM.

1.3. Paper Overview and Organization

1.3.1. Paper Overview

In this paper, we introduce the fundamentals of HAM and elaborate on the way HAM addresses the major design challenges in mapping applications to many-core systems. The fusion of offline mapping optimization and online mapping selection in HAM, however, also gives rise to new challenges that must be addressed to boost its effectiveness. In this paper, we also provide an overview of the main challenges encountered when employing HAM and survey a collection of state-of-the-art techniques and methodologies proposed to address these challenges. We also discuss a collection of open topics and challenges in HAM, present a summary of emerging trends for addressing them particularly using machine learning, and outline some promising future directions. The majority of the techniques studied in this paper are developed within the scope of invasive computing which serves as an enabler for HAM and incremental design. An early overview of HAM techniques can, for example, be found in Reference [25].

1.3.2. Paper Organization

The remainder of this paper is organized as follows. In Section 2, a review of static and dynamic application mapping schemes—which can be considered HAM predecessors—is given and schemes for incremental design are presented. Section 3 provides an overview of the application and architecture models commonly used to describe the application mapping problem in embedded many-core systems. In Section 4, the basics of HAM are presented. Sections 5 and 6 present an overview of state-of-the-art methodologies and techniques in HAM—Techniques discussed in Section 5 aim at reducing the complexity of HAM at different stages of design while the approaches presented in Section 6 focus on enabling HAM for real-time systems as a predominant class of embedded systems. Section 7 presents open topics and challenges in HAM, discusses a collection of emerging trends for addressing them particularly using machine learning, and outlines promising future directions. The paper is concluded in Section 8.

2. Related Work

Design methodologies for many-core systems generally deal with the problem of mapping multiple applications to the resources of a many-core platform. In addition to functional correctness, a high-quality mapping must also satisfy the non-functional requirements of the application, for example, timing, reliability, and security, while exhibiting a high performance w.r.t., for example, resource utilization and energy efficiency. Prior to the introduction of HAM, application mapping methodologies for embedded systems were typically classified into two categories: *design-time (static) approaches* and *run-time (dynamic) approaches*. An elaborate survey of these techniques is presented in Reference [25].

The majority of existing application mapping methodologies fall into the category of static approaches, see, for example, References [29–33]. In these approaches, all mapping decisions are made offline. These approaches rely on a global view of the whole system, and in particular, the complete set of applications in the system, and exploit this knowledge to find an optimal mapping of all applications in the system to platform resources [25]. Due to their offline scheme, they can afford compute-intensive mapping optimization and non-functional verification techniques which are often inevitable, for example, in the case of applications with hard real-time constraints. Given the NP-hard nature of the many-core application mapping optimization problem [14,15], static approaches employ DSE techniques based on *meta-heuristic optimization* approaches, for example, evolutionary algorithms [16–18], simulated annealing [34], or particle swarm optimization [19], to find high-quality mapping solutions with a reasonable computational effort. For instance, for the DSE in References [15,35], genetic algorithms are used. In Reference [36], simulated annealing is used, while Reference [37] adopts particle swarm optimization. In spite of its advantages, this static design scheme is practical only for systems with a relatively static workload profile and is, thus, impractical for systems with dynamic workload scenarios or systems in which the complete set of applications is not known statically [25,38].

Dynamic mapping approaches offer a flexible and adaptive application mapping scheme that can be used for the design of systems with dynamic workload scenarios (use cases). In these approaches, mapping decisions are conducted online at the time an application must be launched, see, for example, References [26,39,40]. In spite of their flexibility, the processing power available for the online decision processes is limited to the resources of the underlying platform, and the time for their decision making is restricted by application-specific deadlines. Therefore, dynamic approaches cannot afford compute-intensive techniques for their mapping decisions and, hence, typically yield application mappings of inferior quality, compared to static approaches.

Hybrid Application Mapping (HAM) is an emerging category of mapping methodologies which employs a combination of offline mapping optimization and online mapping selection to address the shortcomings of both static and dynamic mapping methodologies [25]. A large body of work exists on HAM, for example, References [23,41–47]. At design time, HAM approaches employ DSE to compute a set of high-quality mappings for each application. Similarly to static approaches, the offline DSE in HAM benefits from compute-intensive optimization and verification techniques, for example, for worst-case timing verification. By using statically verified mappings as candidates for online application mapping, HAM enables the dynamic mapping of a broad range of applications for which the verification of non-functional requirements involves time-consuming analyses and, hence, cannot be done online. An overview of a collection of state-of-the-art HAM techniques is given in Sections 5 and 6.

The majority of many-core design methodologies, including those listed above, follow an *incremental design* approach consisting of a per-application mapping computation step and a subsequent system integration step. An incremental design, verification, and integration approach alleviates the design complexity significantly [24]. To enable this design scheme, *system composability* is essential. Composability is a system property ensuring that the non-functional behavior of each application in the system is not affected by other applications [24,48]. For instance, in a *timing-composable* system, for example, CoMPSoC [49] or T-CREST [50], concurrent applications are decoupled from each other w.r.t. their (worst-case) timing behavior. This allows timing verification of each application to be performed individually and irrespectively of the other applications.

Composability can be established using temporal/spatial isolation between applications [24]. In the case of *spatial isolation*, resources are exclusively reserved for applications. Spatial isolation has been used to eliminate inter-application timing interferences for real-time applications [51] and side-channel attacks for security-critical applications [52]. In the case of *temporal isolation*, resource sharing among applications is allowed under exclusive resource budget reservation per application and a timing-composable arbitration/scheduling policy. Examples of such policies include Time Division Multiple Access (TDMA) used in References [49,50] or Weighted Round Robin (WRR) used in References [41,51,53]. In the same line, new programming paradigms have emerged which promote the isolation of applications in favor of composability to enable an incremental design scheme. For instance, in the paradigm of invasive computing [6,28], application programs can exclusively allocate (invade) resources and later release them again (retreat). Invasion establishes spatial isolation between concurrently executed applications to achieve timing composability by means of explicit resource reservation per application, see, for example, Reference [27]. Such support is particularly crucial for HAM which relies on separate mapping optimization of individual applications at design time. The HAM techniques discussed in this paper (in Sections 5 and 6) are developed based on these principles.

3. System Model

Most approaches for design automation require a formal system model which serves as a basis for the optimization and verification processes performed throughout the DSE. The design problem of heterogeneous embedded systems, including the many-core application mapping problem, is typically represented using a graph-based system model, referred to as a *specification* which consists of (i) an *application graph* representing the application, (ii) an *architecture graph* representing the target many-core platform architecture and (iii) a set of *mapping edges* which connect these two graphs to reflect the task-to-core assignment options, see References [15,21]. This section provides an overview of the application and architecture models commonly used in the embedded many-core domain and demonstrates how these models can be converted into the graph-based specification.

3.1. Application Model

In the parallel-processing paradigm of multi/many-core systems, an application is typically partitioned into a set of (processing) tasks which communicate with each other via a set of messages [13]. To reflect this structure, each application is modeled by an acyclic, directed, and bipartite graph $G_P(T \cup M, E)$ called the *application graph* (also known as task graph or problem graph) [27]. Here, T denotes the set of tasks, M denotes the set of messages exchanged between the tasks, and $E \subseteq (T \times M) \cup (M \times T)$

is a set of directed edges which specify the data dependencies among tasks and messages. Figure 1a illustrates an exemplary application graph where $T = \{t_0, \ldots, t_5\}$ and $M = \{m_0, \ldots, m_5\}$.

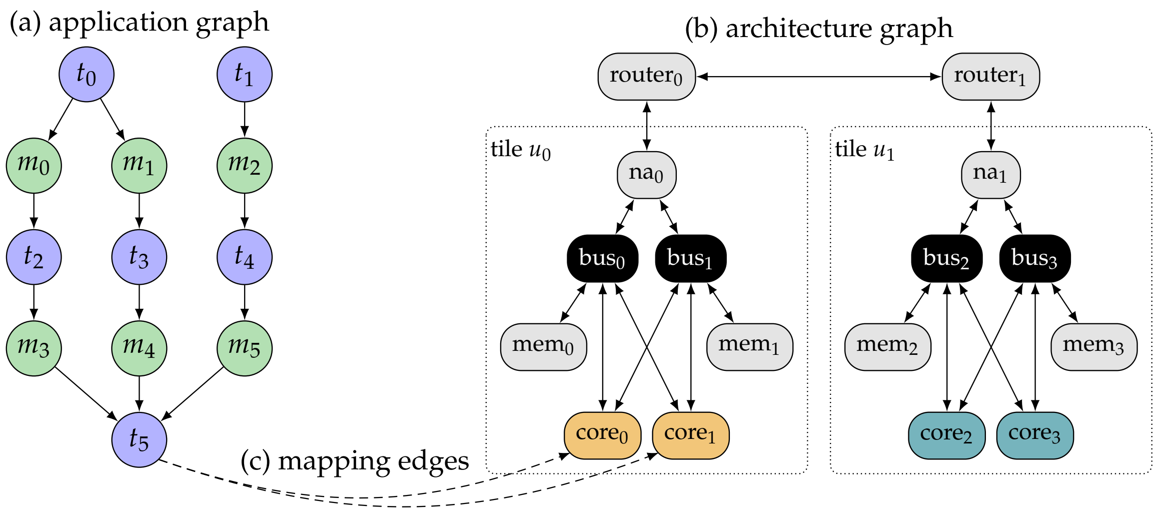


Figure 1. Example of a specification composed of (**a**) application graph, (**b**) architecture graph, and (**c**) mapping edges connecting them (depicted only for task t_5).

Tasks and messages in an application graph may also be annotated with additional information that might be required for the generation and/or the evaluation of the mappings. For instance, in the case of periodic applications, the execution period of each task and the production period of each message is provided to be used for resource scheduling. Similarly, in the case of real-time applications, the Worst-Case Execution Time (WCET) of each task and the maximum number of its memory operations may be provided to be used in the latency analysis and timing verification of the generated mappings. Also, the maximum payload size of a message and the maximum number of memory operations necessary to read/write the message from/to memory might be provided.

3.2. Architecture Model

Many-core platforms, for example, References [1–3], typically follow a regular and two-dimensional organization of resources in which resources are partitioned into a set of so-called *(compute) tiles*, interconnected by a two-dimensional Network-on-Chip (NoC) with a mesh topology, see, for example, Figure 2. Each tile in a many-core platform consists of a set of processing cores, one or multiple shared memories, and a Network Adapter (NA) which connects the on-tile resources to the routers of the NoC. The resources on each tile are interconnected via one or multiple (memory) buses. Each core may have a private L1 cache, and each tile may have an L2 cache shared among the resources located on that tile. In a heterogeneous many-core platform, the cores within each tile are typically from the same type (homogeneous) while different tiles comprise cores of different types. In addition to the compute tiles, a many-core platform may also contain one or multiple memory tiles to provide mass storage on the chip. Moreover, a set of I/O tiles may be available for connectivity with off-chip media, for example, cameras or sensors.

Many-core platforms employ a NoC interconnection infrastructure for inter-tile connectivity, chiefly due to its scalability [54]. A NoC consists of a set of routers and NAs which are interconnected via a set of links. Each router is associated with one tile of the platform and is connected to the NA located on that tile. In a mesh NoC, each router is also connected to its adjacent routers in the four cardinal directions. Each NoC connection enables the exchange of data in both directions using two uni-directional links.

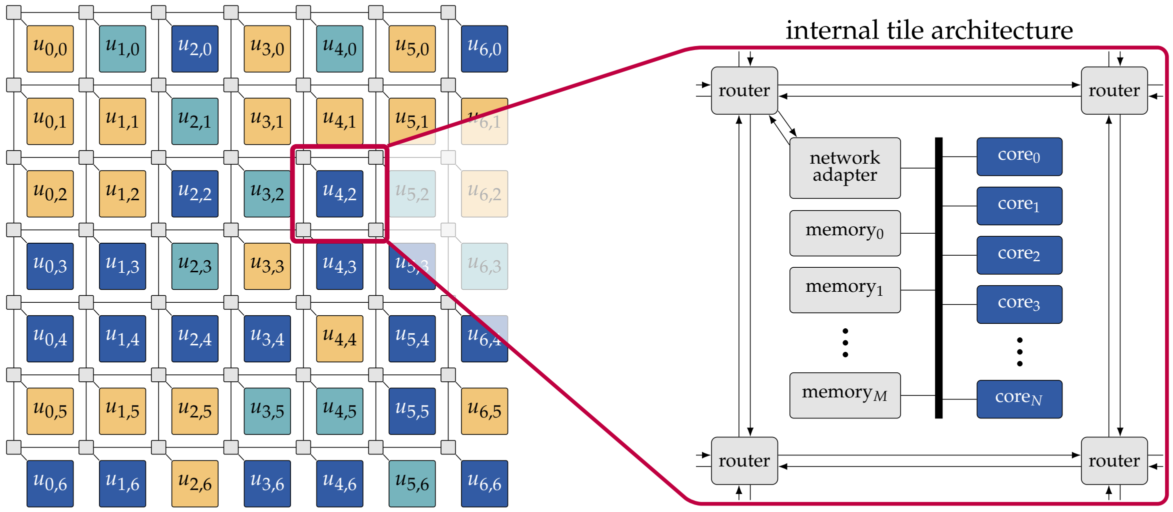

$u_{0,0}$ $u_{1,0}$ $u_{2,0}$ $u_{3,0}$ $u_{4,0}$ $u_{5,0}$ $u_{6,0}$
$u_{0,1}$ $u_{1,1}$ $u_{2,1}$ $u_{3,1}$ $u_{4,1}$ $u_{5,1}$ $u_{6,1}$
$u_{0,2}$ $u_{1,2}$ $u_{2,2}$ $u_{3,2}$ $u_{4,2}$ $u_{5,2}$ $u_{6,2}$
$u_{0,3}$ $u_{1,3}$ $u_{2,3}$ $u_{3,3}$ $u_{4,3}$ $u_{5,3}$ $u_{6,3}$
$u_{0,4}$ $u_{1,4}$ $u_{2,4}$ $u_{3,4}$ $u_{4,4}$ $u_{5,4}$ $u_{6,4}$
$u_{0,5}$ $u_{1,5}$ $u_{2,5}$ $u_{3,5}$ $u_{4,5}$ $u_{5,5}$ $u_{6,5}$
$u_{0,6}$ $u_{1,6}$ $u_{2,6}$ $u_{3,6}$ $u_{4,6}$ $u_{5,6}$ $u_{6,6}$
internal tile architecture
router
router
network adapter
core$_0$
core$_1$
memory$_0$
core$_2$
memory$_1$
core$_3$
memory$_M$
core$_N$
router
router

Figure 2. Example of a 7×7 heterogeneous tiled many-core architecture. Tiles are interconnected by a 2D mesh Network-on-Chip (NoC). Each tile consists of a set of processing cores, a set of memories, and a Network Adapter (NA), interconnected via one or more (memory) buses. Different tiles may contain different types of cores, here denoted by color.

The architecture of a many-core platform is typically modeled by a graph $G_A(R, L)$ called *architecture graph*, see, for example, References [27,41]. Here, R denotes the set of resources on the platform, that is, cores, memories, buses, NAs, and routers. The connections between these resources are reflected by the set of edges $L \subseteq R \times R$. Resources which access the NoC over the same NA are grouped into one tile $u \in U$. In heterogeneous systems, the processing cores C on different tiles can differ in architecture, instruction set, frequency, energy consumption, and so forth. To reflect this, each tile $u \in U$ can be of a certain resource type $r\_type \in P$ with $|P|$ different resource types in the system. Tiles that contain the same types of resources have the same resource type. Figure 1b illustrates the architecture graph of an exemplary heterogeneous many-core platform which is (for simplicity of illustration) composed of only two tiles: tile$_0$ and tile$_1$. Each tile consists of two cores (the type of each core is indicated by color), a NA, two shared memories, and two memory buses. Each bus connects the cores and the NA to one of the memories. The two tiles are each composed of different types of cores and are, thus, of different resource types.

3.2.1. Memory Model

Due to their scalability, distributed memory schemes are widely used in many-core systems. These schemes—also known as No Remote Memory Access (NORMA) memory architectures—restrict the accessibility of memories in each tile to resources located on that tile only [55]. The memory space in each tile may also be further partitioned into regions dedicated to individual resources or individual tasks on that tile.

3.2.2. Communication Model

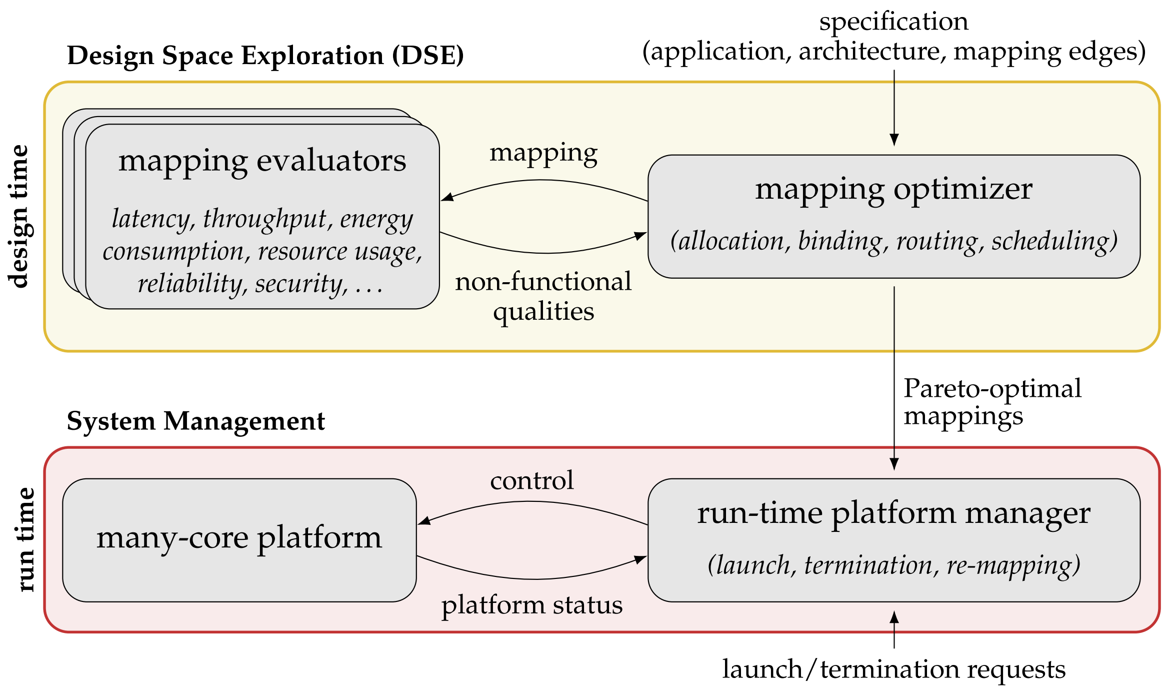
The memory scheme of a many-core platform heavily impacts its viable choices for the communication of messages between tasks. Given the shared-memory scheme inside each tile, an exchange of data between resources located on one tile (*intra-tile communications*) is realized by the producer (sender) writing the data in a given space in the tile's shared memory and the consumer (receiver) reading the data from that memory location afterwards. For an exchange of data between different tiles (*inter-tile communications*), however, explicit message passing between the producer and the consumer is necessary due to the distributed memory scheme between tiles. To that end, once the producer (sender) writes the data into the memory of the source tile, the local NA reads the data and injects it into the local router. Then, the data is forwarded through a chain of NoC routers on a hop-by-hop basis and in a pipeline fashion towards the destination router which provides the data to the NA on the destination tile. The destination NA stores the data in a dedicated memory space from where the consumer (receiver) can read the data thereafter.

3.3. Mapping

The application graph and architecture graph in the specification are connected by so-called *mapping edges* V. Each mapping edge $v = (t, c) \in V$ denotes that task $t \in T$ can be executed on core $c \in C$. Moreover, each mapping edge $v = (t, c)$ may be annotated with a set of attributes which reflect the execution characteristics of task t when executed on core c. For instance, in the context of heterogeneous many-core systems, mapping edges of a task t can also reflect the execution time of t on different types of cores in the system. Figure 1c, illustrates two exemplary mapping edges for task t_5, indicating that t_5 can be executed on core_0 or core_1.

4. Fundamentals of Hybrid Application Mapping

Hybrid Application Mapping (HAM) methodologies employ a pseudo-dynamic application mapping strategy, embodying a combination of offline mapping computation and online mapping selection. The standard flow of HAM is illustrated in Figure 3. This flow consists of (i) a design-time (offline) Design Space Exploration (DSE) step per application, followed by a (ii) run-time (online) system management step. These steps are detailed in the following.



specification
(application, architecture, mapping edges)
Design Space Exploration (DSE)
design time
mapping evaluators
latency, throughput, energy consumption, resource usage, reliability, security, ...
mapping
non-functional qualities
mapping optimizer
(allocation, binding, routing, scheduling)
Pareto-optimal mappings
System Management
run time
many-core platform
control
platform status
run-time platform manager
(launch, termination, re-mapping)
launch/termination requests

Figure 3. Flow of Hybrid Application Mapping (HAM). At design time, Design Space Exploration (DSE) is used to compute a set of Pareto-optimal mappings per application (**top**). At run time, a Run-time Platform Manager (RPM) launches applications using their precomputed mappings (**bottom**).

4.1. Offline Design Space Exploration (DSE)

In HAM, the computation of mappings for each application is performed in a DSE at design time (offline), see Figure 3 (top). The DSE takes as input the specification (detailed in Section 3) describing the space of design decisions of the currently considered application. During the DSE, various mappings of the application on the platform are generated by a *mapping optimizer* and are, subsequently, examined by a (set of) *mapping evaluator(s)* which assess the quality of each mapping w.r.t. a given (set of) non-functional design objective(s), for example, latency, throughput, and energy consumption. Subject to the application domain and the type of each non-functional objective, the respective evaluation can be performed using simulation, measurement, formal analysis, or a combination of them.

Each mapping candidate of the application on the given platform is generated in the course of four steps of design decisions, namely, resource *allocation*, task-to-core *binding*, message *routing*, and task/message *scheduling*, following the classical practice of system-level synthesis [15]. In the (i) *allocation* step, a set of platform resources, for example, cores, NAs, and routers, are specified and allocated for the execution of the application's tasks and/or for the communication of messages among them. In the (ii) *binding* step, the assignment of each task to the allocated cores is specified. In the (iii) *routing* step, a NoC route (sequence of connected routers) is specified for the communication of each message exchanged between data-dependent tasks which are bound to different tiles. Recall that messages communicated between tasks bound to the same tile are exchanged implicitly through the shared memories on that tile and, therefore, do not require a NoC route. Finally, in the (iv) *scheduling* step, the schedule of tasks $t \in T$ and messages $m \in M$ on their respective resources is specified, for example, a periodic (time) budget is computed for each task/message in case of a periodic application.

Given the NP-hard complexity of the many-core mapping optimization problem [14,15], the DSE usually employs a meta-heuristic optimization technique to find high-quality (Pareto-optimal) mappings at an acceptable computational effort. The majority of these meta-heuristic techniques operate based on an *iterative* optimization of a so-called *population* of solutions (mappings). Here, in the course of several optimization iterations, the population is used to generate new mappings (e.g., through genetic operators of mutation and crossover), and is updated with the new mappings. The set of non-dominated Pareto-optimal mappings iteratively generated thus far is always preserved. Generally, a large share of mappings generated this way could be infeasible (invalid) solutions, for example, mappings lacking the necessary NoC links for inter-tile communications. This harms the optimizer's performance as it would strive for finding feasible mappings rather than the optimization aspect [56]. As a remedy, hybrid optimization approaches combining exact (e.g., SAT or ILP) and meta-heuristic (e.g., evolutionary algorithm) techniques have emerged. These approaches, for example, SAT-decoding [57], implement powerful repair mechanisms which are capable of unambiguously mapping every point in the search space to a feasible solution, see also Reference [58].

4.2. Online System Management

In HAM, the statically computed set of mappings for each application is used at run time to launch that application on demand. For this purpose, the mappings are provided to a so-called *Run-time Platform Manager (RPM)*, see Figure 3 (bottom). Whenever an application shall be launched, the RPM selects one of the precomputed mappings of that application for which the required resources are currently available and uses that mapping to launch the application. In addition to launching applications, the RPM also terminates applications on demand and, if necessary, modifies the mapping of running applications (re-mapping) in reaction to unexpected events, for example, resource failures, or to enhance the system utilization, for example, through load/thermal balancing.

5. Tackling the Complexity

The task of application mapping is to find an allocation, binding, routing, and scheduling that is best with respect to the objectives of interest. However, with the huge amount of resources on many-core systems and more and more parallel tasks and messages of applications in modern use cases, the amount of possible mappings is immense. In particular, the large number of task-to-resource assignment options contributes significantly to the size of the search space. Finding the best or even only an optimized mapping out of this huge search space is, thus, a complex and time-consuming task. The HAM scheme described in Section 4 allows us to split the problem of application mapping between design time and run time. The general idea is to explore as much as possible of the search space already at design time. At the same time, there should be sufficient options left for the RPM to react to dynamic and unforeseeable system scenarios.

To achieve this, the DSE has to efficiently find the Pareto-optimal mapping options within this huge search space to be handed to the RPM. At the same time, the RPM must be able to efficiently find a feasible mapping candidate that can be realized on the available system resources. This section summarizes a selection of techniques that cope with the immense search space of application mapping by eliminating architectural symmetries (i.e., recurring resource-organization patterns) as well as applying architecture decomposition to decompose the complex problem into more tractable sub-problems. These techniques are likewise applicable as part of the design-time DSE and the run-time management.

The design-time DSE typically produces a huge number of Pareto-optimal mappings due to the large number of design objectives. Considering all Pareto-optimal mappings for RPM is not practical since a large set of candidates can quickly exhaust the available storage and computational capacity of the RPM. Therefore, only a fraction of the Pareto-optimal mappings must be retained, whose choice is particularly crucial for the system performance and requires new multi-objective truncation techniques tailored to many-core mapping selection. In this section, we also present a technique called *mapping distillation* that aims at reducing the number of mapping options determined by DSE so that the RPM can actually benefit from the DSE-based pre-optimization.

5.1. Constraint Graphs for Symmetry Elimination from the Search Space

Many-core architectures are heterogeneous and composed of different types of resources interconnected via a communication infrastructure. However, with the increasing parallelism, also an increasing amount of resources of equivalent resource types will be present on the chip, appearing in recurring patterns across the architecture. This means that the search space contains a large degree of redundancy in terms of symmetries, that is, mappings with equivalent resource requirements and non-functional properties. A major solution to deal with the scalability issue is, therefore, to choose a mapping representation that eliminates such symmetries. The classical application mapping as presented in Section 3 represents every possible mapping of tasks to resources. The representation introduced in Reference [41] and applied for DSE in Reference [45] instead uses a *task-cluster-to-resource-type* representation. A task cluster thereby describes a subset of application tasks which must be mapped on the same resource at run time. Each task cluster is also annotated with a resource type which specifies the type of the resource to which it must be mapped.

Figure 4 presents an example where an application consisting of tasks t_0, t_1, and t_2 should be mapped onto an architecture containing four tiles $u_{0,0}$, $u_{1,0}$, $u_{0,1}$, and $u_{1,1}$. A classical task-to-resource application mapping results in $4^3 = 64$ mapping combinations. Figure 4c illustrates four different concrete mappings. In each mapping, tasks t_0 and t_1 are mapped to a resource of type r_type_0 while t_2 is mapped to a resource of type r_type_1. The mappings differ from each other in their choice of resource instances. However, all four mappings are identical in terms of the number of allocated resources (one instance of resource type

r_type_0 and one of r_type_1), the assignment of tasks to the allocated resource types, and the hop distance, direction, and allocated bandwidth for the messages exchanged between tasks $t_0 \rightarrow t_2$ and $t_1 \rightarrow t_2$. This is indicated in the abstract representation in Figure 4d which is referred to as a *constraint graph*.

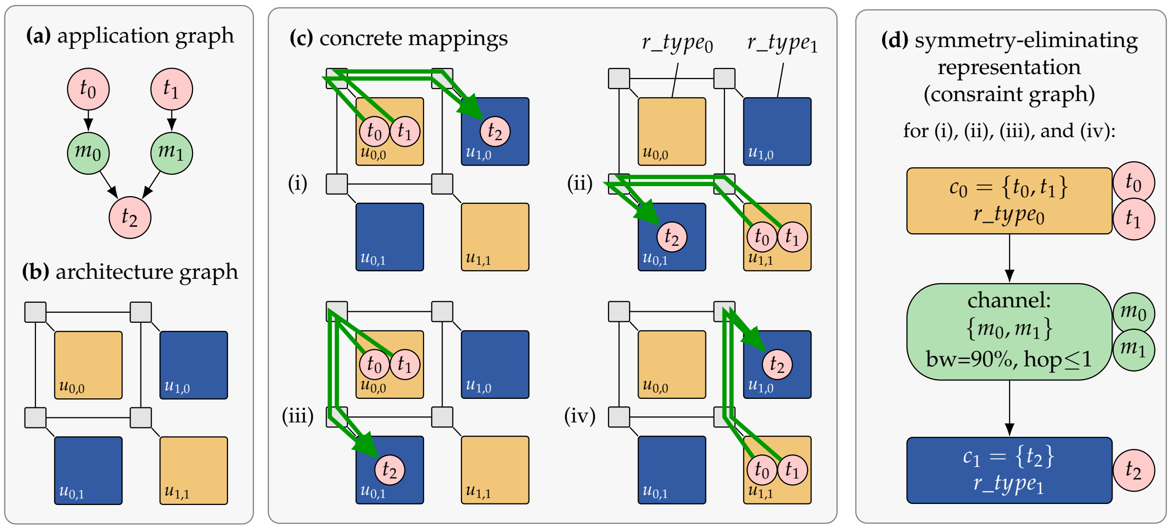


Figure 4. Illustration of application mapping and architectural symmetries, adopted from Reference [45]. The specification consists of (**a**) an application graph and (**b**) an architecture graph. (**c**) A design-time DSE explores mappings. Shown are four different concrete mappings for the three tasks t_0, t_1, and t_2 of the application onto the architecture containing four tiles $u_{0,0}$, $u_{1,0}$, $u_{0,1}$, and $u_{1,1}$ where tile types are distinguished by color. (**d**) Each mapping is transformed into an intermediate representation called *constraint graph* [41]. The constraint graph encodes rules for the RPM on how to feasibly embed the application. As depicted, the resulting constraint graph is identical for all four concrete mappings in (**c**).

The constraint graph is a representation that allows us to remove the symmetries from the search space when performing DSE based on this representation. However, it is also a representation that abstracts from concrete positional information. Determining a concrete application mapping based on a constraint graph is referred to as *constraint graph embedding*.

5.2. Symmetry-Eliminating DSE Using Constraint Graphs

Symmetry-eliminating DSE based on constraint graphs is introduced in Reference [45]. The main idea of symmetry-eliminating DSE is to explore symmetric task-cluster-to-resource-type mappings on a given target architecture based on constraint graphs instead of exploring concrete task-to-resource-instance mappings. As depicted in Figure 5, the steps to construct a mapping candidate in symmetry-eliminating DSE include (i) to explore on which resource type to bind each task (task assignment problem), (ii) to cluster subsets of tasks that are assigned to the same resource type together into task clusters (task clustering problem), and (iii) to combine the messages exchanged between the resulting task clusters to message clusters (to be routed over the NoC) and construct the constraint graph. The task assignment, task clustering, and message routing problems can be formulated as a 0-1 Integer Linear Program (ILP). The DSE can then work on this formulation by making use of SAT-decoding (see Section 4.1).

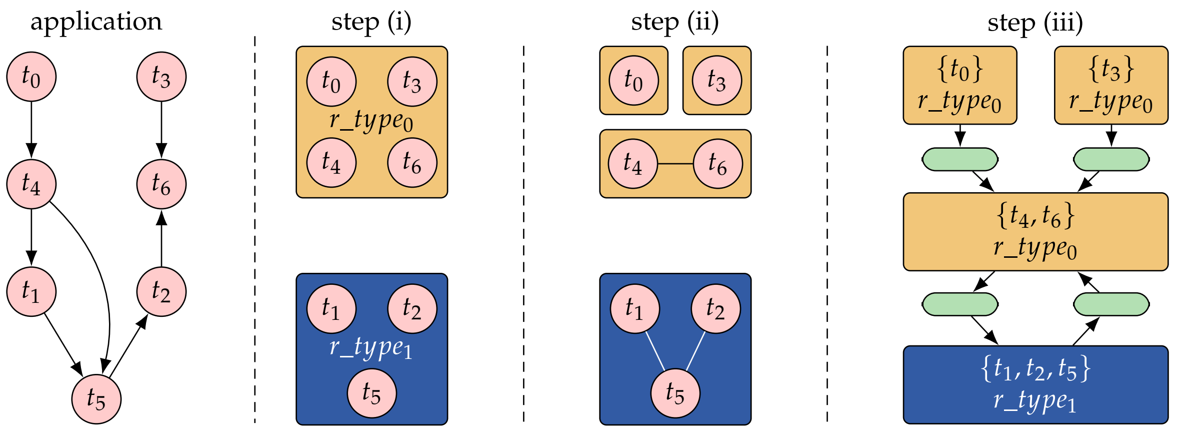


Figure 5. Symmetry-eliminating exploration consisting of three steps: (**i**) mapping tasks to resource types, (**ii**) clustering of tasks, and (**iii**) construction of constraint graph. For the sake of clarity, messages are not shown in the application. The example is adopted from Reference [45].

While the task-cluster-to-resource-type based mapping representation significantly reduces the search space, it is shown in Reference [45] that it still over-approximates the search space of feasible mappings: Due to the platform-independence of the representation (constraint graphs just encode a set of mapping solutions, but do not provide a concrete feasible one), the search space may still contain solutions, that is, constraint graphs, that cannot be feasibly mapped to a given target architecture instance due to topological constraints in a concrete architecture. Figure 6 depicts such an example of a constraint graph that cannot be feasibly mapped to the concrete architecture since resources of the required type are only available at a minimum hop distance of 4 in the target architecture, whereas the routing constraint in the constraint graph restricts the allowed distance to a maximum hop distance of 3. As a remedy, the DSE also has to perform a formal *feasibility check* to ensure that all considered solutions can be feasibly mapped to a concrete instance on the given target architecture. Techniques for determining feasible constraint graph embeddings on a given target architecture are discussed in Section 5.3. However, by means of Satisfiability Modulo Theories (SMT) techniques, it is possible to take the result of such a feasibility check as feedback for improving the DSE subsequently. For this purpose, the conditions that render a solution infeasible are extracted, and then this knowledge is added to the 0-1 ILP formulation so that not only this single but all other solutions that fulfill these conditions are removed from the search space. For the example in Figure 6, it can be deduced that all solutions with identical clustering of tasks and an identical mapping but a lower maximum hop distance (i.e., hops ≤ 1 and hops ≤ 2) will only be harder to embed and, thus, can also be excluded from the search space. Since also bandwidth requirements, hop distances, and the task clustering are included in the 0-1 ILP formulation, this can be learned by formulating respective constraints and adding them after each failed feasibility check. In Reference [45], it has been shown that this problem-specific learning technique has the potential of excluding large parts of the search space with much fewer feasibility checks.

5.3. Constraint Graph Embedding

Embedding a constraint graph in a given architecture requires (i) binding of task clusters to resources and (ii) routing of messages between them on the NoC. Predictable application execution is only possible when embedding follows the resource reservation configuration of the constraint graph. This basically means that sufficient computation resources have to be provided to bind all task clusters as well as sufficient bandwidth on communication resources to route all message clusters with the maximum allowed hop distance. Selection of resources could be done by counting the required number of resources of each resource type and, then, selecting available resources on the architecture, that is, treating this problem as a knapsack problem as, for example, done by References [59–61]. However, these approaches neglect the routing of messages between the selected resources. Also, restricting the resource selection to resources which lie within a maximal hop distance (as, for example, done by Reference [46]) neglects constrained availability of shared resources as well as resource consumption.

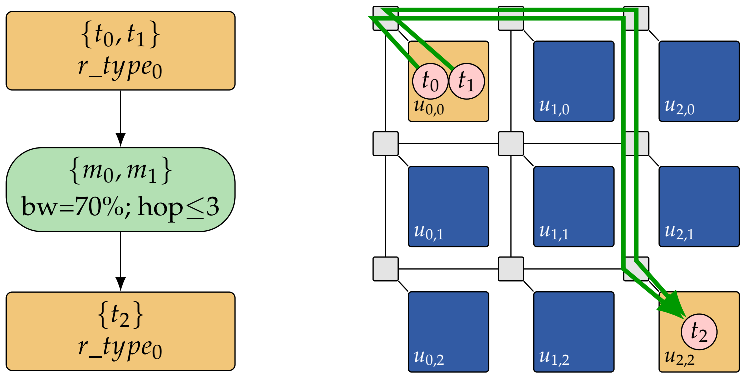


Figure 6. Illustration of a constraint graph which cannot be embedded on a given architecture: Since the constraint graph **(left)** formulates the requirement for a maximum hop distance of 3 for the transfer of m_0 and m_1 between the two task clusters, it is not embeddable on the given architecture (**right**), where the minimum hop distance between the corresponding resource types is 4.

5.3.1. Constraint Satisfaction Problem (CSP)

The above approaches may serve as a preliminary test for deciding whether there exists a feasible embedding of a constraint graph at all, as they have polynomial time complexity and form at least a *necessary condition* for feasibility. However, for determining the actual embedding, all constraints for a feasible binding have to be tested. Therefore, Reference [41] proposes to handle the embedding problem as a *Constraint Satisfaction Problem (CSP)* based on the constraint graph. Generally, a CSP is the problem of finding an assignment for a given set of variables which does not violate a given set of constraints.

The specific problem of *constraint graph embedding* consists of finding a binding for each task cluster of the constraint graph, as well as a routing between the sender and the receiver of each message. For each task cluster, a feasible binding fulfills the following *binding constraints*: (i) The resource type of the selected resource matches the required type annotated to the task cluster. (ii) The target resource provides sufficient capacity for scheduling the tasks in the cluster. For each message cluster, a feasible routing fulfills the following *routing constraints*: (i) The hop distance between the resource of the sending task cluster and the resource of the receiving task cluster is no greater than the hop distance annotated to the message cluster. (ii) Each link along the route provides sufficient capacity to meet the bandwidth requirement of the message cluster.

5.3.2. Constraint Solving Techniques

There exists a smorgasbord of techniques for solving CSPs in general. For the specific problem of constraint graph embedding, two major techniques have been evaluated which are briefly introduced in the following.

Constraint Graph Embedding Using SAT Solvers

The authors of Reference [45] formulate the constraint graph embedding problem as a satisfiability (SAT) problem. Here, the binding and routing constraints are described by a set of linear

Pseudo-Boolean equations over binary decision variables, thus, forming a 0-1 ILP. This formulation is passed to a SAT solver that returns the binding of task clusters and the routing of messages, if existent.

Constraint Graph Embedding Using Backtracking Solvers

The constraint graph embedding problem can also be solved by a backtracking algorithm as initially proposed in Reference [41]. In contrast to SAT solving techniques which work on binary decision variables, backtracking techniques work directly based on an application-specific representation of the problem. They recursively try to find a binding of each task cluster of the constraint graph to a target resource of suitable type in the architecture, while ensuring that a feasible routing between assigned variables remains possible. In case no feasible binding can be determined, a backtracking step from the most recent assignment is performed, and then, it is recursively proceeded until either all task clusters are bound or it has been verified that no embedding exists at all.

The major advantage of backtracking approaches over SAT solving is that application-specific optimizations can be applied as suggested in Reference [62], for example, restricting the resource candidates for binding a task cluster to the hop distance of already mapped connected task clusters as well as executing parallel solvers which start their search in different partitions of the architecture. With such measures, backtracking solvers exhibit better scalability for RPM as they have less memory demands compared to SAT solving techniques and are even able to determine feasible embeddings at run time within a few milliseconds also for systems with more than 100 cores. A run-time management technique to manage the mapping of multiple applications in a dynamic many-core system by applying these backtracking solvers has been proposed in Reference [42].

5.4. Architecture Decomposition for Complexity Reduction

Another natural way to reduce the problem complexity in both the design-time DSE and the run-time management in HAM is a *decomposition* of the input specification. In particular, a decomposition of the target architecture (cf. Figure 7) is well-suited for large-scale many-core architectures since they oftentimes contain multiple instances of the same resource types in a (semi-)regular topology. A careful elimination of available resources from a specification via architecture decomposition significantly reduces the number of mapping possibilities so that speed-ups and quality improvements can be achieved for both the design-time DSE and the run-time embedding in HAM.

5.4.1. Design-Time Decomposition

In the design-time DSE, architecture decomposition can be applied to reduce the size of the search space by eliminating allocatable resources and, consequently, mapping possibilities from the input specification (see Section 3). This allows for a more efficient exploration of the reduced search space and, consequently, results in a better optimization of mapping candidates. A first approach to decompose the architecture is *static decomposition* as proposed in Reference [63]. This variant of architecture decomposition removes a predetermined number of computational resources from the input specification before performing DSE, so that a sub-architecture of predetermined topology and size remains, see, for example, the three statically determined sub-architectures in Figure 7. This approach works especially well for regular many-core architectures, since it can easily be ensured that at least one resource of each required resource type remains in each sub-architecture. By performing the DSE on a number of different sub-architectures—whilst aggregating the results—it can be ensured that a variety of optimized mapping candidates is derived.

The authors of Reference [64] propose a second possibility of architecture decomposition that is better suited for irregular architectural topologies or for cases where an a-priori decision about the number and type of resource instances to be removed cannot be made. There, a *dynamic decomposition* approach is presented which utilizes information from a short preliminary DSE, that is, a pre-exploration based on the complete architecture, to determine resources to be pruned dynamically for the actual extensive DSE. During the pre-exploration, a *heat map* of the architecture is generated which stores information about resources allocated in high-quality mappings. Low-temperature areas of the heat map, that is, resources *not* part of high-quality mappings, are subsequently pruned from the architecture before the actual DSE is performed. State-of-the-art *data-mining* techniques are demonstrated to be able to extract suitable sub-architectures as well [65]. Similarly to dynamic architecture decomposition using heat maps, data mining is applied during a pre-exploration of the complete architecture. In particular, frequent-itemset mining and emergent-pattern mining are used to determine differences in resources allocated in high- vs. low-quality mappings during the DSE. Based on the obtained results, un-promising areas of the search space can thus be pruned while a reduced sub-architecture is used as input for the main DSE. All approaches discussed above are demonstrated to result in a higher quality of solutions derived by the DSE and reduce the exploration time of DSE significantly for many-core application mapping in the general case but also for constraint graphs in symmetry-eliminating DSE (cf. Section 5.1).

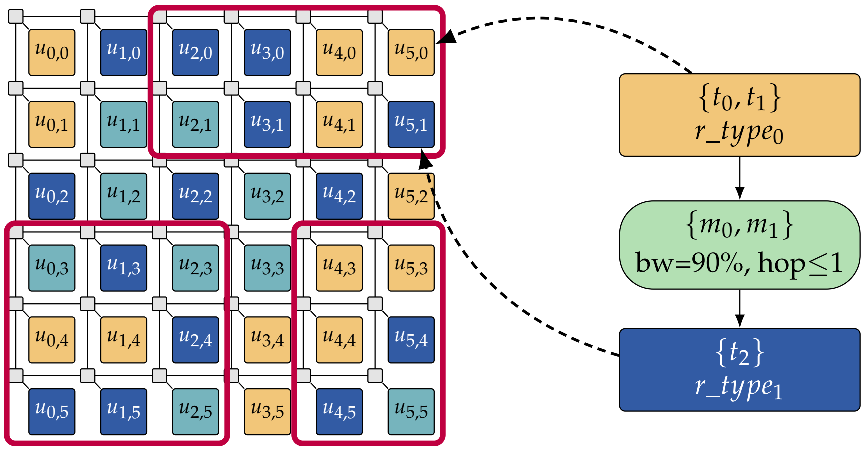

$u_{0,0}$ $u_{1,0}$ $u_{2,0}$ $u_{3,0}$ $u_{4,0}$ $u_{5,0}$
$u_{0,1}$ $u_{1,1}$ $u_{2,1}$ $u_{3,1}$ $u_{4,1}$ $u_{5,1}$
$u_{0,2}$ $u_{1,2}$ $u_{2,2}$ $u_{3,2}$ $u_{4,2}$ $u_{5,2}$
$u_{0,3}$ $u_{1,3}$ $u_{2,3}$ $u_{3,3}$ $u_{4,3}$ $u_{5,3}$
$u_{0,4}$ $u_{1,4}$ $u_{2,4}$ $u_{3,4}$ $u_{4,4}$ $u_{5,4}$
$u_{0,5}$ $u_{1,5}$ $u_{2,5}$ $u_{3,5}$ $u_{4,5}$ $u_{5,5}$
$\{t_0, t_1\}$ r_type_0
$\{m_0, m_1\}$ bw=90%, hop≤1
$\{t_2\}$ r_type_1

Figure 7. Constrain graph embedding on a 6×6 tiled many-core architecture with 3 possible sub-architectures (red boxes) created by *architecture decomposition*. The complexity of the constraint graph embedding problem for both the design-time DSE and the run-time embedding in HAM is significantly reduced by limiting the set of allocatable resources to such decomposed sub-architectures.

As mentioned in Section 5.2, a symmetry-eliminating DSE requires an additional *feasibility check* to guarantee that there exists at least one feasible concrete mapping on the given target architecture [45]. Since the complexity of the NP-complete constraint graph embedding problem grows exponentially with the number of resources in the architecture (see Section 5.3), architecture decomposition is a suitable method to reduce the complexity of such feasibility checks as well. For example, it is shown in Reference [66] how to apply architecture decomposition during feasibility checks by creating a large set of increasingly complex sub-architectures and searching for a feasible embedding on each of them. This achieves noteworthy speed-ups on average, despite the fact that the constraint graph embedding problem must eventually be solved for the complete architecture if no embedding on any generated sub-architecture exists. However, if embedding on a sub-architecture is possible, the embedding time is crucially reduced. Since each and every mapping out of the hundreds of thousands of mapping candidates generated during DSE must undergo this feasibility check, the speed-ups achieved for individual mappings accumulate to a tremendous speed-up of the overall DSE.

5.4.2. Run-Time Decomposition

At run time, the system synthesis problem must be solved to find a feasible mapping of an application on the target architecture which may already be partially occupied by concurrently running applications. The same holds true when using the constraint graph representation for run-time embedding. Architecture decomposition can decrease the embedding time in this scenario as well by limiting the search for a feasible embedding to selected parts of the architecture [62,66]. For both SAT- and backtracking-based formulations of the constraint graph embedding problem (cf. Section 5.3.2) with architecture decomposition, it is furthermore possible to parallelize the solving process by using separate solvers for different decompositions of the architecture and collating the results for even greater embedding speed-ups [62].

5.5. Mapping Distillation

The relatively high run-time overhead of constraint-graph embedding often restricts the number of mapping candidates that can be considered by the RPM. Yet, the offline DSE in HAM often delivers a huge set of Pareto-optimal mappings as it considers many design objectives: On the one hand, mappings are optimized w.r.t. several *quality objectives*, for example, latency, energy, and reliability, subject to the application domain. On the other hand, several *resource-related objectives* are often incorporated to diversify the resource demand of mappings for a better fit in various resource-availability scenarios [41,43,44]. The resulting high-dimensional objective space results in an immense number of Pareto-optimal mappings. Due to timing (and storage) restrictions at run time, only a fraction of these mappings can be provided to the RPM, necessitating the distillation (truncation) of the mappings set [67].

In the domain of multi-objective optimization, the truncation problem is well studied [68], and numerous techniques have been proposed for retaining a representative subset of Pareto-optimal points by maximizing the *diversity* of retained points in the space of design objectives, see, for example, References [69–71]. However, when adopted for mapping distillation, these well established yet generic truncation techniques typically retain mappings which exhibit a prohibitively low embeddability. The main problem here lies in the fact that these techniques regard all design objectives similarly, whereas quality objectives and resource-related objectives are of very different natures: Quality objectives denote *independent* qualities of a mapping where a high *diversity* of retained mappings is desired to offer a representative blend of quality trade-offs. In contrast, resource-related objectives *jointly* affect the embeddability of a mapping and, hence, must be considered collectively during the truncation process where both resource *diversity* and *efficiency* are desired.

In line with these observations, an automatic mapping distillation technique is presented in Reference [67] which operates as follows: The original set of Pareto-optimal mappings is first projected into the space of resource-related objectives where Pareto ranking [72] is used to sort the mappings. Then, the mappings are projected into the space of quality objectives where a grid-based selection scheme is employed to retain mappings from different regions of the quality space (ensuring diverse quality trade-offs) based on the previously computed Pareto ranks (ensuring resource efficiency and diversity). Experimental results in Reference [67] demonstrate that while retaining only a fraction (as few as only 3%) of the original set, this distillation technique highly preserves the embeddability and quality diversity of the original set and outperforms generic truncation techniques substantially.

6. Support for Hard Real-Time Applications

Embedded applications often have a set of non-functional requirements in terms of timing, safety, security, reliability, and so forth. A mapping of such applications is considered useful only if it is verified to satisfy the application's requirement(s). *Real-time applications* which have timing constraints, for example, w.r.t. their latency and/or throughput, are particularly prevalent in embedded systems.

While *soft real-time* applications can tolerate occasional violation of their timing constraints, for a *hard real-time* application, any timing violation can lead to a system failure which is not tolerable. In recent years, the rapid spread of many-core systems in various embedded domains, for example, safety-critical areas of automotive electronics, avionics, telecommunications, medical imaging, consumer electronics, and industrial automation, has led to a significant increase in the number and diversity of embedded applications with hard real-time requirements, see, for example, Reference [73] as an overview.

The hybrid (design-time/run-time) mapping scheme in HAM offers a unique opportunity for supporting hard real-time applications in dynamic embedded systems. As a result, several techniques have been proposed in recent years which enable a predictable and adaptive execution of hard-real time applications in dynamic embedded systems using HAM. In this section, we review a collection of these works after introducing the key system properties necessary for the adoption of HAM for real-time applications.

6.1. Predictability and Timing Composability

Hard real-time applications require *worst-case timing guarantees* to ensure a strict satisfaction of their timing constraints. Deriving temporal guarantees in many-core systems is particularly challenging due to their typically unpredictable execution context: On the one hand, uncertain resource behaviors, for example, (pseudo-)random cache replacement policies, branch prediction, or speculative execution, often lead to intractable variations in the timing behavior of applications such that *useful* worst-case timing guarantees cannot be derived. On the other hand, contention between concurrent applications for accessing shared resources renders the timing behavior of each application dependent not only on the arbitration policy of shared resources but also on the behavior of the concurrent applications. In a dynamic system, this dependence often results in an extensive number of possible execution scenarios which complicates the timing analysis of applications such that even if timing guarantees can be derived, they are typically too loose to be of any practical interest.

In this context, to enable deriving practical (useful) worst-case timing guarantees, two complexity-reducing system properties have been introduced: *predictability* and *timing composability*. Here, *predictability* ensures that each and every resource in the system has a predictable behavior which enables deriving useful bounds on the worst-case timing behavior of applications by means of *formal timing analysis and verification* [48]. *Timing composability*, on the other hand, ensures that concurrent applications are separated and, therefore, cannot affect the (worst-case) timing behavior of one another [48]. In a timing-composable system, resources (or resource budgets) are exclusively assigned per running application so that concurrent applications are temporally and/or spatially isolated from each other. This enables analyzing the worst-case timing behavior of each application based on its reserved resources, regardless of the presence or behavior of other applications in the system. Together, predictability and timing composability serve as the key system properties necessary for enabling an incremental design of systems with real-time applications.

Application mapping in such systems, on the one hand, involves compute-intensive timing analyses to examine the satisfaction of hard real-time constraints of each application. On the other hand, the typically dynamic nature of the application workloads in these systems necessitates workload-adaptive deployment and management of applications. These requirements render HAM methodologies particularly effective as they enable mapping optimization and timing verification for hard real-time applications at design time while empowering adaptive deployment and management of applications at run time. In this context, several HAM techniques have been proposed lately, enabling a predictable and adaptive execution of hard-real time applications in dynamic embedded systems using HAM. A collection of these works is presented in the following.

6.2. Adaptive Inter-Application Isolation and Timing Verification

In a many-core system, timing composability can be established by means of *spatial isolation* and/or *temporal isolation* among concurrent applications where resources (and/or resource budgets) are exclusively reserved for each running application at launch time. The resource reservation policy followed by the applications in a system is specified by the so-called *(inter-application) isolation scheme* selected for that system. Existing many-core systems typically employ one of the three following isolation schemes and a timing analysis tailored to their choice of isolation scheme to derive worst-case timing guarantees for real-time applications: (i) *tile reservation* in which each tile is exclusively reserved to one application, for example, References [41,46,47], (ii) *core reservation* in which each core is exclusively reserved to one application, for example [35], and (iii) *core sharing* in which core budgets are exclusively reserved per application such that a core may be shared among multiple applications. Noteworthy, sharing the NoC can hardly be avoided [73]. The choice of a system's isolation scheme regulates the amount of resources reserved for each application. This not only affects the timing behavior of that application, necessitating a timing analysis *tailored* to the system's isolation scheme to derive worst-case timing guarantees but also has a significant impact on other non-functional qualities, for example, resource utilization and energy efficiency.

A *fixed* isolation scheme imposes a single resource reservation policy on *each and every* application in the system where the amount of resources reserved per application cannot be fine-tuned according to its specific resource demands. Consequently, the majority of hereby obtained mappings either fail to satisfy the timing constraints of the application (common under a core-sharing scheme), or they exhibit an over-provisioning of resources which results in their poor performance w.r.t. other properties, for example, resource utilization and energy efficiency (common under core/tile-reservation schemes).

This issue can be lifted by *exploring the choices of isolation schemes* for each application during its mapping optimization process to find mapping solutions in which the amount of reserved resources is adjusted to the application's demands [51]. The advantage of this practice is exemplified in Figure 8 for an illustrative mapping of an application deployed on two adjacent tiles with a hard deadline of 600 ms. Figure 8c–e correspond to the three cases of fixed isolation schemes introduced above while, in Figure 8f,g, a combination of multiple isolation schemes is used. The resulting latency and resource cost reported below Figure 8c–g denote that the fixed-scheme solution in Figure 8e fails to meet the application's deadline, and those in Figure 8c,d are respectively outperformed by the ones in Figure 8f,g where isolation schemes are used in combination.

Applying isolation schemes in combination requires a timing analysis that is applicable to mappings with a mix of different isolation schemes. To address this, an *isolation-aware timing analysis* is presented in Reference [51] which is applicable to mappings with arbitrary combinations of isolation schemes on different used resources. This analysis captures the interplay between the applied mix of isolation schemes and automatically excludes inter-application timing-interference scenarios that are impossible under the given mix of isolation schemes. Reference [51] then extends the offline DSE of HAM to also perform *isolation-scheme exploration* during mapping optimization. During the DSE, the choice of isolation scheme for each resource (core/tile) is explored, and the worst-case timing behavior of each thereby obtained mappings is analyzed using the aforementioned timing analysis. This approach has been shown to improve the quality of the obtained mappings significantly (up to 67%) compared to classical fixed-scheme approaches [51].

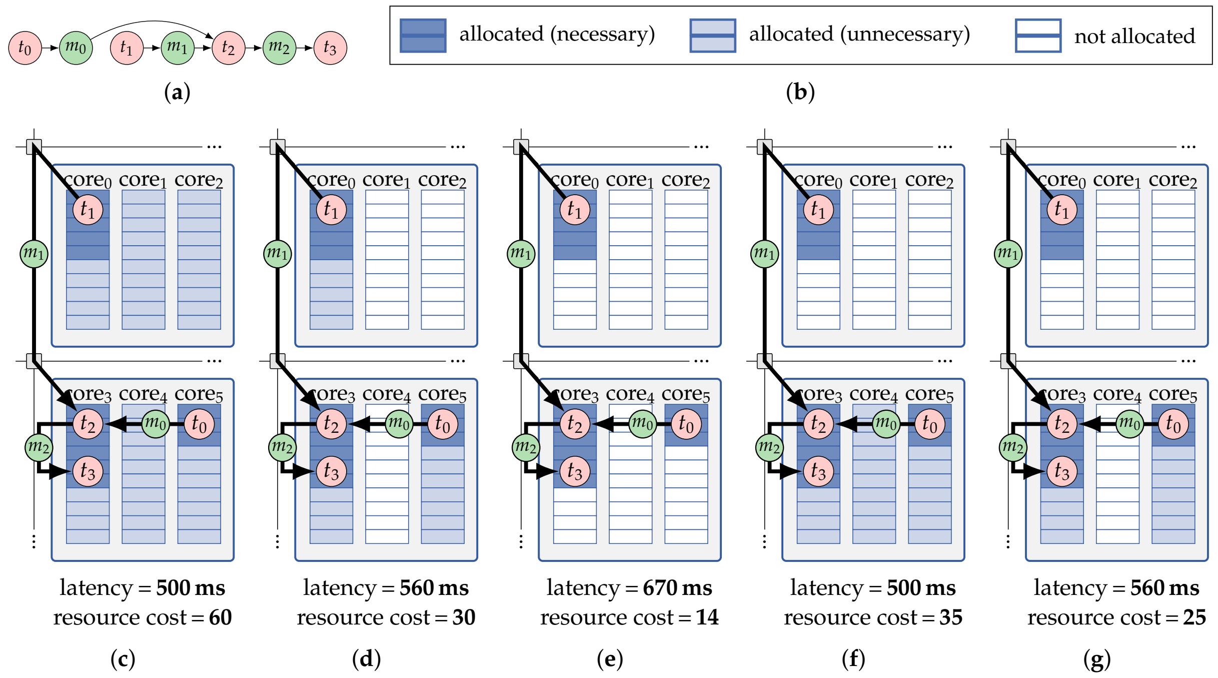


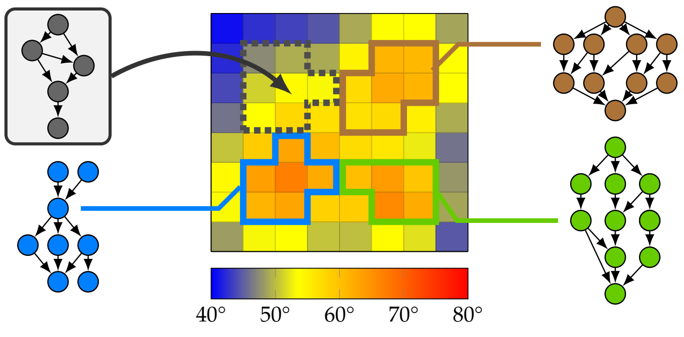
Figure 8. Example of inter-application isolation schemes for a mapping of an application (**a**). The mapping is visualized under five isolation-scheme scenarios: (**c**) fixed tile-reservation isolation scheme; (**d**) fixed core-reservation isolation scheme; (**e**) fixed core-sharing isolation scheme; (**f**) a combination of core sharing (core$_0$) and tile reservation (bottom tile) isolation schemes; (**g**) a combination of core sharing (core$_0$) and core reservation (core$_3$ and core$_5$) isolation schemes. The resulting latency and resource cost of the mapping under each scheme is given below the respective sub-figure. The description of the color code for the core (compute) budgets in (**c**)–(**g**) is given in (**b**).

6.3. Thermal Safety and Thermal Composability

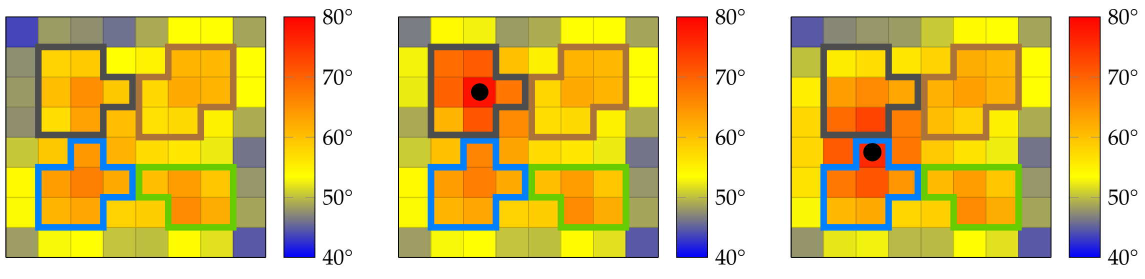
The dense integration of resources in a many-core chip results in a high density of power consumption on the chip which, in turn, leads to an increased on-chip temperature. Due to their technological limitations, chip packaging and cooling systems often fail to dissipate the generated heat fast enough which may result in overheated regions (so-called hot spots) and even lead to a chip burn-down [74]. To preserve a thermally safe operation, many-core systems employ Dynamic Thermal Management (DTM) schemes which monitor the thermal state of the chip and use mechanisms such as power gating or Dynamic Voltage and Frequency Scaling (DVFS) to prevent or counteract hot spots [75]. Since DTM countermeasures interfere with the execution of applications running in the hot spots, they may lead to the violation of hard real-time constraints which is not acceptable.

In order to preserve both temperature and timing guarantees, it is necessary to ensure the thermal safety of real-time applications *proactively*, for example, using worst-case thermal analysis of their mappings during DSE. Moreover, due to heat transfer between adjacent regions of a chip, the thermal interactions between concurrent applications must also be accounted for to ensure that the thermal behavior of one application will never lead to DTM countermeasures which affect other (possibly real-time) applications. Such an indirect inter-application interference can arise in cases where the scope of DTM countermeasures extends beyond a single core, for example, tile-level or even chip-level DVFS. Moreover, in systems where a core can be shared between multiple applications (see the core-sharing isolation scheme in Section 6.2), such indirect interferences can happen even under core-level DTM countermeasures. To eliminate such temperature-related inter-application interferences, *thermal composability* must be established and preserved in the system.

Figure 9 illustrates the significance of thermal composability in an example where a new application (gray) is to be launched in a system which is partially occupied by other running applications and is initially in a safe thermal state, see Figure 9a.



(**a**) thermal state of the chip before the new application (gray) is launched on the dashed region



(**b**) post-launch scenario 1 (thermal safe)

(**c**) post-launch scenario 2 (thermal violation)

(**d**) post-launch scenario 3 (thermal violation)

Figure 9. Example of possible thermal scenarios subsequent to launching a new (gray) application. Dots denote overheated cores. Initially, the platform is partly occupied and in a safe thermal state (**a**). Three possible post-launch thermal scenarios are shown: no thermal violations occur (**b**), a core in use by the new application is overheated (**c**), or a core in use by another application is overheated (**d**).

The mappings of all applications are individually verified at design time to be thermally safe. Although in some scenarios, the thermal safety of the system remains unaffected by the launch (e.g., see Figure 9b), subject to the initial thermal state of the system and the thermal behavior of the new application, thermal scenarios may arise in which the heat transfer between cores in use by different applications leads to thermal violations. This can result in two types of dangerous situations: (i) An application can be launched using a mapping that causes thermal violations on one or more cores it uses, see Figure 9c. This triggers DTM countermeasures, for example DVFS, that may affect the execution of this application and may violate its real-time constraints. (ii) Due to heat transfer between adjacent cores, the mapping used to launch an application can affect the temperature profile of the neighboring cores used by other applications and cause a thermal violation there, see Figure 9d. This exposes the applications running on the affected core(s) to DTM countermeasures, though they have not induced the thermal violation in the first place.

Establishing *thermal composability* is a challenging task. Whereas timing composability can be achieved by exclusive resource reservation and/or proper choice of arbitration policies to regulate the timing impact of concurrent applications on each other, achieving thermal composability is more difficult since heat transfer between neighboring cores used by different applications cannot be anticipated or controlled. To address this issue, Reference [76] presents a HAM approach which establishes thermal composability by introducing a (i) *thermal-safety analysis* to be used offline during/after the DSE and (ii) a set of *thermal-safety admission checks* to be used online by the RPM. There, the offline thermal-safety analysis computes a so-called *Thermally Safe Utilization (TSU)* for each mapping generated by the DSE. The TSU of a mapping denotes the maximum number n of active cores in the system for which that mapping is guaranteed not to lead to any thermal violations. By using the Thermal Safe Power (TSP) analysis from References [77,78], the TSU of each mapping is derived based on its power-density profile and for the worst-possible selection of n active cores (resulting in the highest temperature) so that the thermal-safety guarantee holds for *any* selection of n active cores. At run time, when launching a new application, the RPM uses a set of lightweight thermal-safety admission checks to examine the thermal safety of each mapping candidate for the current system state based on the TSU of that mapping, the TSU of other running applications, and the number of active cores in the system. By avoiding mappings that do not pass these checks, the RPM preserves thermal safety proactively and establishes thermal composability.

While TSU can be calculated for the mappings *after* the offline DSE before they are provided to the RPM, the authors of Reference [76] show that by incorporating TSU as an additional design objective to be maximized *during* the offline mapping optimization process, the DSE will deliver mappings with a higher TSU, meaning that these mappings are thermally safe for a higher number of active cores in the system. Therefore, they can be used in a larger number of system utilization levels, each corresponding to a given number of active cores in the system. This not only enables launching the application in a higher occupation of the system, but it also enhances the flexibility of the RPM as it enlarges the number of admissible mapping options available to it at different system states. This flexibility can be exploited towards secondary goals, for example, load balancing.

6.4. Online Mapping Adaptation with Hard Timing Guarantees

Run-time resource management approaches generally benefit from adapting the mapping of running applications *during* their execution, for example, for load balancing (see, for example, References [14,79]), temperature balancing (see, for example, References [80,81]), or to release the resources that are required for launching a new application. Besides such beneficial but often optional adaptions, in some situations, changing the mapping of a running application becomes inevitable. For instance, due to technology downsizing, many-core systems are subject to an increased rate of temporary/permanent resource failures as a consequence of, for example, overheating or hardware faults. A resource failure necessitates a mapping adaptation for the application(s) that depend on the affected resource in order to preserve their execution. Moreover, the performance requirements of an application may also change dynamically, for example, upon user request, such that in some cases the newly imposed requirements cannot be satisfied by the mapping already in use, thus, necessitating an online adaptation of the application's mapping.

Mapping adaptation involves changing the distribution of an application's task on the platform and is mainly realized by means of *task migration*. The adaptation process typically interferes with the timing behavior of the application and may lead to the violation of its hard real-time constraints which is not acceptable. Therefore, worst-case timing verification of the adaptation process and the post-adaptation mapping becomes necessary to ensure a seamless satisfaction of the real-time constraints. Such a timing verification, however, often relies on compute-intensive timing analyses that are not suitable for online use. In this context, the design-time/run-time scheme of HAM provides a unique opportunity to enable dynamic mapping adaptations with hard real-time guarantees at a negligible run-time compute overhead. In this line, References [82,83] present a methodology for hard real-time mapping adaptation in the form of a *reconfiguration* between the statically computed mappings of an application. Since in HAM, the set of mappings to be used at run time for each application are computed offline, the timing verification of possible reconfigurations between the mappings can also be performed offline to obtain worst-case timing guarantees for each reconfiguration option. These guarantees can then be provided to the RPM to be used for conducting reconfiguration decisions, hence, eliminating the need for online timing verification.

Mapping reconfiguration between two mappings of an application is illustrated in Figure 10. In each mapping, the dashed red arrows denote the destination tile to which the respective task must be migrated if a reconfiguration to the other mapping is performed. The authors of References [82,83] present a (i) *deterministic reconfiguration mechanism* which enables the RPM to perform each reconfiguration (involving possibly several migrations) predictably so that worst-case reconfiguration latency guarantees can be derived using formal timing analysis. They also present an (ii) *offline reconfiguration analysis* developed based on the proposed reconfiguration mechanism. During the offline analysis, first, efficient migration routes with minimized allocation overhead and migration latency are identified for the migrating tasks of each reconfiguration. Then, the worst-case latency of the whole reconfiguration process is bounded base on the worst-case timing properties of the source and target mappings and the identified migration route for each migrating task. The computed migration routes and timing guarantee of each reconfiguration are then provided to the RPM. At run time, the RPM verifies the real-time conformity of each reconfiguration candidate based on this information, the current timing requirements of the application, and the actual resource availability.

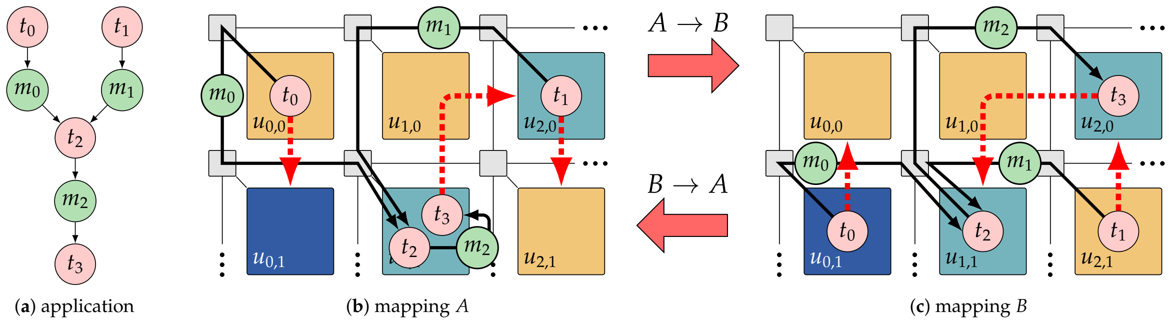


Figure 10. Example of reconfiguration between two mappings of an application (**a**) on a 2×3 section of a many-core platform. In each mapping, namely A (**b**) and B (**c**), the red dashed arrows denote migrating tasks and their destination tile for a reconfiguration to the other mapping.

This mapping reconfiguration approach is improved upon in Reference [83]: Generally, a large part of the reconfiguration latency is imposed due to the migration of tasks between tiles over the NoC. Given that the latency analysis of migration routes in a composable system is a lightweight process, in Reference [83], this part of the reconfiguration analysis is postponed to run time where the *actual* NoC load is known. Therefore, instead of relying on pessimistic assumptions about the online NoC load, the actual available bandwidth of the NoC is considered to alleviate the pessimism in the reconfiguration latency guarantee. The resulting reduction in the derived latency bounds renders many reconfiguration options admissible which would have been rejected based on their statically derived latency guarantees.

Recently, it has been demonstrated that in a composable many-core system, task migrations can be performed in such a way that a *lightweight* analysis of worst-case migration latency becomes possible. In this line, the authors of Reference [84] present a (i) *deterministic task migration mechanism* supported by a (ii) *lightweight worst-case timing analysis* which enables on-the-fly timing verification for the migration of any arbitrary subset of an application's tasks. Using this approach, the RPM is able to conduct migration decisions dynamically at run time. Thus, instead of being restricted to a limited set of reconfiguration options which were pre-explored at design time, the RPM can fine-tune its choice of migrating tasks according to the given situation at run time and verify the admissibility of the migration timing overhead on-the-fly.

7. Upcoming Trends and Future Directions: A Machine Learning-Based Perspective

Machine Learning (ML) techniques have recently gained tremendous attention from both academia and industry and are considered as promising solutions in many application domains. In this section, we discuss how ML techniques can be used to further enhance HAM methodologies. For the sake of brevity, we refrain from discussing approaches which focus on individual components of HAM, for example, the use of ML techniques for guiding the mapping optimizer during the offline DSE [85–88]. Instead, our discussion will be focused on some promising recent approaches which are more specifically tailored to a combination of mapping optimization at design time and dynamic system management at run time. In our following discussion, we categorize the approaches into two groups, see Figure 11: (i) Approaches which focus on learning the *properties* of individual mappings, the platform, or its environment and (ii) approaches which focus on learning the *actions* suited for different run-time conditions.

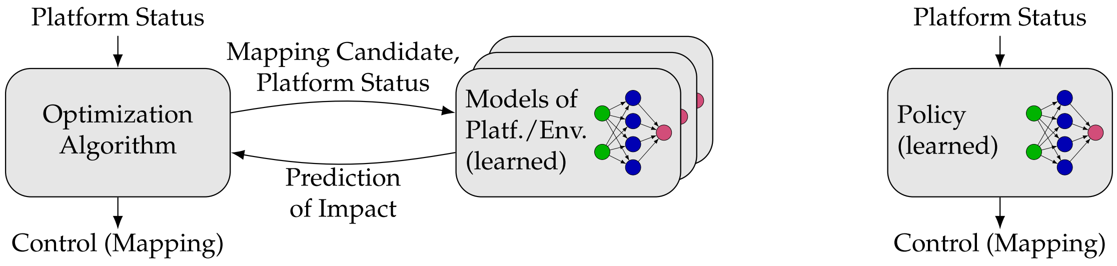


Figure 11. HAM can be supported by Machine Learning (ML)-based techniques in two ways. (**a**) Learning properties of mappings, the platform, or its environment (visualized here for learning platform and environment properties to predict the impact of mapping candidates). (**b**) Directly learning actions, i.e., learning the policy that decides the suitable action for each given situation, e.g., selecting mappings based on current platform status.

7.1. Learning Properties

The majority of challenges in HAM root in the increasing complexity of the *design space* of DSE and the *decision space* of the RPM. The exploration of the design/decision space in quest of near-optimal mappings involves the consideration of a large number of concrete mappings, each of which must be evaluated w.r.t. multiple design objectives. The extent of both spaces depends exponentially on the number of applications and tasks in the system, the number of cores on the platform, the number of possible core configurations (e.g., voltage/frequency-levels), and so forth, leading to a *combinatorial explosion* of the aforementioned spaces. Consequently, exploring the whole design/decision space in its entirety becomes impractical. Instead, a trade-off between the search overhead and the quality of the obtained solutions must be made which depends on factors such as the number of considered design points and the accuracy of their quality evaluation. This trade-off must be tackled differently by the DSE (at design time) and the RPM (at run time).

Traditionally, DSE relies on *accurate methods*, for example, simulation, to evaluate the quality of a mapping. The time required for the evaluation of each mapping can be considerably high and can even become the main timing bottleneck of the entire DSE, dictating whether the DSE is efficient, if at all feasible. Reducing the complexity of evaluations while maintaining a high accuracy is quite challenging. In contrast to the DSE, the RPM always has a strong requirement for low overhead. Hence, *heuristic* policies with negligible overhead have emerged for online use, for example, policies for maximizing the power budget in a greedy manner [89]. However, such policies may result in a low quality of run-time decisions because heuristic metrics cannot accurately capture complex platform and environment behaviors and interdependencies. At the same time, these heuristics impose only a very low overhead. Hence, a slight increase in their overhead is affordable if this improves the quality of RPM's decisions. This, however, is fairly challenging to achieve. In summary, both DSE and RPM require a flexible trade-off between accuracy and overhead.

ML models built based on *Supervised Learning (SL)* are known for their capability in approximating black-box functions. Thereby, both the achievable prediction accuracy and the prediction overhead depend on the complexity of the chosen prediction model. Importantly, SL models facilitate the exploration of different overhead-accuracy trade-offs, for example, by varying the topological parameters of a Neural Network (NN). This is a valuable property for both the design-time DSE and the run-time management in HAM.

Figure 12 illustrates how SL can enhance the overhead-accuracy trade-off in different steps of HAM. The offline DSE inherently relies on the evaluation of mappings w.r.t. several design objectives. The traditional exact analyses (e.g., using simulation) offer high accuracy, yet suffer from high overhead. Ultimately, this overhead becomes the main timing bottleneck of the DSE. The so-called *surrogate approaches* employ a NN to substitute a time-consuming simulation with a fast quality assessment of mappings at a decreased accuracy, see Figure 12 (top). On the other hand, RPMs traditionally rely on heuristics, which commonly have very low overhead but also abstract from many aspects, resulting in sub-optimal run-time decisions and, hence, limiting the achievable overall system performance. A higher performance can be achieved if the impact of each decision on the system can be assessed with a higher accuracy. ML-based models promise increased accuracy, yet at the cost of an inflated overhead, see Figure 12 (bottom). In the following, both research directions are discussed in more detail.

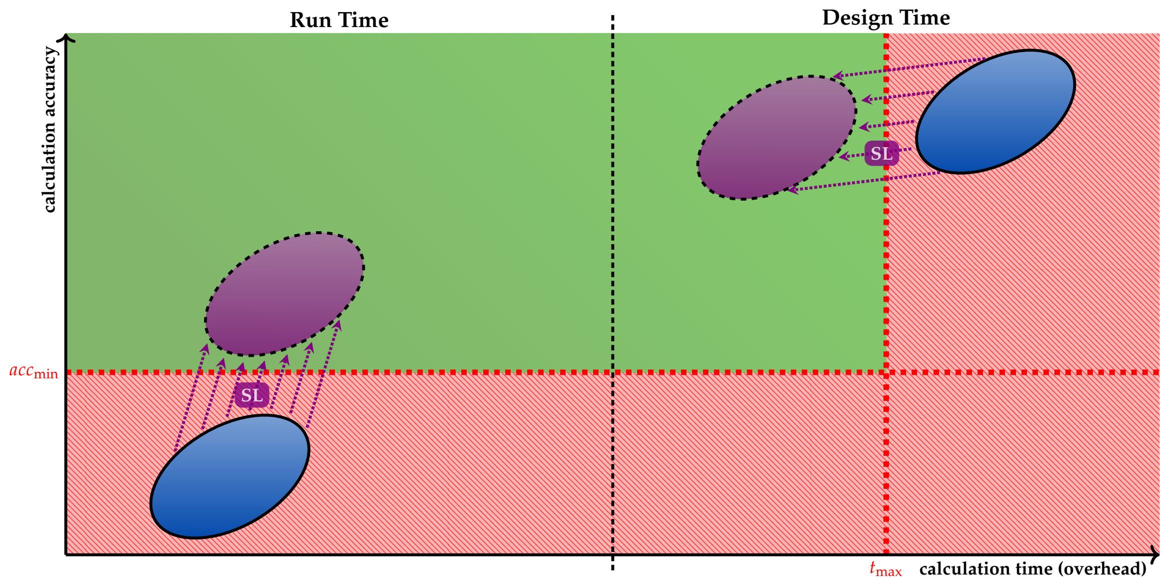


Figure 12. Learning properties using Supervised Learning (SL) can be used to enhance the overhead-accuracy trade-off for both the DSE at design time (**right**) and RPM at run time (**left**) in HAM. The ability of SL in approximating black-box functions enables a higher accuracy of run-time decisions or a faster evaluation of mappings during DSE.

7.1.1. Learning Properties for DSE

As outlined previously, reducing the evaluation time of mappings is an important research objective in the DSE community. One group of approaches which have recently been shown to be particularly effective for this purpose are *surrogate approaches* [90]. These approaches exploit the fact that most optimization techniques used in the context of DSE rely solely on the *relative* quality of each mapping in comparison with other mappings rather than the *exact* (absolute) quality of each mapping. Therefore, they rely on the *fidelity* of the evaluation function rather than its *accuracy*. In this context, surrogate approaches achieve significant evaluation-time reduction by (partially) replacing the computationally intensive exact evaluations with lightweight approximations with acceptable evaluation errors (to establish a high fidelity). The applicability of surrogates depends on the presence of patterns within the evaluation function, which must be detectable with an overhead justified by the speedup achieved through the incorporation of the surrogate method within the DSE. Naturally, their ability to predict/approximate the values of a black-box function based on previous observations makes ML-based approaches—in particular, from the domain of SL—such as linear/polynomial regressors, NNs, or Bayesian approaches, ideal candidates for the creation of surrogates [91–94].

7.1.2. Learning Properties of the Platform and its Environment

As discussed before, the limited quality of RPM decisions is a major restrictive factor of the achieved system performance at run time. In this scope, compared to traditional techniques, ML-based techniques may enhance the trade-off between the quality and the overhead of RPM decisions. One way to achieve this is to use ML-based techniques to learn models that predict the properties of the platform and its environment. These models may be used to predict the impact of a mapping candidate on metrics like power, performance, temperature, and so forth. The input to such a model is the current platform status and some features of the mapping candidate. The platform status also includes relevant features about the characteristics of the current workload. Using the prediction models, the optimization algorithm used by the RPM in its decision processes can consider the impact of many mapping candidates on various system quality metrics.

Models of the properties of the platform and its environment can be built with SL algorithms, where training data is extracted with the help of run-time or design-time profiling. Such techniques have been successfully employed, for example, for deciding task migrations [95,96]. In this scope, Reference [95] uses a lightweight NN to predict the steady-state temperature after a task migration. Reference [96] employs a NN-based model to predict the performance of a task after migrating it to another core. This model takes into account the complex workload-specific dependencies of the performance on average cache latency and power budget. Many migration candidates are assessed based on the performance prediction and the best one is selected for execution.

One advantage of learning properties is its good interpretability compared to learning actions directly. By learning properties, resource-management decisions can easily be understood by designers because the model outputs are physical properties like temperature or performance that are familiar to designers. Furthermore, since the models learn properties of the platform, they generalize to different management strategies. For instance, if a platform has several operation modes (e.g., high performance, low temperature, etc.), the models are valid in all modes and do not need to be retrained. Here, only the optimization algorithm used by the RPM needs to be adapted upon a mode change. A key drawback is that each mapping candidate needs to be assessed individually. This results in a high overhead if the number of potential actions is high [97]. To reduce this overhead, only a limited number of mapping candidates can be assessed. This, however, may result in sub-optimal mappings if mapping candidates are created at run time. HAM offers a potential to mitigate this problem through its offline pre-filtering of the possible mappings during DSE such that only Pareto-optimal mappings will be provided to the RPM. However, future work is required in this direction. We highlight some future perspectives later in Section 7.3.

7.2. Learning Actions

Existing HAM approaches offer numerous advantages over purely static or purely dynamic application mapping approaches. However, in most existing HAM-based approaches, the offline design step and the online management (decision making) step are only weakly interlinked and still strongly resemble static and dynamic design approaches, respectively. In particular, in HAM, the RPM is provided with a set of Pareto-optimal mappings generated within the offline DSE, however, without receiving any information as to which mappings to use in which run-time situations. The amount of Pareto-optimal mappings can still have a considerable size, especially in cases where abstract design goals have to be transformed into (a large number of) quantifiable objectives. The ensuing necessity to search the decision space consisting of these mappings at run time compromises the responsiveness of the RPM and/or the quality of its decisions. In what follows, we discuss a few directions to address this issue by means of ML-based techniques, namely, *Imitation Learning (IL)* and *Reinforcement Learning (RL)*, which can be applied either at design time or at run time to refine the decision strategy of the RPM.

7.2.1. Imitation Learning (IL)

Imitation Learning (IL) uses Supervised Learning (SL) to construct an oracle for sequential mapping-decision processes. The prerequisite for IL is the availability of labeled training data that resembles the platform status occurring at run time. The training data is created at design time with the help of training benchmarks. Each training sample hereby represents a certain platform status and is labeled with a mapping which is considered optimal for this platform status. This typically involves brute-forcing a large number of mappings to find the optimal one (The optimal mapping for a given platform status can be found, e.g, by an enumeration of the available mappings). Since it is not possible to evaluate every existing mapping combination, only a reduced set of pre-optimized mappings found by the DSE are considered as labels. The RPM learns the actions of choosing mappings at design time and then imitates the actions at run time by adapting them to the given platform status.

The authors of Reference [98] propose a HAM approach that uses IL and incorporates—in addition to the platform status—the influence of input data onto the execution characteristics of the applications into the mapping-inference process. Here, no functional properties of the applications have to be known. To reduce the computation complexity, input data with similar execution properties are clustered into *data scenarios*. This allows for a finer granularity of mapping decisions since the workload dynamism induced by the varying input data can be captured by tailored mappings for each data scenario. The clustering of data into scenarios and the optimization of corresponding mappings are performed at design time using the data-driven scenario-aware DSE approach from Reference [99] based on training data.

This approach entails identifying the best-suited scenario for incoming data at run time before inferring the mapping. However, for complex input data like images, it may not be possible to determine the best scenario prior to processing the data and identifying/observing its execution properties. As a consequence, the scenarios are identified after processing the data based on the evoked execution properties. For that, SL is used where a NN is trained at design time to classify the execution data vectors of the training data depending on the current platform status into the best-suited scenarios. This NN is then used to identify the scenario of incoming input data at run time. The scenario for the next data is afterwards derived from the identified scenarios of the previously processed input data. Here, another model is utilized whose selection algorithm is optimized offline by genetic programming based on the sequence of training data. If no correlation between subsequent input data is found in the training sequence, a scenario optimized for the average-case data is chosen.

Finally, a mapping is selected from the set of mappings tailored to the chosen scenario depending on the given objectives. It has been shown for a soft real-time setting in Reference [100], that IL with a NN outperforms a statistic-based approach in terms of both the number of deadline misses and the energy consumption. Here, the NN infers the mapping based on a fixed deadline and the history of execution properties of the previously processed input data. The combination of all three models of (i) scenario identification, (ii) scenario selection, and (iii) mapping selection forms the entity of a scenario-based RPM responsible for the online system management of the scenario-aware HAM. The structure of this HAM approach for a single application is shown in Figure 13 differentiating between the offline (design time) phase and the online (run time) phase.

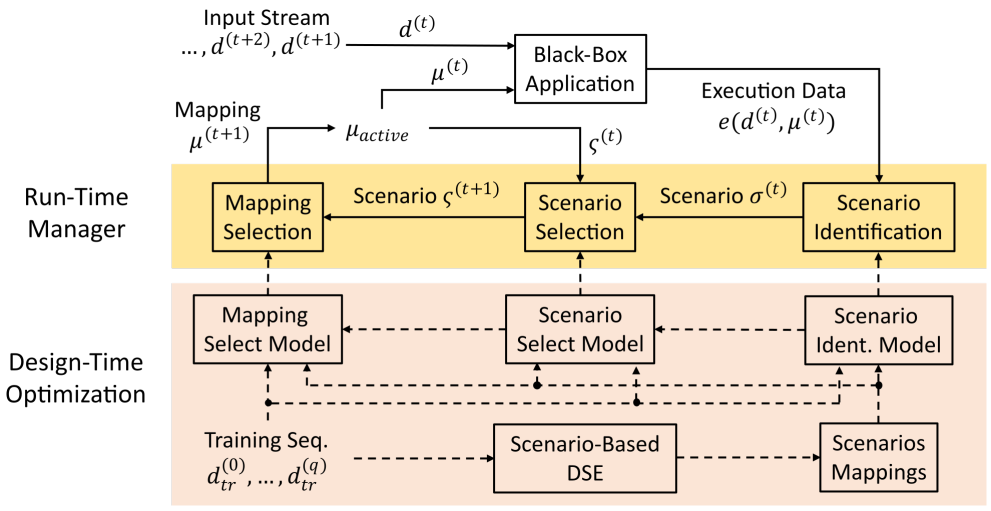


Figure 13. Structure of the data-driven scenario-based HAM. At design time (**bottom**), a DSE optimizes data scenarios and mappings, and ML models are optimized for the mapping inference using training data. At run time (**top**), the RPM uses the optimized models, scenarios, and mappings to select tailored mappings for the application depending on the incoming input data.

In summary, IL can help to tackle the uncertainty of workload distribution at run time by learning patterns in the interplay between input and application characteristics from training data at design time. In the HAM approach above, a mapping is not directly inferred by a single IL model, but instead, by a succession of three separate models specialized on different aspects of mapping selection. This facilitates the training and convergence of the models. Additionally, the whole mapping-decision process becomes more comprehensible which, for example, facilitates the detection of outliers.

7.2.2. Reinforcement Learning (RL)

The majority of existing HAM approaches, as outlined in the previous sections, are established based on a relatively straightforward combination of existing static (design-time) and dynamic (run-time) approaches. A major challenge encountered by these HAM approaches, during both the offline DSE and the online system management steps, is the evaluation of mappings, that is, the estimation of their impact on the overall system performance. In a *static system*, where exactly one mapping is used throughout the entire lifetime of the system, mappings can be evaluated purely based on their non-functional characteristics, for example, energy consumption, which are easy to quantify. The performance of a *dynamic system*, on the other hand, is not determined based on the quality of a single mapping, but instead, based on the entire decision *strategy* of the RPM, that is, the set of available mappings and the rules dictating which mapping to use in which run-time situations. The usefulness of a single mapping can, therefore, (i) only be evaluated when taking into account the other mappings and the decision rules of the RPM's strategy and (ii) in some cases, may only become apparent after a prolonged time interval. The design and decision approaches used in most existing HAM solutions are not capable of (efficiently) considering these complex interrelations between individual mappings and the overall system performance, which significantly impairs their ability to generate RPM strategies.

In the domain of ML, *Reinforcement Learning (RL)* approaches have been specifically designed to generate sophisticated behavior strategies to address various conditions in complex dynamic environments. In particular, these approaches are designed to consider the non-trivial long-term effects of the chosen actions. In the following, we discuss opportunities to adopt RL techniques in HAM.

Reinforcement Learning (RL) at Run Time

When used at run time, RL offers the opportunity to incorporate learning capabilities into the decision process of the RPM in a dynamic system. By observing the effects of the mappings it selects in particular run-time situations, the RPM can, over time, estimate the *utility* of each available policy, denoting the performance impact of a particular choice of mapping in a particular situation. This ability enables the RPM to (i) generate a system management strategy which is precisely tailored to the observed online conditions and (ii) adapt the strategy in the case of (unforeseen) changes in the conditions. For an example of an adaptable system applying RL at run time, see Reference [101].

While RL approaches are capable of dynamically generating a suitable strategy for (previously unknown) dynamic conditions, the time (in terms of the number of mapping selection actions) until a high-quality strategy is found scales with the number of possible conditions and the number of possible actions and, hence, can become prohibitively long. Furthermore, the learning process typically involves a trial-and-error phase, during which the RPM is likely to take undesirable—or, in the case of safety-critical systems, even dangerous—actions. The long adaptation times and the necessity of an unstable exploration phase have, for a long time, been the main impediments to the application of RL in the area of (safety-critical) real-time embedded systems. With its capability to significantly reduce the decision space of the RPM—and, thereby, also the size of the state space of any RL algorithm used therein—HAM offers a unique opportunity to overcome these weaknesses and unleash the potential of run-time RL techniques for dynamically adapted systems.

Reinforcement Learning (RL) at Design Time

Most existing design approaches utilizing RL techniques use them exclusively for the run-time adaptation of the system. However, an integration of RL techniques into the offline design phase of HAM offers several advantages. Extending static optimizers with the concepts of long-term utility and a cooperative usage of the mappings can address the previously outlined weaknesses of HAM w.r.t. the evaluation of individual mappings. Furthermore, considering (parts of) the interactions between the system and its environment (which imposes run-time conditions onto the system) at design time enables the offline optimization of not only the mappings, but also their activation conditions. In such a scheme, the RPM is provided with a decision strategy which is pre-optimized for the (statically known portion of the) environment characteristics, so that a run-time learning phase can be significantly shortened, carried out within safe action bounds, or even completely avoided.

Based on these ideas, the authors of Reference [102] present a novel optimization framework, LOCAL, which is specifically tailored for HAM in adaptable systems. In its structure, LOCAL is inspired by Learning

Classifier Systems (LCSs) [103], optimization frameworks developed specifically for complex and dynamic problems. Instead of individual mappings, LOCAL co-optimizes a set of rule-action tuples, so-called *policies*, where the action corresponds to a concrete mapping and the rule describes the run-time conditions for the usage of this mapping. By emulating the iterative interactions between the adaptive system and its environment, LOCAL considers (i) the available information about the dynamic external events expected at run time, (ii) the way in which the adaptive system influences its environment, and (iii) the long-term utility of individual policies (rule-action tuples). Within each interaction with the environment (corresponding to new conditions imposed by the environment), LOCAL generates/optimizes policies in a *constrained DSE*, which is restricted to explore only the search space of mappings that are applicable under the given conditions. With this search-space reduction, LOCAL is capable of generating mappings which are exactly tailored to realistic run-time situations.

In Reference [102], LOCAL is used to optimize the embeddability of applications in a many-core system featuring a dynamic launch and termination of applications. It has been shown that, compared to the mapping set generated by a static optimizer, the strategy generated by LOCAL offers a higher embeddability, while containing a significantly smaller number of mappings.

7.3. Future Perspectives

While the approaches presented above already demonstrate the great potential of using ML techniques in HAM, there are still several open questions demanding further research and investigation. In the following, we discuss two particularly interesting research directions, both centered on the way that the design approach addresses the availability of information at design time.

7.3.1. Integration of Expert Knowledge

The first research direction focuses on the exploitation of problem-specific knowledge available at design time. Similarly to any other optimization process, the effectiveness of both offline and online steps of HAM is directly influenced by the amount of the injected problem-specific knowledge. Particularly, in embedded domains, where a large amount of experience and expertise has been accumulated over decades, the development of efficient ways to incorporate this information into the design process is of paramount importance. Techniques gathering information about different application classes or their launch patterns (see Sections 7.2.1 and 7.2.2) constitute the first steps in this direction. However, the integration of more complex knowledge such as known property dependencies (e.g., the fact that power increases monotonically with the voltage/frequency level) or functional application knowledge remains challenging. This applies in particular to optimization approaches comprising multiple interdependent phases (e.g., the scenario-based RPM presented in Section 7.2.1) and complex ML approaches such as NNs which, while offering multiple different points for the injection of expert knowledge [97], are especially difficult to customize and to interpret.

7.3.2. Balancing Offline and Online Learning

The second research direction which we would like to highlight focuses on finding the best way to address uncertainties in the run-time conditions. Thanks to the great versatility of ML approaches, uncertainties in the run-time conditions of the designed system can be addressed during either the offline or the online step of HAM. This introduces an interesting trade-off: On the one hand, the time and compute power available at design time can be used to train a complex model, enabling the system to react to any possible outcome of the uncertainty at hand. On the other hand, transferring the learning process to run time may enable the creation of a lightweight model which fits the actually observed situations. While a more lightweight model is likely to result in a faster and more energy-efficient inference, unexpected changes in run-time conditions may render it useless or necessitate a retraining phase. The development of HAM approaches which are capable of automatically finding the optimal balance between offline and online learning—which is highly problem-dependent and has a strong impact on the efficiency and reliability of the designed system—constitutes one of the most challenging and promising directions for future research.

8. Conclusions

This paper provides an overview of Hybrid Application Mapping (HAM) as a promising approach for mapping emerging applications to embedded many-core systems. We introduced the fundamentals of HAM and the major design challenges addressed by HAM. An elaborate discussion of the new challenges encountered by HAM was presented together with an overview of a collection of state-of-the-art techniques proposed to address these challenges. The paper also outlined a series of open topics and challenges in HAM, presented a summary of some emerging machine-learning-based directions for addressing these challenges, and highlighted possible future directions.

Author Contributions: Conceptualization, methodology, investigation, visualization, and writing–review and editing: B.P., M.G., J.H., H.K., M.R., V.R., T.S., F.S., J.S., J.T., A.W., and S.W.; writing–original draft preparation: B.P., M.G., H.K., M.R., V.R., T.S., F.S., J.S., A.W., and S.W.; supervision, project administration, and funding acquisition: M.G., J.H., J.T., and S.W. All authors have read and agreed to the published version of the manuscript.

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

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Tilera Corporation. *Tile Processor Architecture Overview for the TILE-Gx Series*; Tilera Corporation: San Jose, CA, USA, 2012.
2. De Dinechin, B.D.; Ayrignac, R.; Beaucamps, P.E.; Couvert, P.; Ganne, B.; de Massas, P.G.; Jacquet, F.; Jones, S.; Chaisemartin, N.M.; Riss, F.; et al. A clustered manycore processor architecture for embedded and accelerated applications. In Proceedings of the IEEE Conference on High Performance Extreme Computing (HPEC), Waltham, MA, USA, 10–12 September 2013; pp. 1–6.
3. Howard, J.; Dighe, S.; Hoskote, Y.; Vangal, S.; Finan, D.; Ruhl, G.; Jenkins, D.; Wilson, H.; Borkar, N.; Schrom, G.; et al. A 48-core IA-32 message-passing processor with DVFS in 45 nm CMOS. In Proceedings of the International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 7–11 February 2010; pp. 108–109.
4. Bohnenstiehl, B.; Stillmaker, A.; Pimentel, J.; Andreas, T.; Liu, B.; Tran, A.; Adeagbo, E.; Baas, B. A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In Proceedings of the IEEE Symposium on VLSI Circuits (VLSI-Circuits), Honolulu, HI, USA, 15–17 June 2016; pp. 1–2.
5. Kovač, M.; Notton, P.; Hofman, D.; Knezović, J. How Europe is preparing its core solution for exascale machines and a global, sovereign, advanced computing platform. *Math. Comput. Appl.* **2020**, *25*, 46. [CrossRef]
6. Teich, J.; Henkel, J.; Herkersdorf, A.; Schmitt-Landsiedel, D.; Schröder-Preikschat, W.; Snelting, G. Invasive computing: An overview. In *Multiprocessor System-on-Chip*; Springer: New York, NY, USA, 2011; pp. 241–268.
7. Irza, J.; Doerr, M.; Solka, M. A third generation many-core processor for secure embedded computing systems. In Proceedings of the IEEE Conference on High Performance Extreme Computing (HPEC), Waltham, MA, USA, 12–14 September 2012; pp. 1–3.

8. Exynos 5 Octa (5422). 2019. Available online: http://www.samsung.com/exynos/ (accessed on 12 November 2020).
9. Hansen, L. Unleash the Unparalleled Power and Flexibility of Zynq UltraScale+ MPSoCs. *Cell* **2015**, *2*, 3–4.
10. Kissler, D.; Hannig, F.; Kupriyanov, A.; Teich, J. A highly parameterizable parallel processor array architecture. In Proceedings of the International Conference on Field Programmable Technology (FPT), Bangkok, Thailand, 13–15 December 2006; pp. 105–112.
11. Hannig, F.; Lari, V.; Boppu, S.; Tanase, A.; Reiche, O. Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach. *ACM Trans. Embed. Comput. Syst.* **2014**, *13*, 1–29. [CrossRef]
12. Oliver, N.; Sharma, R.R.; Chang, S.; Chitlur, B.; Garcia, E.; Grecco, J.; Grier, A.; Ijih, N.; Liu, Y.; Marolia, P.; et al. A reconfigurable computing system based on a cache-coherent fabric. In Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 30 November–2 December 2011; pp. 80–85.
13. Martin, G. Overview of the MPSoC design challenge. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 24–28 July 2006; pp. 274–279.
14. Kobbe, S.; Bauer, L.; Lohmann, D.; Schröder-Preikschat, W.; Henkel, J. DistRM: Distributed resource management for on-chip many-core systems. In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS), Taipei, Taiwan, 9–14 October 2011; pp. 119–128.
15. Blickle, T.; Teich, J.; Thiele, L. System-level synthesis using evolutionary algorithms. *Des. Autom. Embed. Syst.* **1998**, *3*, 23–58. [CrossRef]
16. Davis, L. *Handbook of Genetic Algorithms*; VNR Computer Library, Van Nostrand Reinhold: New York, NY, USA, 1991.
17. Fonseca, C.M.; Fleming, P.J. An overview of evolutionary algorithms in multiobjective optimization. *Evol. Comput.* **1995**, *3*, 1–16. [CrossRef]
18. Fonseca, C.M.; Fleming, P.J. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Proceedings of the International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA, 17–21 July 1993; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1993; pp. 416–423.
19. Kennedy, J. Particle swarm optimization. In *Encyclopedia of Machine Learning*; Springer: Boston, MA, USA, 2010; pp. 760–766.
20. Marwedel, P. *Embedded System Design*; Springer: Dortrecht, The Netherlands, 2006; Volume 1.
21. Teich, J. Hardware/software codesign: The past, the present, and predicting the future. *Proc. IEEE* **2012**, *100*, 1411–1430. [CrossRef]
22. Hansson, A.; Coenen, M.; Goossens, K. Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), Nice, France, 16–20 April 2007; pp. 1–6.
23. Quan, W.; Pimentel, A.D. A hybrid task mapping algorithm for heterogeneous MPSoCs. *ACM Trans. Embed. Comput. Syst.* **2015**, *14*, 1–25. [CrossRef]
24. Akesson, B.; Molnos, A.; Hansson, A.; Angelo, J.A.; Goossens, K. Composability and predictability for independent application development, verification, and execution. In *Multiprocessor System-on-Chip*; Springer: New York, NY, USA, 2011; pp. 25–56.
25. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Mapping on multi/many-core systems: Survey of current and emerging trends. In Proceedings of the Design Automation Conference (DAC), Austin, TX, USA, 2–6 June 2013; pp. 1–10.
26. Chou, C.L.; Ogras, U.Y.; Marculescu, R. Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2008**, *27*, 1866–1879. [CrossRef]
27. Teich, J.; Glaß, M.; Roloff, S.; Schröder-Preikschat, W.; Snelting, G.; Weichslgartner, A.; Wildermann, S. Language and compilation of parallel programs for *-Predictable MPSoC execution using Invasive Computing. In Proceedings of the International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), Lyon, France, 21–23 September 2016; pp. 313–320.

28. Wildermann, S.; Bader, M.; Bauer, L.; Damschen, M.; Gabriel, D.; Gerndt, M.; Glaß, M.; Henkel, J.; Paul, J.; Pöppl, A.; et al. Invasive computing for timing-predictable stream processing on MPSoCs. *IT-Inf. Technol.* **2016**, *58*, 267–280. [CrossRef]
29. Bonfietti, A.; Benini, L.; Lombardi, M.; Milano, M. An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), Dresden, Germany, 8–12 March 2010; pp. 897–902.
30. Hu, J.; Marculescu, R. Energy-and performance-aware mapping for regular NoC architectures. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2005**, *24*, 551–562.
31. Marcon, C.; Borin, A.; Susin, A.; Carro, L.; Wagner, F. Time and energy efficient mapping of embedded applications onto NoCs. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), Shanghai, China, 18–21 January 2005; pp. 33–38.
32. Lin, L.Y.; Wang, C.Y.; Huang, P.J.; Chou, C.C.; Jou, J.Y. Communication-driven task binding for multiprocessor with latency insensitive network-on-chip. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), Shanghai, China, 18–21 January 2005; pp. 39–44.
33. Hu, J.; Marculescu, R. Energy-aware mapping for tile-based NoC architectures under performance constraints. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), Kitakyushu, Japan, 21–24 January 2003; pp. 233–239.
34. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
35. Mariani, G.; Avasare, P.; Vanmeerbeeck, G.; Ykman-Couvreur, C.; Palermo, G.; Silvano, C.; Zaccaria, V. 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), Dresden, Germany, 8–12 March 2010; pp. 196–201.
36. Giannopoulou, G.; Stoimenov, N.; Huang, P.; Thiele, L.; de Dinechin, B.D. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Syst.* **2016**, *52*, 399–449. [CrossRef]
37. Sahu, P.K.; Shah, T.; Manna, K.; Chattopadhyay, S. Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization. *IEEE Trans. Large Scale Integr. Syst.* **2014**, *22*, 300–312. [CrossRef]
38. Singh, A.K.; Dziurzanski, P.; Mendis, H.R.; Indrusiak, L.S. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. *ACM Comput. Surv.* **2017**, *50*, 1–40. [CrossRef]
39. Al Faruque, M.A.; Krist, R.; Henkel, J. ADAM: Run-time agent-based distributed application mapping for on-chip communication. In Proceedings of the Design Automation Conference (DAC), Anaheim, CA, USA, 8–13 June 2008; pp. 760–765.
40. Schranzhofer, A.; Chen, J.J.; Thiele, L. Power-aware mapping of probabilistic applications onto heterogeneous MPSoC platforms. In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, San Francisco, CA, USA, 13–16 April 2009; pp. 151–160.
41. Weichslgartner, A.; Gangadharan, D.; Wildermann, S.; Glaß, M.; Teich, J. 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), Uttar Pradesh, India, 12–17 October 2014; pp. 1–10.
42. Wildermann, S.; Weichslgartner, A.; Teich, J. Design methodology and run-time management for predictable many-core systems. In Proceedings of the IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, Auckland, New Zealand, 13–17 April 2015; pp. 103–110.
43. Weichslgartner, A.; Wildermann, S.; Gangadharan, D.; Glaß, M.; Teich, J. A design-time/run-time application mapping methodology for predictable execution time in MPSoCs. *ACM Trans. Embed. Comput. Syst.* **2018**, *17*, 1–25. [CrossRef]

44. Weichslgartner, A.; Wildermann, S.; Glaß, M.; Teich, J. *Invasive Computing for Mapping Parallel Programs to Many-Core Architectures*; Springer: Singapore, 2018.
45. Schwarzer, T.; Weichslgartner, A.; Glaß, M.; Wildermann, S.; Brand, P.; Teich, J. Symmetry-eliminating design space exploration for hybrid application mapping on many-core architectures. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2017**, *37*, 297–310. [CrossRef]
46. Singh, A.K.; Kumar, A.; Srikanthan, T. Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.* **2013**, *18*, 1–29. [CrossRef]
47. Khanh, P.N.; Singh, A.K.; Kumar, A.; Aung, K.M.M. 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), Los Alamitos, CA, USA, 4–6 September 2013; pp. 513–521.
48. Kopetz, H. *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed.; Springer: New York, NY, USA, 2011.
49. Hansson, A.; Goossens, K.; Bekooij, M.; Huisken, J. CoMPSoC: A template for composable and predictable multi-processor system on chips. *ACM Trans. Des. Autom. Electron. Syst.* **2009**, *14*, 1–24. [CrossRef]
50. Schoeberl, M.; Abbaspour, S.; Akesson, B.; Audsley, N.; Capasso, R.; Garside, J.; Goossens, K.; Goossens, S.; Hansen, S.; Heckmann, R.; et al. T-CREST: Time-predictable multi-core architecture for embedded systems. *J. Syst. Archit.* **2015**, *61*, 449–471. [CrossRef]
51. Pourmohseni, B.; Smirnov, F.; Wildermann, S.; Teich, J. Isolation-aware timing analysis and design space exploration for predictable and composable many-core systems. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), Stuttgart, Germany, 9–12 July 2019; Volume 12, pp. 1–24.
52. Weichslgartner, A.; Wildermann, S.; Götzfried, J.; Freiling, F.; Glaß, M.; Teich, J. Design-time/run-time mapping of security-critical applications in heterogeneous MPSoCs. In Proceedings of the International Workshop on Software and Compilers for Embedded Systems (SCOPES), Sankt Goar, Germany, 23–25 May 2016; pp. 153–162.
53. Heisswolf, J.; König, R.; Kupper, M.; Becker, J. Providing multiple hard latency and throughput guarantees for packet switching networks on chip. *Comput. Electr. Eng.* **2013**, *39*, 2603–2622. [CrossRef]
54. Borkar, S. Thousand core chips: A technology perspective. In Proceedings of the Design Automation Conference (DAC), San Diego, CA, USA, 4–8 June 2007; pp. 746–749.
55. Madalozzo, G.; Duenha, L.; Azevedo, R.; Moraes, F.G. Scalability evaluation in many-core systems due to the memory organization. In Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS), Monte Carlo, Monaco, 11–14 December 2016; pp. 396–399.
56. Smirnov, F.; Pourmohseni, B.; Glaß, M.; Teich, J. Variety-aware routing encoding for efficient design space exploration of automotive communication networks. In Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems, Crete, Greece, 3–5 May 2019; Volume 1, pp. 242–253.
57. Lukasiewycz, M.; Glaß, M.; Haubelt, C.; Teich, J. SAT-decoding in evolutionary algorithms for discrete constrained optimization problems. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Singapore, 25–28 September 2007; pp. 935–942.
58. Glaß, M.; Teich, J.; Lukasiewycz, M.; Reimann, F. Hybrid optimization techniques for system-level design space exploration. In *Handbook of Hardware/Software Codesign*; Ha, S., Teich, J., Eds.; Springer: Dortrecht, The Netherlands, 2017; pp. 217–246.
59. Wildermann, S.; Glaß, M.; Teich, J. Multi-objective distributed run-time resource management for many-cores. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–6.
60. Bini, E.; Buttazzo, G.; Eker, J.; Schorr, S.; Guerra, R.; Fohler, G.; Arzen, K.E.; Segovia, V.R.; Scordino, C. Resource management on multicore systems: The ACTORS approach. *IEEE Micro* **2011**, *31*, 72–81. [CrossRef]
61. Ykman-Couvreur, C.; Nollet, V.; Catthoor, F.; Corporaal, H. Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In Proceedings of the International Symposium on System-on-Chip, Tampere, Finland, 13–16 November 2006; pp. 1–4.

62. Schwarzer, T.; Roloff, S.; Richthammer, V.; Khaldi, R.; Wildermann, S.; Glaß, M.; Teich, J. On the complexity of mapping feasibility in many-core architectures. In Proceedings of the International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Hanoi, Vietnam, 12–14 September 2018; pp. 176–183.
63. Richthammer, V.; Glaß, M. On search-space restriction for design space exploration of multi-/many-core systems. In *Workshop "Methoden und Beschreigungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen" (MBMV)*; University of Tübingen: Tübingen, Germany, 13–14 May 2018.
64. Richthammer, V.; Fassnacht, F.; Glaß, M. Search-Space decomposition for system-level design space exploration of embedded systems. *ACM Trans. Des. Autom. Electron. Syst.* **2020**, *25*, 1–32. [CrossRef]
65. Richthammer, V.; Scheinert, T.; Glaß, M. Data mining in system-level design space exploration of embedded systems. In Proceedings of the International Conference on Embedded Computer Systems (SAMOS), Samos, Greece, 5–9 July 2020.
66. Richthammer, V.; Schwarzer, T.; Wildermann, S.; Teich, J.; Glaß, M. Architecture decomposition in system synthesis of heterogeneous many-core systems. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 24–28 June 2018; pp. 1–6.
67. Pourmohseni, B.; Glaß, M.; Teich, J. Automatic operating point distillation for hybrid mapping methodologies. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 1135–1140.
68. Vachhani, V.L.; Dabhi, V.K.; Prajapati, H.B. Survey of multi objective evolutionary algorithms. In Proceedings of the International Conference on Circuit, Power and Computing Technologies (ICCPCT), Nagercoil, India, 19–20 March 2015; pp. 1–9.
69. Knowles, J.; Corne, D. Properties of an adaptive archiving algorithm for storing nondominated vectors. *IEEE Trans. Evol. Comput.* **2003**, *7*, 100–116. [CrossRef]
70. Zitzler, E.; Laumanns, M.; Thiele, L. SPEA2: Improving the strength Pareto evolutionary algorithm. *Eurogen* **2001**, *3242*, 95–100.
71. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
72. Abdou, W.; Bloch, C.; Charlet, D.; Spies, F. Multi-Pareto-ranking evolutionary algorithm. In Proceedings of the European Conference Evolutionary Computation in Combinatorial Optimization, Malaga, Spain, 11–13 April 2012; pp. 194–205.
73. Mitra, T.; Teich, J.; Thiele, L. Time-critical systems design: A survey. *IEEE Des. Test* **2018**, *35*, 8–26. [CrossRef]
74. Li, M.; Liu, W.; Yang, L.; Chen, P.; Chen, C. Chip temperature optimization for dark silicon many-core systems. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2017**, *37*, 941–953. [CrossRef]
75. Hankendi, C.; Coskun, A.K. Scale & Cap: Scaling-aware resource management for consolidated multi-threaded applications. *ACM Trans. Des. Autom. Electron. Syst.* **2017**, *22*, 30.
76. Pourmohseni, B.; Smirnov, F.; Khdr, H.; Wildermann, S.; Teich, J.; Henkel, J. Thermally Composable Hybrid Application Mapping for Real-Time Applications in Heterogeneous Many-Core Systems. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Hong Kong, China, 3–6 December 2019; pp. 220–232.
77. Pagani, S.; Khdr, H.; Chen, J.J.; Shafique, M.; Li, M.; Henkel, J. Thermal safe power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon. *IEEE Trans. Comput.* **2016**, *66*, 147–162. [CrossRef]
78. Khdr, H.; Pagani, S.; Sousa, E.; Lari, V.; Pathania, A.; Hannig, F.; Shafique, M.; Teich, J.; Henkel, J. Power density-aware resource management for heterogeneous tiled multicores. *IEEE Trans. Comput.* **2017**, *66*, 488–501. [CrossRef]
79. Fu, W.; Chen, T.; Wang, C.; Liu, L. Optimizing memory access traffic via runtime thread migration for on-chip distributed memory systems. *J. Supercomput.* **2014**, *69*, 1491–1516. [CrossRef]
80. Ge, Y.; Malani, P.; Qiu, Q. Distributed task migration for thermal management in many-core systems. In Proceedings of the Design Automation Conference (DAC), Anaheim, CA, USA, 13–18 June 2010; pp. 579–584.
81. Liu, Z.; Tan, S.X.D.; Huang, X.; Wang, H. Task migrations for distributed thermal management considering transient effects. *IEEE Trans. Large Scale Integr. Syst.* **2015**, *23*, 397–401.

82. Pourmohseni, B.; Wildermann, S.; Glaß, M.; Teich, J. Predictable run-time mapping reconfiguration for real-time applications on many-core systems. In Proceedings of the International Conference on Real-Time Networks and Systems (RTNS), Grenoble, France, 4–6 October 2017; pp. 148–157.
83. Pourmohseni, B.; Wildermann, S.; Glaß, M.; Teich, J. Hard real-time application mapping reconfiguration for NoC-based many-core systems. *Real-Time Syst.* **2019**, *55*, 433–469. [CrossRef]
84. Pourmohseni, B.; Smirnov, F.; Wildermann, S.; Teich, J. Real-time task migration for dynamic resource management in many-core systems. In *Workshop on Next Generation Real-Time Embedded Systems (NG-RES)*; Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2020; Volume 77, pp. 1–14.
85. Deshwal, A.; Jayakodi, N.K.; Joardar, B.K.; Doppa, J.R.; Pande, P.P. MOOS: A multi-objective design space exploration and optimization framework for NoC enabled manycore systems. *ACM Trans. Embed. Comput. Syst.* **2019**, *18*, 1–23. [CrossRef]
86. Smirnov, F.; Pourmohseni, B.; Glaß, M.; Teich, J. IGOR, get me the optimum! prioritizing important design decisions during the DSE of embedded systems. *ACM Trans. Embed. Comput. Syst.* **2019**, *18*, 1–22. [CrossRef]
87. Sagawa, M.; Aguirre, H.; Daolio, F.; Liefooghe, A.; Derbel, B.; Verel, S.; Tanaka, K. Learning variable importance to guide recombination on many-objective optimization. In Proceedings of the IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), Hamamatsu, Japan, 9–13 July 2017; pp. 874–879.
88. Ito, T.; Aguirre, H.; Tanaka, K.; Liefooghe, A.; Derbel, B.; Verel, S. Estimating relevance of variables for effective recombination. In *International Conference on Evolutionary Multi-Criterion Optimization*; Springer International Publishing: Cham, Switzerland, 2019; pp. 411–423.
89. Rapp, M.; Sagi, M.; Pathania, A.; Herkersdorf, A.; Henkel, J. Power-and cache-aware task mapping with dynamic power budgeting for many-cores. *IEEE Trans. Comput.* **2020**, *69*, 1–13. [CrossRef]
90. Jin, Y. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.* **2005**, *9*, 3–12. [CrossRef]
91. Liu, H.Y.; Carloni, L.P. On learning-based methods for design-space exploration with high-level synthesis. In Proceedings of the Design Automation Conference (DAC), Austin, TX, USA, 2–6 June 2013; p. 50.
92. Beltrame, G.; Fossati, L.; Sciuto, D. Decision-theoretic design space exploration of multiprocessor platforms. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2010**, *29*, 1083–1095. [CrossRef]
93. Wang, H.; Jin, Y.; Sun, C.; Doherty, J. Offline data-driven evolutionary optimization using selective surrogate ensembles. *IEEE Trans. Evol. Comput.* **2018**, *23*, 203–216. [CrossRef]
94. Wang, H.; Jin, Y.; Doherty, J. Committee-based active learning for surrogate-assisted particle swarm optimization of expensive problems. *IEEE Trans. Cybern.* **2017**, *47*, 2664–2677. [CrossRef]
95. Ge, Y.; Qiu, Q.; Wu, Q. A multi-agent framework for thermal aware task migration in many-core systems. *IEEE Trans. Large Scale Integr. Syst.* **2011**, *20*, 1758–1771. [CrossRef]
96. Rapp, M.; Pathania, A.; Mitra, T.; Henkel, J. Prediction-based task migration on S-NUCA many-cores. In Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE), Florence, Italy, 25–29 March 2019.
97. Rapp, M.; Amrouch, H.; Wolf, M.C.; Henkel, J. Machine learning techniques to support many-core resource management: Challenges and opportunities. In Proceedings of the Workshop on Machine Learning for CAD (MLCAD), Canmore, AB, Canada, 3–4 September 2019.
98. Spieck, J.; Wildermann, S.; Teich, J. Run-time scenario-based MPSoC mapping reconfiguration using machine learning models. In Proceedings of the Workshop on Machine Learning for CAD (MLCAD), Canmore, AB, Canada, 3–4 September 2019.
99. Spieck, J.; Wildermann, S.; Schwarzer, T.; Teich, J.; Glaß, M. Data-driven scenario-based application mapping for heterogeneous many-Core systems. In Proceedings of the International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC), Singapore, 1–4 October 2019.
100. Spieck, J.; Wildermann, S.; Teich, J. Scenario-based soft real-time hybrid application mapping for MPSoCs. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 20–24 July 2020.

101. Zeppenfeld, J.; Bouajila, A.; Stechele, W.; Herkersdorf, A. Learning classifier tables for autonomic systems on chip. In *INFORMATIK 2008. Beherrschbare Systeme-dank Informatik. Band 2*; Gesellschaft für Informatik e.V.: Bonn, Germany, 2008.
102. Smirnov, F.; Pourmohseni, B.; Teich, J. Using learning classifier systems for the DSE of adaptive embedded systems. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 957–962.
103. Urbanowicz, R.J.; Moore, J.H. Learning classifier systems: A complete introduction, review, and roadmap. *J. Artif. Evol. Appl.* **2009**. [CrossRef]

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).