This is the authors' version of the work. The definitive work was published in *Proceedings of the 6th IEEE Workshop on Self-Organizing Real-Time Systems (SORT), 2015. This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89).*

Design Methodology and Run-time Management for Predictable Many-Core Systems

(Invited Paper)

Stefan Wildermann, Andreas Weichslgartner, Jürgen Teich
Hardware/Software Co-Design
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Email: {firstname.lastname}@fau.de

***Abstract*—Many-core systems provide a feasible means to build high-performance multi-application systems. They are increasingly exposed to dynamic changes due to varying and online modified application mixes, as well as unavailability of hardware resources due to thermal and power management or faults. Particularly, when applications with real-time requirements are executed, these constraints may become a problem. Here, self-adaptive mechanisms are a means to optimally and feasibly manage such a system at run-time. This paper gives an overview of a design flow for hybrid application mapping for predictable many-core systems. The idea behind this concept is to perform a complex timing analysis and verification of an application during design time and then exploit this information for run-time management (RM) of the system. We propose a novel optimization algorithm to perform RM, which statically calculates a set of application mappings that are guaranteed to fulfill real-time requirements while heuristically optimizing the system objectives. Here, minimization of the energy consumption is exemplary chosen. The algorithm achieves significantly better results than a state-of-the-art approach, especially for systems with a high resource congestion.**

I. Introduction

Modern chips are characterized by a high availability of heterogeneous resources such as processing cores, distributed memories and parallel communication interconnects. Technology enables the design of embedded systems that may integrate more and more functionality while executing varying application mixes on a single chip. But also in safety-critical systems, such as avionics [14] and automotive systems [5], multi- and many-core technology is becoming more important, as the integration of more functionality on a single chip may reduces weight and cost of the electronic system.

Such multi-application systems are increasingly subject to dynamic changes during run-time. First of all, not all applications have to be active all the time but may rather be dynamically started and stopped depending on various operational system modes. This results in different *use cases* of the system, i.e., mixes of concurrently running applications, where the number of use cases theoretically increases exponentially with the number of applications [1]. By inclusion and modification of applications during operation (firmware/software update, application download), new use cases might even only emerge during run-time. This is the case in "open" systems like smartphones, which have to support different communication and multimedia interfaces and standards on the one hand. On the other hand, users of such devices also want to execute multiple applications concurrently. But also for safety-critical systems such as near-future cars [9]. Furthermore, a temporal or even a permanent unavailability of hardware resources is expected to be experienced more often (cf. [12]) because of either hardware faults (manufacturing variability and aging) or due to shutoffs caused by temperature/power management.

The challenge now is that application mapping approaches have to support these dynamic run-time scenarios and online variations, which is not considered in commonly applied design-time mapping techniques. *Self-adaptive mechanisms* have been identified as one of the most promising solution concepts [12]. Here, the system is able to monitor its resources and dynamically re-organize the mapping of applications to the available hardware resources. However, run-time management (RM) of systems which include applications with real-time requirements has to anticipate the impact of the different application mapping options onto the timing characteristics of its applications. Therefore, many sophisticated methodologies follow a *hybrid application mapping* strategy [17]. Based on a specification of each application, different mapping candidates are generated already at design time and evaluated with respect to their resource requirements and obtainable non-functional properties. Of particular importance is that also real-time properties can be verified during this stage. This information is then accessed at run-time to support dynamic application mapping, with the goal to partition the available resources such between currently active applications that one or multiple system objectives (like energy consumption) are optimized while all real-time requirements are fulfilled.

However in general, the temporal characteristics of a mapped application can be influenced by other applications because of resource sharing, so that – in the worst-case – deadlines are violated. In [19], we have proposed a hybrid mapping approach which makes it possible to bound the temporal interference of other applications and provide predictability despite resource sharing. The idea is that a feasible mapping of an application has to fulfill multiple binding and routing constraints for guaranteeing that the application's execution properties, which were analyzed at design time, will actually be satisfied at run-time despite resource sharing. As a consequence, a *constraint satisfaction problem (CSP)* has to be solved during run-time application mapping for being able to give these guarantees. However, this increases the complexity of run-time management significantly. Especially as it may be necessary to not only solve a single CSP per application, but potentially multiple CSPs to test multiple application mapping combinations and find the one that optimizes the system objectives.

Constraint solving has exponential complexity in the worst case (cf. [19]), which restricts the usage for run-time management.

As a major contribution of this paper, we propose a heuristic for efficient determination of optimized, though feasible application mappings to enable self-adaptive run-time management of predictable many-core systems. The paper is outlined as follows. After an overview of the related work in Section II, our design methodology is presented in Section III. In Section IV, the run-time management algorithm is then presented as a novel concept within this methodology. The experimental evaluation in Section V shows the achievable efficiency compared to previous approaches, before concluding the paper in Section VI.

II. Related Work

Application mapping approaches for embedded multi-/many-cores can be classified as design-time mapping, run-time mapping, and hybrid application mapping techniques. However, *design-time mapping* approaches do not support varying applications mixes and dynamic system management, while pure *run-time mapping* approaches cannot guarantee that real-time applications will never miss their deadlines. *Hybrid application mapping* approaches have emerged to enable predictable application execution in dynamic systems with dynamic use cases. Sophisticated design methodologies [2], [13] perform *design space exploration (DSE)* to characterize the impact of different resource allocations on the non-functional properties of mapping and executing an application on them. The outcome of DSE is a set of mapping options referred to as *operating points*, each defined by the required amount and types of resources and the obtainable execution properties. Run-time management can now exploit this information when partitioning the available resources between multiple, concurrently executed applications. This is done by selecting one of the offline analyzed operating points for each application such that the overall objectives are optimized (e.g., minimizing the overall energy consumption) under the constraint that the amount of resources required for mapping this selection does not exceed the amount of available resources. Many related run-time management techniques formulate this problem as a *multi-dimensional multiple choice knapsack problem (MMKP)* and apply ILP solvers [2] or fast knapsack heuristics [22], [15], [20]. While these approaches provide a means for selecting operating points, they generally do not consider the actual mapping of application tasks to resources, and – more severe – the routing of data dependencies over the communication infrastructure. In fact, [19] has shown that such approaches which neglect communication routing are too optimistic and application execution may result in deadline misses.

As a remedy, the hybrid strategy proposed in [16] is based on the idea of using a communication infrastructure which provides *dedicated point-to-point connections* between all pairs of computational resources. This has the major advantage that the usage of such end-to-end connections results in fixed communication latencies between computational resources, and thus supports the verification of real-time guarantees. However, the provision of dedicated connections between all pairs of computational resources is hardly practicable and scalable in envisioned many-core systems that consists of tens, hundreds or even thousand of processing elements (PEs) [3]. Contrary, modern approaches rely on *packet-switched network-on-chips (NoCs)* that follow the idea of partitioning each communication into packets which are routed over shared links [7]. This enhances scalability, but of course makes it harder to give guarantees regarding the communication latency as it requires a communication infrastructure with quality-of-service (QoS) guarantees.

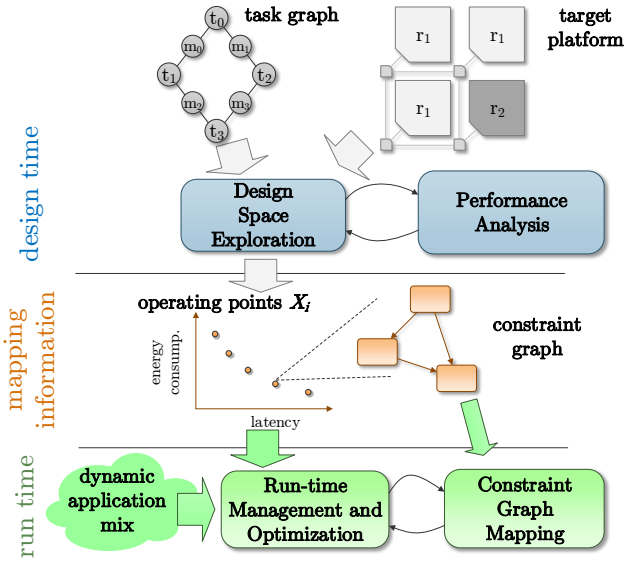


Fig. 1: Design flow of hybrid application mapping for predictable many-core systems according to [19].

The approach in [19] provides a design methodology that enables *predictability* for hybrid mapping of real-time applications by employing such a QoS-driven NoC (details described in the next section). It also applies DSE to generate a set of optimized operating points of an application. For enabling communication-aware application mapping, each of these operating points is associated with a set of additional *binding and routing constraints*, illustrated in Fig. 1. Therefore, [19] proposes a constraint solving algorithm as additional part of the methodology for mapping an application feasibly with respect to the selected operating point. This has the consequence that run-time management cannot be formulated as a MMKP anymore in a straight-forward fashion as constraints specific to each operating point have to be checked when optimizing the mapping of multiple applications. While [19] discusses preliminary strategies for run-time optimization, we investigate this aspect in more detail and develop a novel and highly efficient mapping strategy in this paper.

III. Design Methodology for Predictable Hybrid Application Mapping

Figure 1 illustrated the design methodology from [19], which concepts are summarized in the following.

A. Application Model

Considered are periodic real-time applications from such domains like image/signal processing, control loops, and streaming and multimedia applications. Each such application typically can be expressed by a task graph $G_A(V,E)$. The set of vertices $V = T \cup M$ consists of tasks $t \in T$ that represent concurrent program segments which perform parallel data processing based on messages $m \in M$ exchanged between tasks. Data dependencies are represented by edges $e \in E$. Thus, the task graph represents the application's data flow which is periodically executed with a given period. For real-time applications, the latency for the execution of one period must not exceed a given *deadline*. An example of such a task graph is given in Fig. 2(a).

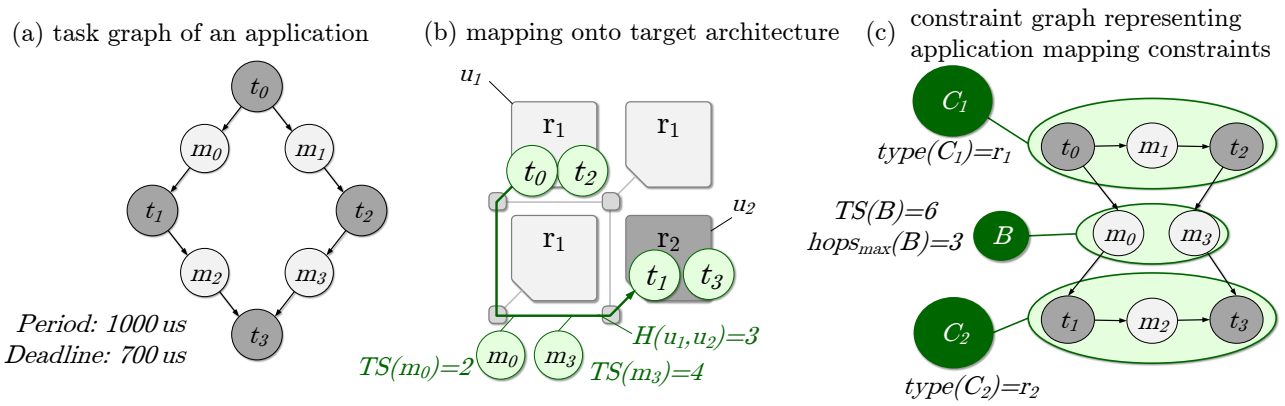


Fig. 2: Examples of (a) task graph, (b) a concrete mapping of this task graph, and (c) the constraint graph corresponding to this mapping. The constraint graph is a means to represent a whole class of possible mappings that correspond to the concrete mapping. It specifies the resource type for each task cluster as well as the bandwidth reservation and the maximal hop distance for each message cluster. The results of performance analysis of the concrete mapping can also be implied on mappings generated according to the constraint graph.

B. Target Platforms

System architectures targeted by our methodology are many-core systems which consist of a set U of heterogeneous PEs. Each unit $u \in U$ is of a certain resource type $r_k \in \{r_1,...,r_m\}$. With m different resource types being available, this is specified by a function $rtype(u) \in \{r_1,...,r_m\}$. PEs are connected by a NoC assuming a 2-dimensional mesh topology. Messages are partitioned into *flits* which are transmitted one after the other over the communication infrastructure. The distance of the route between the sending and receiving PEs u_1 and u_2 is determined by the *hop count* $H(u_1,u_2)$, i.e., the number of routers along the route. In order to verify any real-time requirements of a communication over such a route, upper bounds for the transmission latency are a prerequisite. However, it is a necessity to compute this upper bound without having to consider every possible interference with other communications. Here, the concept of *composabilty* offers a viable solution. In composable NoCs with QoS, e.g., [10] or [11], an upper bound for the worst-case communication latency can be calculated for a given bandwidth reservation.

For the bandwidth reservation scheme proposed in [11], an amount of TS_{max} time slots for accessing each communication link of a given NoC are periodically scheduled between messages routed over this link. For transmission of a message m, a number of time slots $TS(m)$ can be reserved within this *arbitration interval* of size TS_{max}. In contrast to a global *time division multiple access (TDMA)* scheme the position of these time slots in the arbitration interval does not correspond to a global schedule but only to the local schedule for the individual link. Therefore, the worst-case latency for transmitting a message m consisting of f flits can be bounded according to the following equation [19]:

$$CL_m = (f + H(u_1,u_2)\cdot\Delta_R) + \left(\left\lceil \frac{f}{TS(m)} \right\rceil + H(u_1,u_2)\right)\cdot(TS_{max} - TS(m)), \quad (1)$$

where Δ_R is the delay of one router. The message's communication latency will always stay within this *bound* if the hop distance between sender and receiver is at most $H(u_1,u_2)$ and the amount of time slots reserved on each link for transmitting m is at least $TS(m)$.

C. Design Space Exploration

For these application and architecture assumptions, a DSE coupled with a formal performance analysis [19] delivers resource allocations and task mappings that are (i) verified with respect to real-time requirements and (ii) optimized with respect to additional objectives specified by the programmer (e. g. regarding execution time, power consumption, resource utilization). In this approach, a vast number of different mappings can be evaluated according to different optimization criteria. For each of the mappings (an exemplary mapping is depicted in Fig. 2(b)) a formal analysis (cf. [19]) calculates the *worst-case end-to-end latency*. This is composed of the worst-case execution latencies of the tasks and the worst-case communication latencies of the message. However, in contrast to traditional DSEs, only one application is considered together with the target architecture for the exploration. Yet, when applying techniques from *invasive computing* [18], which allow to dynamically claim a set of PEs for the exclusive usage of an application, and by employing a NoC with QoS, we enable *composability* to guarantee that the *worst-case bounds* determined by static performance analysis are still valid when multiple individually analyzed applications are mapped together during run-time.

D. Application Mapping

The result of the design-time analysis is a set X_i of Pareto-optimal mappings which is handed over to the RM. The mappings $x_i \in X_i$ are called *operating points*. To satisfy the real-time requirements of such an operating point, the RM has to find an application mapping $\beta(x_i)$ according to the resource reservations used during the offline analysis of x_i. The corresponding constraints for task binding and message routing are encoded as a *constraint graph*. A constraint graph $G_C(x_i)$ is a bipartite graph, like the task graph, and consists of task clusters C, message clusters B, and edges. A task cluster is formed by all tasks mapped to the same PE and has the required resource type of the PE annotated. All messages which are routed from a common source to a common destination are combined to a message cluster B, which is annotated with the maximal hop distance and their accumulated amount of time slots, i.e., $\sum_{m\in B} TS(m)$, for bandwidth reservation. As an example, consider the mapping shown in Fig. 2(b). The tasks t_0 and t_2 are mapped to a resource of type r_1, while tasks t_1 and

Algorithm 1: Backtracking algorithm, proposed in [19], for solving the CSP related to finding a feasible mapping of a given constraint graph G_C.

```
1  backtrack(A, G_C, G_NoC)
2  if (A is complete) then
3      return A;
4  C = selectNextTaskCluster(G_C);
5  D_C = getFeasibleUnits(C, G_C, G_NoC);
6  for each (u_C ∈ D_C) do
7      L_C = routeMessageClusters(C, u_C, G_C, G_NoC);
8      if u_C enables feasible binding and routing L_C then
9          A' = backtrack(A ∪ ⟨C,u_C,L_C⟩, G_C, G_NoC);
10         if (A' ≠ ∅) then
11             return A';
12 return ∅;
```

t_3 are mapped to a PE of type r_2. The messages m_0 and m_3 are routed over three hops between these two PEs. The constraint graph corresponding to this mapping can be seen in Fig. 2(c). It consists of the task cluster C_1, C_2 and the message cluster B.

The goal of application mapping is to find a mapping that does not violate the constraints represented by the constraint graph. In [19], a *backtracking algorithm* (Algo. 1) has been proposed to solve this CSP. The algorithm starts with an empty assignment $\mathcal{A}=\emptyset$. Then, the next task cluster C to be processed is selected in line 4. In line 5, all *available* (i.e., free and non-faulty) PEs which are of the type $type(C)$ are selected to build the domain D_C. Then in line 7, routing is performed for each of C's message clusters which sending or receiving communication partners are already assigned. By applying deterministic routing algorithms, like xy routing, the route can be directly related to the positions of the sender and the receiver. For each route, it is tested whether its hop distance lies within the specified maximum and whether there are sufficient free time slots for bandwidth reservation on each link along the route. When binding and routing are feasible, the algorithm is called recursively in line 9. Only when all task clusters are assigned, a feasible mapping is returned, which is guaranteed to fulfill the real-time requirements analyzed during design time. While this enables the mapping of a single operating point, we consider the run-time management of multiple application mixes in the next section.

IV. Run-time Management

So far, the design flow has been presented for a single application only. Once multiple individually designed applications shall be executed concurrently on the same system, it becomes desirable to partition the available resources optimally between them. This task has to be performed by the run-time management (RM). Besides the binding and routing constraints, each operating point x_i, explored at design time, is also annotated with quality numbers for a number of non-functional properties. In the remainder of this paper, we exemplary select the *energy consumption* $e(x_i)$, so that the goal of RM is to minimize the system's overall energy consumption. But note that the presented approach can also be applied for arbitrary objective functions $\tilde{e}(x_1,...,x_n)$, where n is the number of simultaneously executing applications.

Basically, run-time management consists of selecting one operating point per application such that the overall energy consumption is minimized. This is subject to that a sufficient amount of free and non-faulty resources is available and each operating point can be mapped according to the constraints represented by its constraint graph. As the real-time guarantees are already verified for each operating point, they do not have to be tested anymore. This optimization problem is formalized as follows:

$$\text{minimize} \quad \sum_{i=1}^{n} e(x_i) \tag{2a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_k(x_i) \leq \theta_k, \quad k=1,...,m \tag{2b}$$

$$C(\beta(x_i), G_C(x_i)) \leq 0, \quad i=1,...,n \tag{2c}$$

$$C_{sys}(\beta(x_1),...,\beta(x_n)) \leq 0. \tag{2d}$$

In the constraint in Eq. (2b), $r_k(x_i)$ specifies the amount of instances from resource type k required to map x_i. It ensures that the overall amount of resource instances required for mapping all n applications does not exceed the overall available amount θ_k per resource type r_k. Eq. (2c) expresses that a feasible mapping $\beta(x_i)$ has to be determined per selected operating point x_i which fulfills the set of constraints C associated with its constraint graph $G_C(x_i)$. Finally, Eq. (2d) formalizes that all applications have to be mapped in such a way that the set of system constraints C_{sys} are fulfilled, i.e., no PE is assigned to more than one application and the capacity of TS_{max} time slots is not exceeded on any NoC link.

The above optimization problem is NP-hard [19]. In the following, we provide a heuristic for solving it. It consists of two steps. In the first step, the problem is considered as a *multi-dimensional multiple choice knapsack problem (MMKP)* by ignoring the constraints in Eqs. (2c) and (2d), and it is then solved by applying *Lagrangian relaxation*. In the second step, the applications are mapped incrementally by applying the constraint solving algorithm Algo. 1. The order in which the applications are mapped as well as the order in which their operating points are tested significantly influences the outcome of this incremental mapping procedure. A contribution of this paper is that we apply the result from Lagrangian relaxation to determine good orders. Both steps are described in the next sections.

A. Optimization of Knapsack Problem

In the first step, the constraints related to application mapping (Eqs. (2c) and (2d)) are not considered. The resulting optimization problem is a *MMKP*, i.e.: Select one operating point for each application (*multiple choice*) such that this selection minimizes the overall energy consumption while the amount of instances required per resource type k does not exceed the overall available amount θ_k (*multi-dimensional knapsack*). This problem is still NP-hard [22], but several heuristics are available for finding optimized operating point selections. Our approach relies on the algorithm from [20] which applies *Lagrangian relaxation* and is summarized in the following.

Lagrangian relaxation works by solving the dual optimization problem which relaxes the hard constraints from the original optimization problem. It is based on the *Lagrangian function*, which is formulated for the MMKP according to

$$\mathcal{L}(x,\lambda) = \sum_{i=1}^{n} e(x_i) + \sum_{k=1}^{m} \lambda_k \cdot (r_k(x_i) - \theta_k), \tag{3}$$

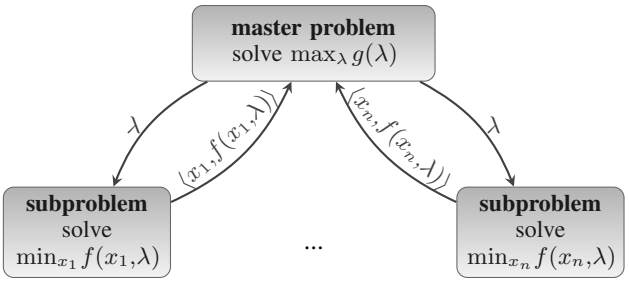


Fig. 3: Illustration of solving the Lagrangian relaxation of the MMKP by decomposing it into a master problem and into several subproblems.

with $x = (x_1, ..., x_n)$ and $\lambda = (\lambda_1, ..., \lambda_m)$. In this function, the constraints are combined with the objective function by a weighted sum, where weights λ_k are denoted as *Lagrangian multipliers*.

The *dual function* is defined as the minimum value of the Lagrangian function over x:

$$\begin{aligned} g(\lambda) &= \min_x \mathcal{L}(x,\lambda) \\ &= \sum_{i=1}^{n} \min_{x_i} \underbrace{\left\{ e(x_i) + \sum_{k=1}^{m} \lambda_k \cdot r_k(x_i) \right\}}_{f(x_i,\lambda)} - \sum_{k=1}^{m} \lambda_k \cdot \theta_k. \end{aligned} \quad (4)$$

Here, we define the function $f(x_i,\lambda)$ which can be interpreted as the *cost* of application i for selecting operating point x_i [4], where the Lagrangian multipliers specify the *price* per resource:

$$f(x_i,\lambda) = e(x_i) + \sum_{k=1}^{m} \lambda_k \cdot r_k(x_i). \quad (5)$$

This is the basis for defining the *dual optimization problem*:

$$\begin{aligned} &\text{maximize} && g(\lambda) \\ &\text{subject to} && \lambda \geq 0. \end{aligned} \quad (6)$$

This problem can be solved by applying a *subgradient method*, as described in [20], by decomposing the problem into a master problem and several subproblems (cf. Fig. 3). The master problem adapts the multipliers λ according to the subgradient method, while a subproblem is solved for each application i by determining its minimal cost $\min_{x_i \in X_i} f(x_i,\lambda)$ for the Lagrangian multipliers provided by the master problem. The outcome of each subproblem is used by the master problem to adapt the Lagrangian multipliers, and the approach is repeated until a termination criterion is reached (usually a maximal number of iterations). This algorithm can be speeded up up by solving the subproblems in a distributed fashion, as investigated in [20].

The solution of the dual problem is denoted by $\lambda^* = (\lambda_1^*, ..., \lambda_m^*)$. Furthermore, the selected operating point of each application i is given by

$$x_i^* = \arg\max_{x_i \in X_i} f(x_i,\lambda^*). \quad (7)$$

The selection $x^* = (x_1^*,...,x_n^*)$ does not necessarily have to be an optimal or even feasible solution of the MMKP because of the relaxation of the resource constraints. Also the mapping constraints of the original optimization problem in Eq.(2) are not considered at all. However, in the following, we present a heuristic for run-time management that relies on the optimized Lagrangian multipliers λ^*.

B. MMKP-based Application Mapping

The goal of our proposed run-time application mapping approach is to determine a heuristically optimized solution for the original problem. This means that besides the selection of operating points x_i which minimize the energy consumption, this step also requires to perform the application mapping $\beta(x_i)$ for each application i, $1 \leq i \leq n$ and testing the respective mapping constraints. Feasible mappings $\beta(x_i)$ can be determined by constraint solving based on the presented backtracking algorithm (Algo. 1).

Therefore, a straight-forward idea for run-time application mapping would be to use the outcome x^* of the MMKP optimization and test the constraints for this selection by performing backtracking for their combined constraint graphs, i.e., $\texttt{backtrack}(\emptyset, G_C(x_1^*) \circ ... \circ G_C(x_n^*), G_{NoC})$. However, x^* is related to the solution of the dual problem of the MMKP, which does (a) relax the resource constraints and (b) not consider the mapping constraints. So, with a high probability, no feasible mapping might actually exist for the solution x^*. This has two severe implications. First, time for performing the backtracking algorithm grows exponentially with the number of tasks clusters. So, when we combine the constraint graphs of all applications, we will also severely increase the execution time of the algorithm. Second, when the backtracking algorithm returns, no feasible mapping of any application will have been determined in many cases, as the algorithm either finds a mapping for all applications according to x^* or for none. Both implications are experimentally validated in [19].

As a remedy, [19] proposes a mapping heuristic where the set of operating points X_i of each application i are sorted according to their objective value, in our case the energy consumption in non-descending order. Applications are then mapped incrementally by subsequently testing the operating points in the sorted order. While [19] has shown improvements in both execution time and success rate of this approach compared to trying to mapping the MMKP result, it has the problem that

(a) there is no rule for optimizing the order in which the applications are mapped incrementally and

(b) it does not consider that some resource types are scarcer than others.

In the following, we present an algorithm which uses the result $\langle \lambda^*, x^* \rangle$ from Lagrangian relaxation of the MMKP and the cost function in Eq. (5). Algo. 2 gives the details. After solving the MMKP (line 1), the applications are sorted non-decreasing according to the costs of their results (line 2). They are then incrementally mapped in this order (lines 3 to 8). Therefore, the operating points are sorted in non-decreasing order according to their costs for the given Lagrangian multipliers λ^*. In this order, the backtracking algorithm is applied until a feasible mapping is determined for one operating point (line 8) or all points are tested.

V. Experimental Evaluation

For the experimental evaluation of the proposed methodology, we used the E3S benchmark [8]. It contains the tasks graphs of several embedded real-time applications like automotive (18

Algorithm 2: Algorithm for MMKP-based application mapping. The Lagrangian multipliers λ^* and the corresponding selection of operating points x^* are used to sort the applications and their operating points. This determines the order in which applications are incrementally mapped by applying the backtracking algorithm (Algo 1).

```
⟨λ*,x*⟩ = solve Lagrangian relaxation of MMKP;
Sort applications i=1,...,n according to f(x_i^*,λ*) in
    non-decreasing order so that
        f(x_i^*,λ*) ≤ f(x_j^*,λ*), ∀j>i;
for i=1 to n do
    Sort item x_i ∈ X_i according to f(x_i,λ*) in
    non-decreasing order so that
        f(x_i^(ℓ),λ*) ≤ f(x_j^(o),λ*), ∀o>ℓ;
    ℓ=1;
    β(x_i)=∅;
    while β(x_i)=∅ and ℓ≤|X_i| do
        β(x_i) = backtrack(∅, G_C(x_i^(ℓ)), G_NoC);
```

tasks), telecommunication (14 tasks), consumer (11 tasks) and networking (7 tasks). The values for energy consumption and worst-case execution time (WCET) of the tasks as well as the bandwidth requirements of messages reflect realistic scenarios of current embedded multiprocessor system-on-chips (MPSoCs). The communication energy consumption was calculated by models proposed in [6], [21]. The methodology was applied for heterogeneous NoC-based architectures composed of three different processor types (from [8], including an IBM Power PC and variants of AMD K6). We tested a 5×5, a 6×6, and a 10×10 NoC layouts.

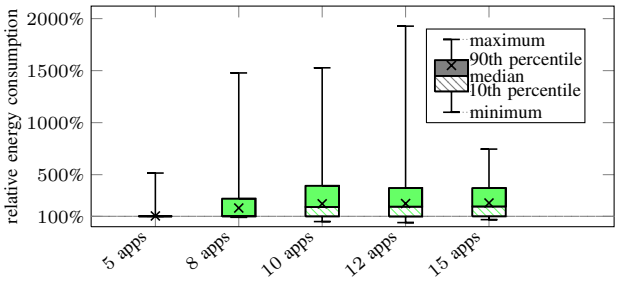
We applied the DSE described in Section III to determine optimized mappings for each application. Those (near) Pareto-optimal solutions which do not violate an individual application deadline determine the set of operating points $x_i \in X_i$ together with their constraint graphs $G_C(x_i)$ and the values of the evaluated energy consumption $e(x_i)$. For each application, the evaluation resulted in less than 100 operating points with verified real-time properties.

For the evaluation of the run-time management, we evaluated application mixes consisting of different amounts n of these applications. We randomly generated and tested 500 different combinations per application mix size. Each mix was mapped by applying the incremental mapping strategy from [19] (*incremental mapping*) and the mapping strategy proposed in this paper in Algo. 2 (*MMKP-based mapping*).

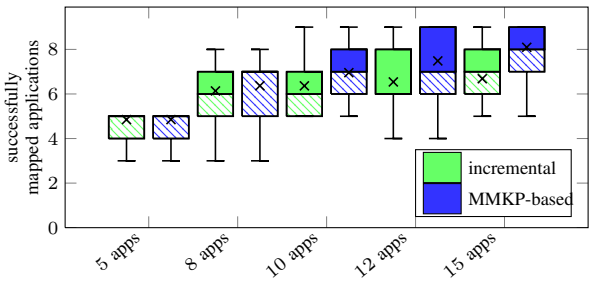
A. Application Mapping Results

For each test case, we measured (a) the amount of applications from the mix which were successfully mapped and (b) the overall energy consumption. Let $E_{inc}(j)$ be the energy consumption obtained for incremental mapping and $E_{MMKP}(j)$ the energy consumption obtained for the proposed approach in the j-th test case. Then throughout this section, we will present the relative energy consumption obtained for incremental mapping put into relation to the result of the proposed mapping algorithm, i.e.,

$$E_{rel}(j) = \frac{E_{inc}(j)}{E_{MMKP}(j)} \cdot 100\%. \quad (8)$$



(a) relative overall energy consumption E_{rel}



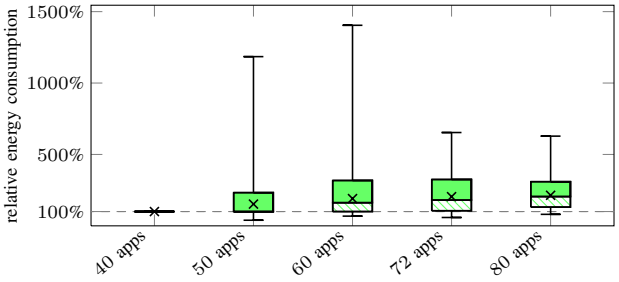
(b) number of successfully mapped applications

Fig. 4: Box plots summarizing the results of mapping application mixes of different sizes onto a 4×4 NoC.

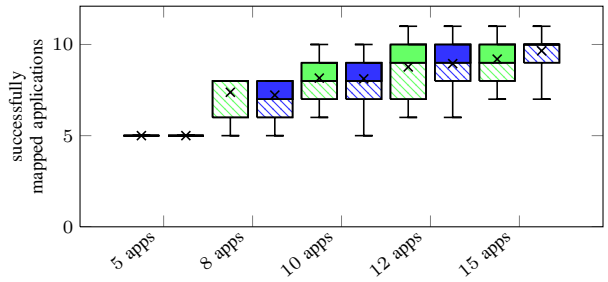
The results of the experiments for the 4×4 NoC is shown in Fig. 4 summarizing the 500 test cases per application mix size. The box plots indicate minimum, maximum, 10th, 50th, and 90th percentile of the results, and the mark (x) represents the average result. The results of the relative energy consumptions in Fig. 4(a) show that the energy consumption obtained by incremental mapping significantly exceeded the energy consumption obtained by the proposed approach in more than 90% of the test cases. For application mixes with 10 or more applications, the energy consumption of incremental mapping was even more than twice as high as for MMKP-based mapping ($E_{rel}(j) > 200\%$) in more than 50% of the test cases.

Commonly, mapping of fewer applications will also result in a lower overall energy consumption. Therefore, it is additionally necessary to evaluate how many applications were successfully mapped by both approaches. Only then is it possible to conclude whether the reduction of energy consumption is not only to the expense of a lower number of applications being actually executed. Fig. 4(b) summarizes the number of applications which were successfully mapped throughout the experiments. Not in all cases it was possible to map all applications due to the limited availability of resources and binding and routing constraints. An advantage of the proposed RM is, however, that it recognizes that an application cannot be executed without violating its real-time requirements, and does not start it at all. Both approaches performed almost similar for test cases with 5 applications. However, the proposed algorithm was able to feasibly map more applications in a higher amount of the remaining cases. Particularly for 8, 10, and 15 applications, 50% of the MMKP-based mapping results were significantly better than almost 90% of the incremental mapping results.

The results for the 5×5 NoC architecture are shown in Fig. 5. Also here, as indicated by the relative energy consumption summarized by Fig. 5(a), the energy consumption of the incremental mapping approach exceeded the energy

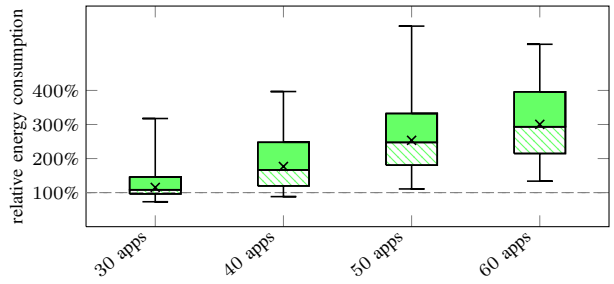


(a) relative overall energy consumption E_{rel}

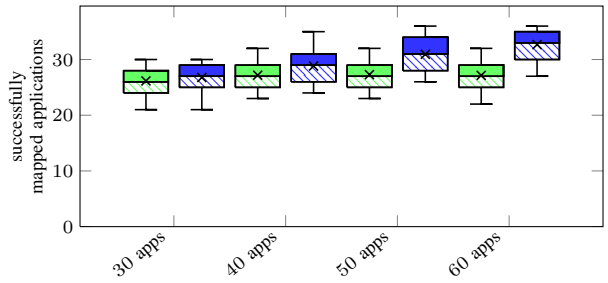


(b) number of successfully mapped applications

Fig. 5: Box plots summarizing the results of mapping application mixes of different sizes onto a 5×5 NoC.



(a) relative overall energy consumption E_{rel}



(b) number of successfully mapped applications

Fig. 6: Box plots summarizing the results of mapping application mixes of different sizes onto a 10×10 NoC.

consumption of the proposed approach in more than 90% of the test cases. Fig. 5(b) summarizes the number of successfully mapped applications of both approaches in the experiments. While for test cases with 8 applications, incremental mapping was able to feasibly map more applications in a higher amount of test cases, the proposed mapping algorithm had a comparable (5 and 10 applications) or even better overall mapping success rate (12 and 15 applications) than the existing approach.

The superiority of the proposed approach becomes even more significant when increasing the NoC size to a 10×10 layout and the number of applications, shown in Fig. 6. MMKP-based mapping produced significantly higher energy consumption in almost all test cases. The proposed algorithm was also able to feasibly map a significantly higher amount of real-time applications, despite the increased congestion on the 10×10 NoC.

The experiments show that the proposed MMKP-based mapping heuristic performs better particularly then when there is a higher congestion in the system, i.e., only a limited amount of resources is available for executing multiple real-time applications. Particularly in these cases, MMKP-based mapping is able to map a higher amount of applications even at a lower energy consumption compared to incremental mapping.

B. Execution Times

Fig. 7 shows the average overall execution times of both approaches for the above experiments with solid lines. Experiments were carried out on an Intel i7-3770. The execution time obviously increases with the number n of applications and with the size of the NoC. The average execution time of the presented approach stays below that of incremental mapping until a certain application mix size from which it raises above that of incremental mapping (i.e, (a) $n\approx12$, (b) $n\approx15$, (c) $n\approx42$). The dashed lines in Fig. 7 show the average execution times spent during application mapping for searching feasible mappings of constraint graphs for which no valid mapping actually existed in the NoC. This significantly took most of the time in both algorithms. Solving the Lagrangian relaxation in MMKP-based mapping and sorting was done in the negligible remaining time. Fig. 8 shows the average execution times it took for applying the backtracking algorithm (Algo. 1) for testing one constraint graph for which no valid mapping existed. The interesting observation is that, for incremental mapping, the time for verifying that no feasible mapping exits decreases with the number of applications. Whereas for MMKP-based mapping, this time increases with the number of applications.

To summarize, the incremental mapping of the MMKP-based algorithm generates mappings where in average more real-time applications can be mapped. However, the task of verifying unsatisfiability for constraint graphs that cannot be mapped becomes increasingly complex. Further investigations of the reasons of this phenomenon and possible means to improve the scalability are future work.

VI. Conclusion

This paper investigated the design of predictable many-core systems under dynamic workload scenarios. Particularly, when the active application mix dynamically changes or in case of unavailability of resources, self-adaptive run-time management techniques have to be applied, to partition the available resources optimally between real-time applications. We described a design flow of a hybrid application mapping methodology and presented a novel application mapping algorithm. The experimental evaluation demonstrated that the algorithm is more efficient in determining optimized, though feasible application mapping than a state-of-the-art algorithm. The improvements become particularly obvious when the system's resource congestion increases.

The results have also shown that the execution time of the proposed RM algorithm increases with the NoC size and the number of applications. Therefore, techniques for decentralizing

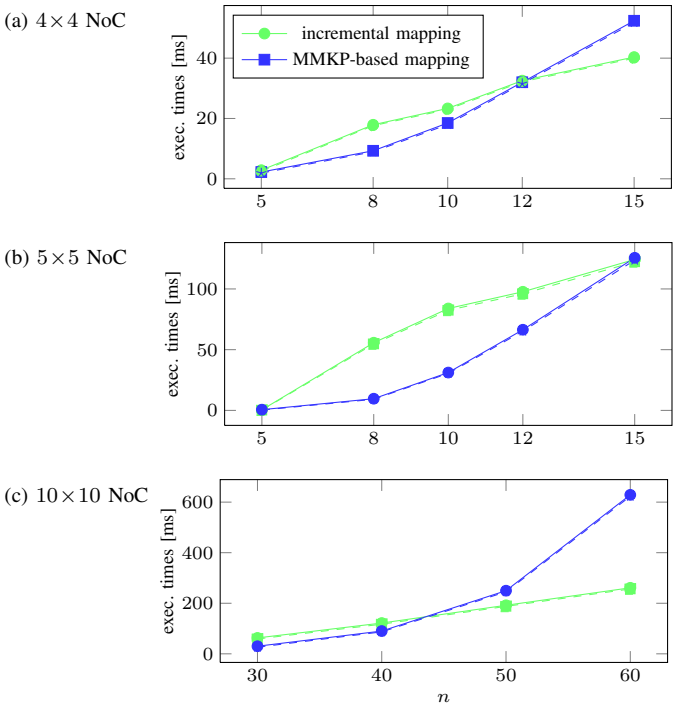


Fig. 7: Average execution times of application mapping for the different application mix sizes (n) and NoC sizes. The dashed lines indicate the overall execution time spent for searching feasible mappings of constraint graphs for which no valid mapping existed in the system. This time dominates the execution time of both algorithms.

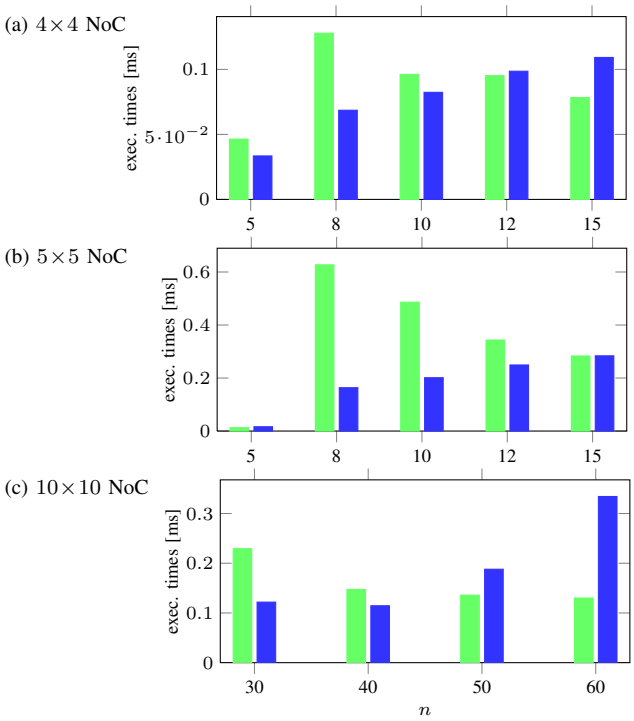


Fig. 8: The average execution times for applying the backtracking algorithm (Algo. 1) on a constraint graph for which no valid mapping existed for the experiments with incremental mapping and MMKP-based mapping and the different application mix sizes (n) and NoC sizes.

run-time management by decomposing the algorithm are planned as future work.

Acknowledgment

This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Invasive Computing" (SFB/TR 89).

References

[1] B. Akesson, A. M. Molnos, et al. Composability and predictability for independent application development, verification, and execution. In *Multiprocessor System-on-Chip*, pp. 25–56. Springer Verlag, 2011.

[2] E. Bini, G. Buttazzo, et al. Resource management on multicore systems: The ACTORS approach. *Micro, IEEE*, 31(3):72–81, 2011.

[3] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of Design Automation Conference (DAC)*, pp. 746–749. 2007. ISBN 978-1-59593-627-1.

[4] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

[5] M. Broy. Challenges in automotive software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pp. 33–42. 2006. ISBN 1-59593-375-1.

[6] C.-L. Chou, U. Ogras, et al. Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels. *TCAD*, 27(10):1866–1879, 2008.

[7] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings*, pp. 684–689. 2001. ISSN 0738-100X.

[8] R. Dick. Embedded system synthesis benchmarks suite (E3S), 2010. http://ziyang.eecs.umich.edu/dickrp/e3s/.

[9] S. Fries and R. Falk. Electric vehicle charging infrastructure–security considerations and approaches. In *INTERNET 2012, The Fourth International Conference on Evolving Internet*, pp. 58–64. 2012.

[10] K. Goossens and A. Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp. 306–311. June 2010. ISSN 0738-100X.

[11] J. Heisswolf, R. König, et al. Providing multiple hard latency and throughput guarantees for packet switching networks on chip. *Computers & Electrical Engineering*, 9(8):2603 – 2622, 2013.

[12] J. Henkel, L. Bauer, et al. Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *DAC*, pp. 99:1–99:10. 2013.

[13] G. Mariani, V. Sima, et al. Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures. In *DATE*, pp. 1379–1384. 2012.

[14] J. Rushby. Partitioning for safety and security: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, Jun. 1999.

[15] H. Shojaei, T. Basten, et al. A fast and scalable multidimensional multiple-choice knapsack heuristic. *TODAES*, 18(4):51:1–51:32, 2013.

[16] A. K. Singh, A. Kumar, et al. Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs. *TODAES*, 18(1):9:1–9:29, 2013.

[17] A. K. Singh, M. Shafique, et al. Mapping on multi/many-core systems: Survey of current and emerging trends. In *DAC*, pp. 1–10. 2013.

[18] J. Teich. Invasive algorithms and architectures. *it - Information Technology*, pp. 300–310, 2008.

[19] A. Weichslgartner, D. Gangadharan, et al. DAARM: Design-time application analysis and run-time mapping for predictable execution in many-core systems. In *CODES*, pp. 34:1–34:10. 2014.

[20] S. Wildermann, M. Glaß, et al. Multi-objective distributed run-time resource management for many-cores. In *DATE*, pp. 1–6. 2014.

[21] P. Wolkotte, G. J. M. Smit, et al. Energy model of networks-on-chip and a bus. In *SOCC*, pp. 82–85. 2005.

[22] C. Ykman-Couvreur, V. Nollet, et al. Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In *SOCC*, pp. 1–4. 2006.