

Article

Exact Design Space Exploration Based on Consistent Approximations

Kai Neubauer * , Benjamin Beichler  and Christian Haubelt 

Applied Microelectronics and Computer Engineering, University of Rostock, 18051 Rostock, Germany; benjamin.beichler@uni-rostock.de (B.B.); christian.haubelt@uni-rostock.de (C.H.)

* Correspondence: kai.neubauer@uni-rostock.de

Received: 14 May 2020; Accepted: 24 June 2020; Published: 27 June 2020



Abstract: The aim of design space exploration (DSE) is to identify implementations with optimal quality characteristics which simultaneously satisfy all imposed design constraints. Hence, besides searching for new solutions, a quality evaluation has to be performed for each design point. This process is typically very expensive and takes a majority of the exploration time. As nearly all the explored design points are sub-optimal, most of them get discarded after evaluation. However, evaluating a solution takes virtually the same amount of time for both good and bad ones. That way, a huge amount of computing power is literally wasted. In this paper, we propose a solution to the aforementioned problem by integrating efficient approximations in the background of a DSE engine in order to allow an initial evaluation of each solution. Only if the approximated quality indicates a promising candidate, the time-consuming exact evaluation is executed. The novelty of our approach is that (1) although the evaluation process is accelerated by using approximations, we do not forfeit the quality of the acquired solutions and (2) the integration in a background theory allows sophisticated reasoning techniques to prune the search space with the help of the approximation results. We have conducted an experimental evaluation of our approach by investigating the dependency of the accuracy of used approximations on the performance gain. Based on 120 electronic system level problem instances, we show that our approach is able to increase the overall exploration coverage by up to six times compared to a conservative DSE whenever accurate approximation functions are available.

Keywords: system synthesis; approximation; answer set programming; background theory

1. Introduction

The design of embedded systems is continuously becoming more arduous as complex applications have to be mapped onto heterogeneous hardware platforms. In order to optimally exploit parallel structures for concurrent execution, a vast number of possible mapping options have to be evaluated and compared. Besides good performance of the resulting system implementation, further objectives like energy requirements, monetary costs, and reliability are typically conflicting with each other so that a single optimal solution, which dominates (i.e., evaluates better for all objective) all other design points, does not exist. Instead, a set of Pareto optimal, mutually non-dominated design points (Pareto front) is obtained that render the best compromise solutions to a given problem.

In order to obtain the Pareto front, a design space exploration (DSE) including a multi-objective optimization is executed which can be conceptually split into a search for feasible solutions, an evaluation, and an optimization of found solutions. As depicted in Figure 1, the search is performed in parameter space by filtering infeasible solutions from the set X of all solutions. The set of feasible design points X_F is then converted into objective space by evaluating all designs w.r.t. to the desired objective and constraint functions. The optimization step first removes invalid designs, that is, solutions

that do not fulfill specified requirements, resulting in the set of valid solutions X_V . Finally, after the Pareto filter, only Pareto optimal design points remain in the set X_P .

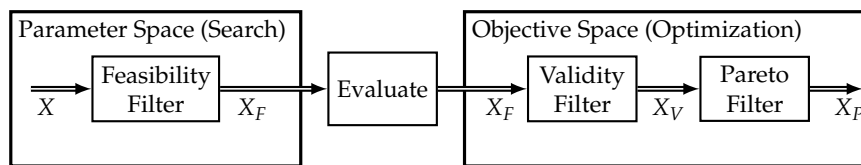


Figure 1. Design space exploration (DSE) as a filtering process.

Due to the sheer size of the search spaces of real world design, the enumeration of sets X_F , X_V , and X_P is not viable. Thus, a search engine is used to iteratively present new design candidates to the feasibility filter. If a candidate x_c passes this check, it will be evaluated, validated and finally compared to previously identified *good* solutions stored in X_P . Afterwards X_P is updated based on this comparison. If the candidate x_c is dominated by any solution in X_P , it will be discarded. However, if x_c dominates solutions in X_P , these solutions will be removed from X_P and x_c is added. The candidate is also added to X_P if it is incomparable to all solutions in X_P . As a consequence, with increasing exploration time, the set X_P changes and converges towards the true Pareto front.

While designing complex embedded systems, the evaluating step is typically the most time-consuming one. Even worse, it is frequently executed as it has to be conducted independent of the validity and optimality of a feasible solution. Hence, diminishing the evaluation time without deteriorating the exploration quality would increase the overall exploration performance and significantly improve the applicability of the DSE.

One possibility to accelerate the evaluation is the use of approximations. That is, instead of costly calculating the precise objective value of a design point, a rough estimation is performed. Due to its lower complexity, the estimation executes in a fraction of the time necessary for the exact calculation. However, the result of an optimization which only utilizes approximate evaluations might differ from the exploration result obtained from exact evaluations. To overcome this drawback, we propose to combine approximations and exact evaluations. That is, the quality of a design point is only calculated exactly if the approximation already promises good results. This way, the performance improvement of approximations can be coupled with an accurate optimization process.

In this work, we focus on the integration of approximations into a state-of-the-art DSE approach. As a search engine, we utilize the answer set programming (ASP) solver clingo that has been shown to perform especially well for determining feasible mapping and routing decisions [1]. Furthermore, clingo contains a rich interface for defining arbitrary background theories following the answer set programming modulo theories (ASPmT) paradigm [2]. These background theories are subsequently integrated directly into the solving process. This allows us to tightly integrate the approximations into the DSE and utilize its results to further improve the conflict analysis of the ASP solver. To summarize, our contribution is threefold:

1. We propose a DSE methodology that diminishes exploration time by using approximated evaluations in a way that the correctness of the obtained Pareto is guaranteed when the entire search space has been explored. Even if only a part of the search space is explored, we can still ensure that no optimal solutions are removed after they have been found.
2. We integrate the proposed methodology into a state-of-the-art ASP-based DSE approach utilizing the ASPmT interface of clingo. This way, the search engine can be directly informed about the reason why a design candidate did not pass the optimality checks. Thus, the search can be pruned efficiently.
3. We investigate the performance of our approach on the basis of 120 test instances on the electronic system level. Therefore, we compare our approach with a reference DSE utilizing SystemC simulations of the underlying hardware architecture implementing a network-on-chip.

2. Related Work

As demands on embedded systems are consistently increasing, much research on efficient design space exploration techniques has been conducted throughout the past two decades. Different DSE approaches have recently been classified in a survey by Pimentel [3] into two types: (Meta-)heuristics utilizing population-based optimizations like evolutionary algorithms [4] and formal methods such as integer-linear-programming (ILP) and boolean satisfiability (SAT), for example, References [5,6]. In this paper, however, we focus on the use of approximations in DSE. Therefore, we will propose a different classification of state-of-the-art DSE approaches with respect to the use of approximations in the DSE.

In general, there is no DSE approach that guarantees obtaining the true Pareto front of highly complex embedded systems in a viable amount of time. Hence, each approach tries to find a set of implementations that approximates the true Pareto set as good as possible. It is important to state that the sense of the term approximation is not used consistently throughout literature. Two different interpretations can be identified: search-related and evaluation-related approximations.

In the former, the central aim is to steer the search into regions of the parameter space where they expect Pareto optimal solutions. Representatives of works in these search-related approximations are References [7–14]. ReSPIR [7] and MULTICUBE [8] use the technique design of experiment (DoE) to choose appropriate sample configurations for simulating a multi processor system on chip (MPSoC). The results of the simulations are then used to create an approximation of a Pareto front with response surface modeling (RSM). From this RSM, new configurations for simulations are derived to incrementally enhance the approximation of the Pareto front. Contrary, we approximate the actual result of the simulation, to avoid costly simulation. Thereby, we also use the DoE technique as we are able to efficiently eliminate bad design candidates. However, our candidate choice is not steered by the shape of the current Pareto front approximation. The authors of References [9,10] present comparable approaches by utilizing stochastic kriging, a stochastic meta-model simulation, to select potentially good regions of the design space to search for new solutions. The results show that it finds up to 91% of all true Pareto points of common multi-objective optimization problems. Instead of stochastic simulations, the authors of Reference [14] propose the use of generic and easily computable guiding functions that steer the search into promising regions of the design space. Compared to a reference approach running over a period of two weeks, they obtain a 20% better Pareto front approximation within four days. In Reference [11], the search time is reduced by using a heuristic that prunes the search of potentially inferior regions. Thus, expensive evaluations can be reduced to a minimum. With their heuristic, the authors can achieve a speedup of up to 80 while simultaneously maintaining the quality of Pareto front approximations compared to a state-of-the-art MOEA. Liu et al. [12] present an approximation technique that is based on a compositional approach. Their algorithm only explores each subsystem once and, with that, is capable of finding good Pareto front approximations for the composed system. Compared to an exhaustive search, their results show a performance gain of 52% up to 87% and an error rate between 1.5% and 4.7%.

The work proposed in the paper at hand belongs to the second group which can be characterized as evaluation-related approximations. The DSE is accelerated by approximating the objective function calculations that are performed to obtain the quality of found solutions. The authors of References [15,16] propose approximation techniques to accelerate the calculation of performance indicators and power consumption of multiprocessor systems-on-chip, respectively. Although the corresponding results show that their approximations reach error rates of less than 18% for performance and only 9% for power consumption, both works do not integrate their approximations into a DSE. The combination of objective approximations with a DSE are proposed for example in References [17–20]. Both References [19,20] propose to combine the use of inexpensive approximations and accurate (but costly) simulations. In Reference [19], initially, exact evaluations are performed that are used to train an estimator. After the training phase is finished, the estimator is then used instead of the exact simulation to save evaluation time. Only promising solutions are still evaluated exactly. The

authors of Reference [20], on the other hand, save exploration time by statically evaluating a given percentage of all designs exactly. Thus, the quality of the remaining designs is only approximated. In References [19,20], the exploration is partly conducted with approximated evaluations. As we will show in Section 4, this can lead to incomplete and incorrect archives where optimal designs might be missing or non-optimal designs are included, respectively. The MILAN framework by Mohanty et al. [18] and the approach of Herrera [21] are, on first sight, similar to the approach at hand. The authors of Reference [18] propose a hierarchical design space optimization where, in each phase, the evaluation accuracy is increased to gradually remove potentially inferior designs. The difference to the approach presented in this paper is twofold. First, MILAN relies on the manual selection of designs after each phase, whereas our methodology works fully automated. Second, the design points explored by MILAN are not guaranteed to be optimal. Herrera [21] also uses an analytical approximation filtering valid solutions. Subsequently, an exact simulation on each of these safe solutions is performed to find the optimal ones. However, if a solution is erroneously not considered to be safe by the approximation, it will not be found by this approach. Furthermore, postponing the exact simulation (after all safe solutions have been determined) prevents pruning the design space early. RAPIDITAS [22] steers the search for good solutions iteratively where in each iteration, the processor count of the target platform is reduced. At first, the application of n tasks is mapped to n processors and simulated. In the next step, all mappings that contain $n - 1$ mappings are explored and evaluated using approximations. The best design is chosen and once again simulated. This is repeated until no solution with fewer processors can be found. As they only have to evaluate a relatively small number of solutions, they can reduce the DSE time by 72%. Compared to our approach, RAPIDITAS does not guarantee completeness of the non-dominated front as only one design per processor count is considered. If, for example, two designs with n processors would be Pareto optimal, only one of them would be represented in the final non-dominated front. Furthermore, RAPIDITAS assumes homogeneous platform models whereas our approach models heterogeneous computing resources. The work presented in Reference [23] uses individual application profiling to accelerate the design space exploration. The key point of this approach the extraction of execution traces during the DSE and shifting the actual mapping decisions to the run time of the system. The execution traces approximate the time needed for individual applications assuming each task is executed on one distinct processor. At run time, when the active applications are known, a platform manager is invoked that composes the mappings of the individual execution traces such that the resulting throughput of the application mix is minimal. This can significantly accelerate the exploration at design time. Again, this work considers homogeneous architectures. The most significant difference to our approach presented in this paper is, however, the relaying of mapping decisions into the run time of the system. Instead, our aim is to provide a set of Pareto-optimal solutions at design time from which good compromise designs can be selected for subsequent steps in the development process of the overall digital systems.

To best of our knowledge, the earliest work in this area has been conducted by Abraham et al. [17]. With the help of bounded approximations, they show that the true Pareto front can be obtained. The approximation functions must not exceed given error thresholds Δ . They define a bounded Pareto dominance relation that filters out all designs that lie beyond a $2 \cdot \Delta$ threshold. Exact evaluations have to be performed only for the remaining designs. Note that this process fails if at least one design approximation is outside the threshold Δ . For real world objective functions, this requirement is (in general) not feasible as the estimation is highly dependent on design decisions made.

In our opinion, the approach presented in this paper cannot compete directly with the aforementioned works as the scope differs too much. Although many of them aim at obtaining Pareto fronts as a result of a DSE, they do not guarantee completeness and correctness even if the entire search space is explored. Furthermore, different boundary conditions such as homogeneous hardware platforms (e.g., Reference [22]), run time decisions (e.g., Reference [23]), manual refinement steps (e.g., Reference [18]), or a different approximation scope (e.g., References [7,11,14]) make a direct comparison of exploration time and quality of the results meaningless. Even a direct comparison with

the early work of Abraham et al. [17] would be misleading in our opinion as the constraints put on the approximation function are hard to fulfill in real use cases. Hence, we constructed a reference DSE for the experimental results in Section 6 that implies the very same scope and boundary conditions as our presented approach. That way, we can be sure that performance and quality differences stem from our contribution and not from a methodological mismatch. In fact, we do not understand the present work as a direct competition. Instead, we consider it as orthogonal work that can be used in other frameworks with little overhead. For instance, our approximation approach may be included in the evaluation step of evolutionary algorithms (e.g., Reference [3]) where many evaluations have to be carried out in each iteration.

3. Fundamentals

In this section, we will give a brief overview over the key concepts that are imperative for the rest of the paper. First we will define the concept of dominance relations between design points leading to Pareto optimality. Afterwards, an introduction to answer set programming and the utilization of background theories is given.

3.1. Pareto Optimality

Finding optimal solutions to a given problem often involves multiple, contradicting objectives f_i that have to be optimized simultaneously. In such multi-objective optimization problems, a single optimal solution generally does not exist as solutions are not totally, but only partially ordered through the dominance relation $\succ$. The dominance relation $\succ$ is defined for n -dimensional quality vectors of two distinct solutions. A candidate solution x dominates another solution y ($x \succ y$) if x evaluates at least as good in every objective and better in at least one objective compared to y . Without loss of generality, for a minimization problem with n objectives, it is formally defined as follows:

$$x \succ y \leftrightarrow \forall i \in \{1, \dots, n\} : f_i(x) \leq f_i(y) \wedge \exists j \in \{1, \dots, n\} : f_j(x) < f_j(y). \quad (1)$$

A solution x is said to be Pareto optimal if no dominating solution y exists. Hence, by definition, Pareto optimal solutions in the Pareto set X_P for a given problem are mutually non-dominated to each other: $\nexists x, y \in X_P : x \succ y \vee y \succ x$.

In most complex multi-objective optimization problems, however, an exhaustive search for all Pareto optimal solutions is not feasible due to the vast search space. The result of an optimization run is often only an approximate Pareto front. It still contains solely mutually non-dominated solutions. For sake of confusion avoidance regarding the ambivalent meaning of the word approximation, we will call an approximated Pareto front non-dominated front in the following.

3.2. Answer Set Programming

In the paper at hand, we utilize answer set programming (ASP) for exploring the search space and finding feasible solutions. We will provide a short overview of ASP, its properties and illustrate why it is well suited for the DSE of embedded systems. ASP is a programming paradigm that stems from the area of knowledge representation and reasoning. It is tailored towards NP-hard search problems and is based on the stable model (i.e., answer sets) semantics. The input is a logic program formulated in a first-order language, typically separated into a general problem description (i.e., rules) and a specific problem instance (i.e., facts). A stable model is a feasible variable assignment to the input that can be inferred by applying the rules to the given facts of the instance. In contrast to other symbolic techniques such as boolean satisfiability (SAT), ASP is based on a closed-world assumption. That is, variables that have not been (yet) inferred during solving, are assumed to be false. Hence, the truth value of a variable does not have to be decided if it is not present in a specific stable model. This makes ASP especially powerful for problems where only a small subset of decision variables has to be selected, whereas the remaining ones are not relevant. For the paper at hand, a good example is the message

routing inside a network on chip (NoC). Simplified, the route is created by concatenating individual links from sender to receiver. The truth values of links that are not on that path do not have to be decided as they are automatically assumed to be not taken in ASP.

Decision variables as well as unconditionally true facts, are encoded as n -ary predicates (atoms) that consist of a predicate name and n parameters. In order to select a feasible subset of those atoms, rules are defined that describe how a new atom can be inferred from already existing knowledge. For instance, the atoms $\text{processor}(p)$, $\text{task}(t)$ and $\text{bind}(t,p)$ encode the existence of a processor with id p and a task with id t as well as the binding that t shall be executed on p , respectively. In this context, a rule defined as $\text{alloc}(P) := \text{bind}(T,P), \text{processor}(P), \text{task}(T)$ states that a processor P is allocated if any task T is bound onto it. The separation of problem instance and problem description entails another benefit of ASP in this context. The rules are generally applicable to each problem instance and, thus, do not have to be regenerated for a new instance of the problem. In order to achieve this, the input is first grounded into a variable free representation and then relayed to the solver. The actual solving process is, however, out of scope of the paper at hand, and we refer to References [1,2] for further information.

3.3. Background Theories

Pure symbolic theories such as ASP and SAT are predestined for solving linear Boolean problems. That is, a subset of decision variables is selected according to linear constraints and rules. Often however, real constraints do not have linear dependencies. One example is finding and optimizing a valid schedule for a selected binding of tasks. As resources are shared and tasks executions are dependent on each other, selecting a binding for a task might influence (i.e., increase) further task execution times or does not affect the overall timing at all (i.e., the task execution is fitted into a free time slot). In fact, the schedule is not only dependent on the binding of the task but also the order of execution and the communication routing over the network. In order to account for all those constraints, a symbolic encoding would need decision variables that represent every possible time slot for every task. With an increasing application size, the number of these decision variables would increase exponentially. Therefore, handling such constraints is not feasible for realistic problem sizes.

One possibility to handle that problem is the application of background theories. The idea is that a part is split from the original problem and solved using a specialized technique. This result is then fed back to the foreground theory (e.g., ASP, SAT) with the help of indicator variables that are known by both fore- and background theories. The methodology is originally known as satisfiability modulo theories (SMT) and was first applied to SAT. Analogously, the methodology has been recently applied to ASP (ASP modulo theories, ASPmT). Sticking to the scheduling problem, the decision process can now be relayed to a more appropriate logic. In this case, difference logic can be applied, where exact time slots for each task are irrelevant and only the order of tasks must be known. This can be easily determined by ASP and delayed to the difference logic solver.

In the work at hand, ASPmT is utilized beside scheduling also for determining Pareto optimality and the application of safe approximation as described in the next section.

4. Consistent Approximations

In this section, we present a methodology that allows us to reduce the number of expensive objective evaluations of design points in order to accelerate the DSE. Compared to previous works, the central goal of our approach is to obtain the real Pareto front when the entire search space has been explored. The final result must include every Pareto optimal solution (completeness) and must not contain dominated designs (correctness). In order to guarantee these properties, we propose the combination of safe approximations and the exact calculation of potentially good design points. Note that the exhaustive exploration of the entire search space is in general not viable for complex design problems. Hence, completeness and correctness properties are relaxed in the following in a way that they must hold w.r.t. to the explored region of the search space. In other words, if we coincidentally find a Pareto optimal design point during

DSE, our approach assures that it is not replaced by a suboptimal design point even if its approximated quality evaluates better. Otherwise, if an optimal design point is not explored by the underlying search (i.e., by the ASP solver in our case), the obtained non-dominated front is not the same as the true Pareto front. This might happen if only a specific time budget for exploration is available and the search space is too large to be covered entirely.

4.1. Safe Approximations

When calculating the quality vector of a design point, it has to be evaluated with respect to each objective function individually. Such evaluations may be costly, when, for example, time-consuming simulations have to be executed. To accelerate the evaluation, a more cost-efficient analytical approach can be applied, which results in an approximated quality vector. However, an approximation often cannot guarantee any or only very vague error bounds for the desired objective functions. This leads to erroneous results when using these values to obtain the Pareto front as shown in Figure 1.

Moreover, a precisely bounded error of the estimation, as required, for example, in the work of Abraham et al. [17], is often not practical for real world objectives such as latency where many factors (e.g., parallel execution, resource sharing) influence the calculation. Therefore, we do not require strict error bounds of the approximated values in our approach. Instead, the approximation has to guarantee a *consistent* estimation, which means that an approximation function either only exceeds the exact value or vice versa.

Definition 1. Given an evaluation function $f : X_F \mapsto \mathbb{R}$, its approximation $f' : X_F \mapsto \mathbb{R}$ is a safe approximation iff

$$\forall x_i, x_j \in X_F : \text{sgn}(f(x_i) - f'(x_i)) \cdot \text{sgn}(f(x_j) - f'(x_j)) \geq 0,$$

where $\text{sgn} : \mathbb{R} \mapsto \{-1, 0, +1\}$ is the sign function.

Definition 2. A safe approximation is called over-approximation $f^\uparrow$ iff $\forall x \in X_F : f^\uparrow(x) \geq f(x)$. Analogously, a safe approximation is called under-approximation $f^\downarrow$ iff $\forall x \in X_F : f^\downarrow(x) \leq f(x)$.

A simple analytical example of safe approximations is visualized in Figure 2. The approximation, depicted as a dashed line, is always larger than or equal to the exact quality value. This satisfies the requirement for a safe approximation in general and, specifically, for an over-approximation. In the domain of embedded systems design, a typical example for an over-approximation would be the calculation of the required execution time of a system by simply adding the WCET of each task without considering possible parallel execution on multiple cores. On the other hand, the calculation of the execution time would be an under-approximation if communication between dependent tasks is neglected.

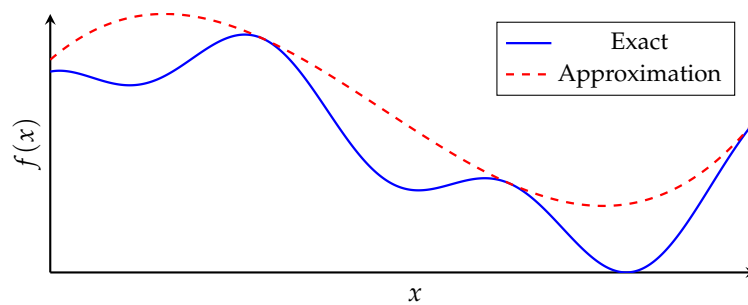


Figure 2. Safe Approximation of an analytical function.

In multi-objective optimization problems, each objective is evaluated by a separate function $f_i(x)$ and its corresponding safe approximations $f'_i(x)$. To simplify the formulation, in the following, the use

of $f(x)$ and $f'(x)$ when considering the evaluation of a design point x with n objectives implies all objective functions $f_1(x), \dots, f_n(x)$ and the corresponding approximations $f'_1(x), \dots, f'_n(x)$.

As indicated above, performing the design space exploration only with approximations does not guarantee the resulting Pareto front X_P to be complete and correct. Assume, for example, a set of valid design points $X_V = \{x_1, x_2, x_3, x_4\}$. The exploration of these points has been performed in a two-dimensional design space. As depicted in Figure 3 and in Table 1, the true Pareto front (based on the exact objective functions $f(x)$) contains x_1, x_2 , and x_4 but not x_3 . In comparison, the Pareto front based on the under-approximation $f^\downarrow(x)$ would not contain x_2 but x_3 instead. Therefore, it is neither correct nor complete.

Table 1. Pareto front of approximations as of Figure 3.

X_V	x_1	x_2	x_3	x_4
$f(x)$	(2, 5)	(3, 3)	(4, 3)	(5, 2)
$x_i \in X_P$	yes	yes	no	yes
$f^\downarrow(x)$	(1, 4)	(3, 3)	(3, 2)	(5, 1)
$x_i \in X_P^\downarrow$	yes	no	yes	yes

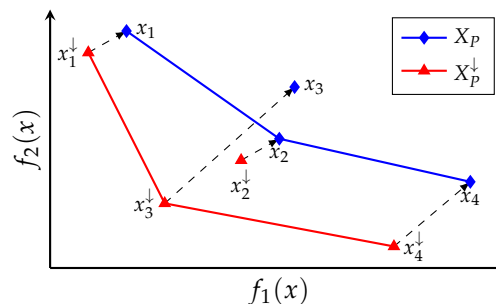


Figure 3. Exact (blue diamond) and approximated (red triangle) Pareto front.

4.2. Pareto Optimality with Approximations

Despite the fact that the use of safe approximations alone does not result in the true Pareto front, they can be used to decrease the amount of necessary exact calculations. The idea is that the approximated quality vector of a newly found solution is compared against already found solutions that are currently present in the set of non-dominated designs (the set X_P). Only if the safely approximated quality vector is not dominated by any design point in the archive, the exact value is calculated. Otherwise, it is directly removed from the search and does not have to be investigated any further.

Theorem 1. Given a set of objective functions that are to be minimized (maximized) $f(x) = (f_1(x), \dots, f_n(x))$, a design point x , its under-approximated (over-approximated) quality vector $f^\downarrow(x)$ ($f^\uparrow(x)$) and a set of exactly evaluated, mutually non-dominated design points X_P , the exactly evaluated quality vector $f(x)$ is dominated by X_P if $f^\downarrow(x)$ ($f^\uparrow(x)$) is dominated by a design in X_P .

Proof. Without loss of generality, we only consider minimization problems here. If the approximated quality vector $f^\downarrow(x)$ is dominated by X_P , at least one design point $y \in X_P$ evaluates better in each objective, that is, $f_i(y) \leq f_i^\downarrow(x)$ for $i=1, \dots, n$. According to Definition 2, the exact evaluation is always larger than the under-approximation $f_i^\downarrow(x) \leq f_i(x)$. Due to the transitivity of the inequality operator, it follows that $f_i(y) \leq f_i(x)$, that is, $y \succ x$. $\square$

A visualization of the idea is depicted in Figure 4 where the objective functions f_1 and f_2 are to be minimized. Two feasible design points A and B are found and are first evaluated approximately through $f_1^\downarrow$ and $f_2^\downarrow$. The real quality vectors of A and B are situated somewhere in the shaded red and

green areas, respectively. Thus, as the fitness vector $f^\downarrow(A)$ is already dominated by the current archive of mutually non-dominated solutions (black line), it can be discarded directly. On the other hand, the approximation $f^\downarrow(B)$ dominates the front and might be Pareto-optimal. To get a definitive result, B has to be evaluated with the exact objective functions f_1 and f_2 and compared to the archive again.

Note that under-approximations are only useful for minimization problems and over-approximations can only be utilized for maximization problems. For example, applying an over-approximation to a minimization problem, the approximated evaluation would not help in reducing the number of necessary exact evaluations. If $f^\uparrow(x)$ was dominated by X_P , $f(x)$ is not guaranteed to be dominated by X_P as the exact evaluation is smaller (i.e., better) than the approximation. On the other hand, if the $f^\uparrow(x)$ is not dominated by the front, the exact value must still be computed to save the solution and avoid the problems described in the previous subsection. Hence, the exact quality vector had to be calculated unconditionally for every found solution which led to an inevitable degradation of performance.

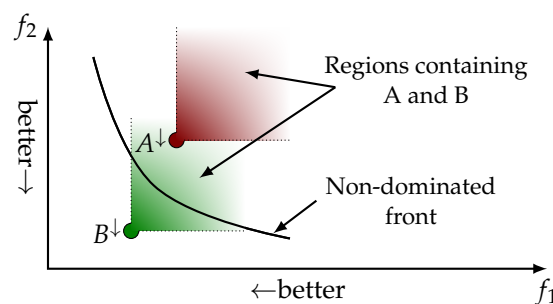


Figure 4. Under-approximations in minimization problems.

The workflow and the integration of the approach into our DSE is detailed in Figure 5. During initialization, we start with an empty archive (representing X_P) of mutually non-dominated design points. As can be seen in Figure 5, our approach is iterative, that is, potential solutions are investigated one after the other. After the feasibility filter provides a feasible solution x , its quality is approximated by its corresponding safe approximations $f^\downarrow(x)$ and checked for validity and optimality. As there are no solutions in the archive yet, the first found solution is not dominated by any other design point and therefore has to be evaluated exactly. The exact value $f(x)$ is again checked for validity and finally saved into the archive. For the next iteration, the feasibility filter delivers the next solutions with which the previous steps are again traversed. However, from now on, the newly found design points are not automatically non-dominated anymore and have to be checked for Pareto optimality twice. Whenever a check fails, the design point x is discarded without the necessity to perform the remaining checks. This process is repeated until the whole design space has been explored or an abortion criteria (e.g., timeout) is fulfilled. At this point, the archive contains either the true Pareto front (if the whole design space was explored) or at least a complete and correct non-dominated front with respect to the explored region of the design space, that is, no designs have been discarded or added wrongfully. Note that with this approach, the archive X_P , at any point in time, contains mutually non-dominated design points only.

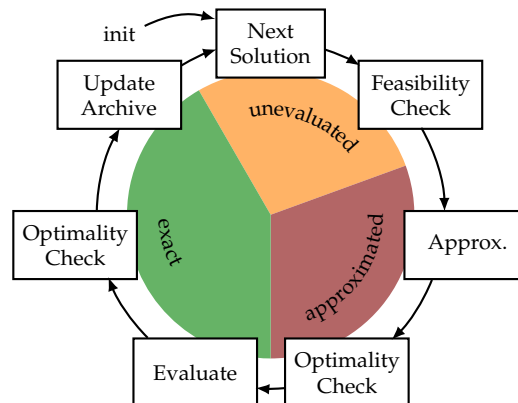


Figure 5. The design flow of the proposed iterative DSE methodology.

4.3. Accuracy Impact

In comparison to a conservative approach, our approach needs one additional validity and Pareto check whenever a promising solution has been found. Thus, if many approximated solutions are assumed to be non-dominated, many solutions will be evaluated twice which will undoubtedly deteriorate the overall performance. Otherwise, if the majority of design points is already found to be worse than the current front, the time for the costly execution can be saved in many cases. As a consequence, the performance gain of our methodology is primarily dependent on the quality, that is, accuracy and performance, of the used approximation. Intuitively, the faster and more accurate (low absolute error w.r.t. the exact evaluation) the approximation for a given objective function is, the higher is the gain.

In a previous preliminary work [24], we already investigated the theoretical boundaries of the approximating approach with respect to accuracy and performance of the approximation function. In this work, we simulated multiple optimization runs in which we adjusted the performance and accuracy of the approximation with respect to the exact evaluation. The graph depicted in Figure 6 summarizes the findings. It is shown that the approach works best when both the accuracy and the performance of the approximation are high (blue region). However, at an accuracy of 0.5 and below (and even higher for slower approximations), the overall execution times deteriorates and becomes worse compared to using a conservative approach (red region). Thus, for our approach, it is imperative to find fast and accurate approximation functions to achieve an increase in performance. Note that in order to show a general trend, we so far made the overly simplified assumption that approximation accuracy and approximation performance are fixed. This assumption generally does not hold for real use cases. Typically, the approximation accuracy and evaluation performance are highly dependent on the application structure and the decisions made during allocation, binding, and scheduling. For example, if the binding sub-step would map all tasks to the same resource (assuming no dependency conflicts), the latency evaluation would be a simple summation of execution times. On the other hand, if no resource is shared by different tasks, the evaluator had to account for additional communication delays. This would impact the performance and, at the same time, the approximation accuracy. On that account, we will present a more realistic experiment in the following sections.

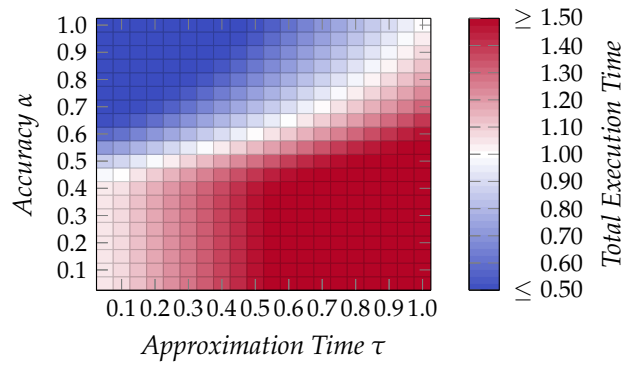


Figure 6. Impact of approximation accuracy and performance on overall run time.

5. Use Case: Symbolic DSE

In this section, we first give a general overview of our problem specification and the design space exploration at the electronic system level before we detail the integration of the approximation-based approach into our DSE.

5.1. Specification Model

As depicted in Figure 7, we model a system specification $S = (A, H, M)$ as a graph separated into applications A and a heterogeneous architecture template H that are connected by a set of mapping options M . The applications are specified as the triple $A = (T, C, E)$. T and C is a finite set of vertices modeling tasks and messages, respectively. The dependency relations are encoded by a set of directed edges $E \subseteq T \times C \cup C \times T$ with the requirement that each message $c \in C$ is sent by exactly one task $t_i \in T$ and read by another task $t_j \in T$. That means, multi-cast messages are encoded by multiple messages that are all sent by the same task.

The architecture template $H = (D, L)$, where the applications shall be executed on, is composed of vertices $D = (D_P, D_R)$ representing hardware devices, separated into processors D_P and routers D_R as well as links $L \subseteq D \times D$ that establish communication channels between devices. While we focus this work on networks-on-chip (NoCs) rather than bus based architectures, our approximation based DSE is not limited to NoCs but can be adopted for other architectures as well. Each device $d \in D$ is annotated with specific area and static power requirements that are defined by the functions $area : D \mapsto \mathbb{N}$ and $P_{stat} : D \mapsto \mathbb{N}$, respectively. Additionally, the routing delay δ_{trans} and energy E_{trans} determine the time and energy necessary to route and transmit a message over a link. Furthermore, the routers are assumed to consist of crossbar switches that allow concurrent routing on independent inputs and outputs. Note that bidirectional links in Figure 7 represent two separate links in the system on which messages can be routed concurrently. For example, the connection between r_1 and r_2 encodes the two links $l_1 = (r_1, r_2)$ and $l_2 = (r_2, r_1)$. The set of mapping options $M \subseteq T \times D_P$ connects the application and architecture graph. Therefore, at least one mapping option $m = (t, p)$ is defined for each task t that indicates that task t may be executed on processor p . In our system model, we support the encoding of heterogeneous architectures. To this end, each mapping (t, p) of the associated task t executed on processor p is annotated with the worst case execution time (WCET) $\delta_m : M \mapsto \mathbb{N}$ as well as the dynamic energy $E_{dyn} : M \mapsto \mathbb{N}$ consumed by p when executing t . By specifying more than one mapping option per task with different target processors, associated WCETs and/or energy requirements, we are able to model heterogeneous systems.

Finally, a period $\mathcal{P}$ is associated to the system that signifies the time after which all tasks repeat their execution.

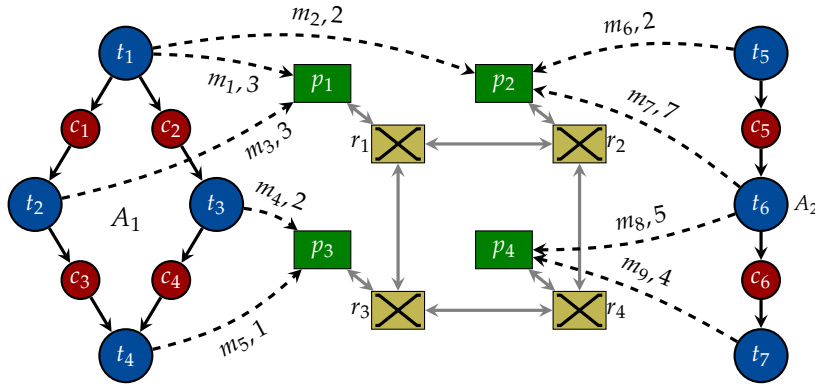


Figure 7. Our system model consists of three parts: the application graph including tasks and messages, the hardware architecture with processors and routers, and mapping options connecting the former two.

5.2. Symbolic Encoding

Given a system specification, in the first step, feasible implementations have to be obtained that consist of an allocation α , binding β , and schedule τ and adhere to their corresponding constraints. The allocation α is composed of devices and links from the heterogeneous architecture template H , that is, $\alpha \subseteq D \cup L$ that shall be used in the specific system implementation and is separated into the device and link allocation α_D and α_L . The binding $\beta \subseteq M$ selects exactly one mapping option for each task and a cycle-free route for each message, depending on the binding of the sending and receiving task. Finally, the schedule τ assigns start times to each task and message hop, that is, $\tau : T \cup C \mapsto \mathbb{N}$.

In the first step of the exploration, that is, searching for feasible solutions, we utilize the answer set programming (ASP) solver clingo [2] as it has been shown to be very efficient in exploring mapping and routing decisions.

The facts depicted in Listing 1 encode the application A_2 and a part of the architecture as shown in Figure 7. For sake of brevity, attributes are only depicted for a subset of all elements (i.e., tasks, devices, etc.) in the system. Each element is identified by a unique identifier encoded as a unary atom that is used to assign attributes to it or to connect it with other elements. Exemplary, the unary predicate processor defines a processor with its corresponding id (line 3) which can be used to specify its area (line 6) or define it as the source of a link (line 4). Facts that are valid for the entire specification (such as period(20)) do not refer to any unique identifier.

Listing 1: Specification Encoding.

```

1 task(t5). task(t6). task(t7). message(c5). message(c6).
2 send(t5,c5). read(t6,c5). send(t6,c6). read(t7,c6).
3 processor(p2). processor(p4). router(r2). router(r4).
4 link(p2,r2). link(r2,r4). link(r4,p4).
5 map(m6,t5,p2). map(m7,t6,p2). map(m8,t6,p4). map(m9,t7,p4).
6 area(p2,200). staticPower(p2,50).
7 executionTime(m6,2). dynamicEnergy(m6,322).
8 period(20). routingDelay(2). [...]
```

Allocation, binding, and routing constraints are defined as rules that are outlined in Listing 2. While the specification has to be encoded for each problem instance individually, the feasibility constraints utilize variables indicated by names beginning with capital letters that allow a uniform problem definition for each specification instance. Variables are interpreted and replaced by ground terms during the solving process. Hence, rules that contain variables represent a set of rules in the grounded, variable-free form. That is, the T in the atom task(T) is an exemplary placeholder for all available task ids and generates an individual rule for each of them. Therefore, the binding constraint in line 2 is applied to each task of a given problem instance. Informally, it states that for each task, the

solver has to select exactly one mapping option and infer a corresponding bind atom. Thus, an answer set containing both `bind(m7,t6,p2)` and `bind(m8,t6,t4)` simultaneously cannot exist.

The routing is separated into five rules (lines 4–8). The idea is that a cycle-free route is created iteratively from the destination to the source. Individual message hops are encoded by `reached` atoms. The allocation is inferred by the rules in line 10–11.

Listing 2: Exploration Encoding.

```

1 %BINDING
2 1 {bind(M,T,P) : map(M,T,P)} 1 :- task(T).
3 %ROUTING
4 root(C,R) :- send(T,C), bind(_,T,R).
5 sink(C,R) :- read(T,C), bind(_,T,R).
6 sink(C,R) :- sink(C,S), reached(C,R,S).
7             :- read(T,C), root(C,R), not sink(C,R).
8 1 { reached(C,R,S) : link(R,S) } 1 :- sink(C,S), not root(C,S).
9 %ALLOCATION
10 allocated(R) :- bind(_,T,R), task(T).
11 allocated(R) :- sink(_,R).

```

In contrast to binding, routing, and allocation, determining a complete static schedule in ASP is very inefficient as all possible task execution slots had to be enumerated and explored. This would lead to an enormous design space explosion even for small problem instances. Therefore, for the determination of a schedule, we assume a prioritized self-timed execution. That is, while the tasks are assumed to be executed and messages to be sent as soon as possible, overlapping execution slots of independent tasks are resolved using a partial order of priorities that are decided by the ASP solver. The reason for favoring a prioritized over a non-prioritized self-timed scheduling approach is mainly the prevention of scheduling anomalies. In a nutshell, a scheduling anomaly is caused when a delayed execution of a task results in better overall performance of the whole application. This may happen whenever two independent tasks are mapped to the same processor, are unequally complex (i.e., different execution times), and have successor tasks that are mapped to different processors. This leads to two implications. First, non-prioritizing could lead to situations where the optimal schedule is not found. Second, applying our under-approximation methodology for the latency by neglecting the communication overhead (see Section 5.4) would not be feasible as it may increase the overall latency. For the sake of brevity, we forgo the presentation of the exact encoding. In general, we determine the transitive shell of dependencies between tasks and assign a partial order between each two tasks that are not transitively dependent. To avoid cyclic dependencies, we utilize the acyclicity constraints that are directly implemented into clingo [25].

5.3. Evaluation

The aim of the design space exploration is the determination of a set of Pareto optimal implementations of the previously defined specification. The DSE is formulated as a multi-objective minimization problem as follows:

$$\begin{aligned}
 &\text{minimize } f(x) = (\text{lat}(x), \text{area}(x), E(x)), \\
 &\text{subject to:} \\
 &\quad x \text{ is a feasible system implementation.}
 \end{aligned}$$

To this end, each feasible solution returned by the ASP solver is evaluated w.r.t. its latency $\text{lat}(x)$, its area costs $\text{area}(x)$, and energy consumption $E(x)$. The evaluation processes are implemented as background theories utilizing the ASP modulo theories (ASPmT) interface of clingo. It allows for a direct exchange of decisions from the ASP solver and evaluation results from the objective functions via indicator variables. An overview of our use case is depicted in Figure 8. For each objective, we implement a dedicated background theory that obtains objective-specific information in form of

these indicator variables and are used to determine the corresponding objective value. Then again, the variables can also be used to return conflict clauses whenever a subset is analyzed to be responsible for invalid and/or non-optimal solutions. Especially when only a subset of decisions is identified to be responsible for constraint violations, the returned conflict clause can prune an entire region from the search space and accelerate the whole exploration process. This might happen when, for example, a selected mapping and routing leads to an execution chain of tasks that exceeds the given specified period $\mathcal{P}$. As a result, this subset of binding and routing decisions does not have to be investigated any further and can be excluded from the search. For a more detailed inspection of ASPmT in the domain of embedded systems design, we refer to the work in Reference [26].

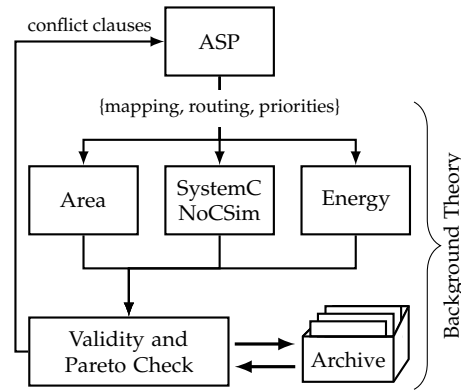


Figure 8. General overview of our DSE framework taken as a use case for our approximation approach.

In this work, the area cost yields from the accumulated area requirements of every allocated device.

$$area(x) = \sum_{d \in \alpha_D} area(d). \quad (2)$$

Devices that are not allocated in the mapping and routing are assumed to be not implemented at all and thus, do not increase the area requirements. Accordingly, the indicator variables that are exchanged between ASP and background theory only need to contain the area requirements of the devices.

The calculation of the energy consumption $E(x)$ is separated into the static E_{stat} and the dynamic part E_{dyn} .

$$E(x) = E_{stat}(x) + E_{dyn}(x), \quad (3)$$

$$E_{stat}(x) = \mathcal{P} \cdot \sum_{d \in \alpha_D} P_{stat}(d), \quad (4)$$

$$E_{dyn}(x) = \sum_{m \in \beta} E_{dyn}(m) + E_{trans} \cdot \sum_{c \in C} hops(c). \quad (5)$$

Again, the static energy arises from the sum of individual power requirements of allocated devices multiplied with the periodicity of the system. The dynamic energy stems from the selected mapping and routing options. As defined in Listing 1, the required dynamic energy is associated to each mapping option and is accumulated to the overall dynamic mapping energy. Furthermore, a message packet that is transferred over the network also adds to the dynamic energy. Thus, the entire energy consumption of each message is dependent on the length of the route it takes from the sending to the receiving device which is obtained by the function $hops : C \mapsto \mathbb{N}$. Note that we assume homogeneous links and routers here, so that the number of hops can be multiplied with the defined value E_{trans} . The number of hops per message are calculated by the ASP solver through integrated aggregate atoms that count the number of reached atoms. As these calculations are inexpensive, further approximations would not help in reducing the computation time.

While the area and energy consumption evaluations are easy to calculate in our considered model, the evaluation of the latency is more complex due to resource sharing, concurrency, and dependent task execution. To obtain an accurate result, the feasible solution is given to a closed-source NoC simulator implemented in SystemC using the transaction level modeling (TLM) standard. Each router in the NoC has at most four independent external links as well as one dedicated home link connected to a processing unit. The processing unit implements the proposed execution scheme and is interfaced via TLM target and initiator sockets. Messages are transmitted through FIFO channels using a wormhole switching strategy with source routing. They are split into equally sized flits and prepended by a configuration dependent number of header flits, which contain information for the transmission through the NoC. The NoC model is designed to have a near to flit accurate granularity but keeping a high simulation performance. As a credit per flit based reservation scheme is implemented, which reserves space in the home link of the destination router, the NoC is deadlock free (except for malicious source routes).

The prioritized self-timed execution paradigm of the tasks is realized by saving tasks according to their decided binding into priority queues of the corresponding processors and sorted through the partial order between them. Furthermore, each task is associated with its dependencies that are resolved whenever the appropriate message has been received, i.e., for each processor, only the first task can be executed if all of its dependencies have been received. After execution, the task is removed from the queue and sends its messages over the network towards the receiving task. If all dependencies have been resolved for the next task in the queue, it can be executed subsequently. Otherwise, the processor waits for new messages to arrive. Eventually, each task has been executed and the queues are empty. The latest finishing equals to the latency of the system implementation and is returned to the caller.

Obtaining the latency by simulating the system requires numerous indicator variables to be submitted by the ASP solver. Besides the binding and routing decision, also the partial order of tasks has to be communicated. Furthermore, SystemC does not allow restarting a simulation with different parameters (i.e., bindings, etc.) once a previous run has finished. That is why the simulation binary has to be executed anew for every implementation and variables must be exchanged via inter-process communication which is realized via a shared memory interface in this work. Consequently, the execution of the SystemC simulation is much more expensive than the former two objectives, area and energy.

After obtaining the quality values for all three objective functions, the validity and Pareto filters are applied to the implementation. If no constraint violations are detected and it is found to be non-dominated regarding already found solutions, the implementation is added to the archive. Simultaneously, each dominated, previously found implementation is removed from the archive to guarantee the mutual non-dominance. Finally, if the implementation is not valid or optimal, a conflict clause is generated that encodes the invalid solution and it is returned to the ASP solver to prevent a reevaluation of this solution. The whole process is repeated until the whole design space is explored or an abortion criteria has been reached.

5.4. Approximation Functions

In order to use our novel approximation-based approach, we have to find safe approximations for each objective function. An optimal implementation of a particular specification is one in which latency, area costs, and energy consumption are minimal. Hence, according to Theorem 1, we must construct under-approximations for them.

The exact area and energy evaluations of an implementation are fairly inexpensive as they are calculated analytically, so we use them directly as approximation. The corresponding latency evaluation of an implementation is much more time-consuming. In fact, the SystemC simulation takes about three orders of magnitude longer than the area and energy calculations. Consequently, we will concentrate on designing an approximation for the latency evaluation only.

The first, naive approach is the shift of finding a schedule from the simulation into the search. While it is possible in theory, the search space would explode as each possible task execution slot had

to be defined and checked for overlaps with other task slots. The number of search parameter would not only depend on the number of tasks and messages in the application and architecture but also on the timing attributes associated with them. This would not be viable because of the exponential complexity of the search.

By utilizing the employment of background theories, however, the schedulability check can be performed with specialized theories such as integer difference logic (IDL). Therefore, the ASP encoding generates only indicator variables that ensure resource sharing constraints, i.e. that message flits must not be transferred at the same time over the same link and tasks must not be executed on a processor at the same time. This can be achieved by the introduction of `diff` atoms of the form $v - w \leq -n$. Here, v and w , denote the variables that encode the start time of each task and message hop while n specifies the current execution time δ of v or the routing delay of a message hop, respectively. For instance, consider the simple implementation depicted in Figure 9a, where an application consists of three tasks t_1, t_2 , and t_3 . The task t_3 expects messages from both t_1 and t_2 and is thus dependent (As stated earlier, the dependency between two tasks may also be forced by a partial order of priorities). With $m_1 = (t_1, r_1)$ to be the selected mapping option for task t_1 , the associated `diff` atom $\tau(t_1) - \tau(t_3) \leq -\delta_m(m_1)$ would delay the execution of t_3 at least until t_1 has been fully executed.

These variables $v - w \leq -n$ are analyzed by the IDL solver in the background theory by generating a constraint graph with nodes v and w and weighted edges with value $-n$ between them. After the constraint graph is set up, the shortest path from a dedicated *zero* node to each regular node is determined which corresponds to the earliest possible start time of that node. However, if a negative cycle is detected within the graph, no shortest path can be calculated and the set of indicator variables that lead to that cycle are returned to the ASP solver as a conflict clause.

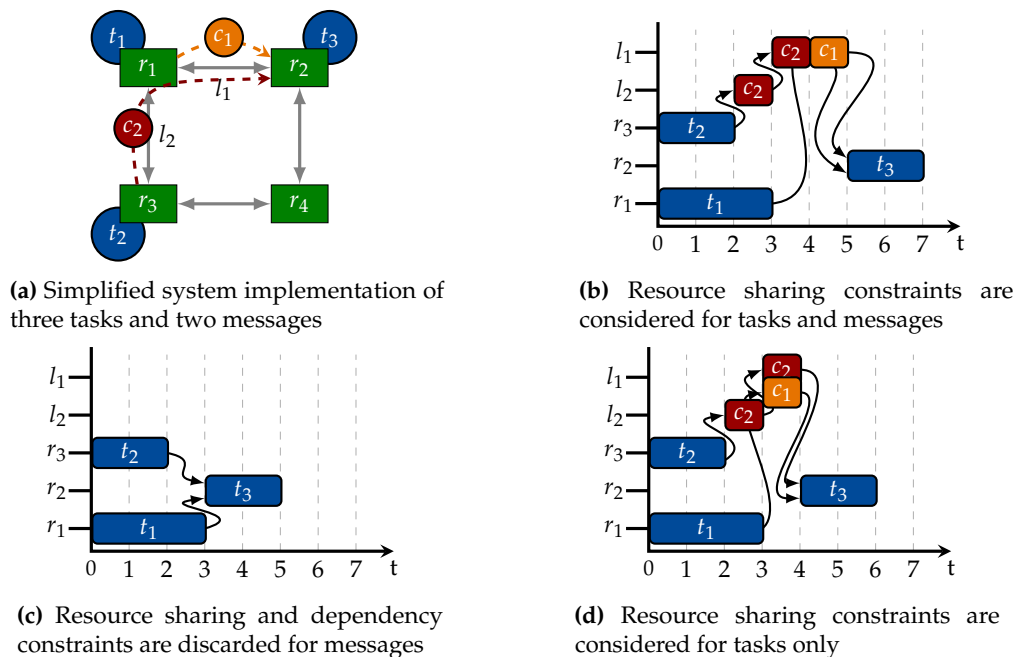


Figure 9. Different under-approximations for the calculation of the latency.

As depicted in Figure 9b, with this method, we can determine a highly accurate approximation of the latency as collisions of both task executions and message transmission are detected on flit level. However, ensuring these constraints for large systems involves a huge number of additional decision variables that have to be explored during the search. Each possible overlap of independent tasks has to be considered and prevented through constraints. Especially, considering messages on flit level granularity, each possible hop may collide with other flits that are sent simultaneously. As a result, the increased evaluation performance is eventually undone by the increased exploration complexity.

Therefore, we investigated three variations to overcome this drawback. The first valid under-approximation for the latency evaluation of an implementation is the complete neglect of resource sharing constraints. That is, tasks that are independent of each other are assumed to be executed concurrently even if they are mapped to the same device at the same time. A second approach is the omission of communication time but still respecting resource sharing constraints. This way, dependent tasks are assumed to be scheduled directly after each other even if they are bound to different resources.

While both approaches are valid under-approximations for the latency and save many decision and indicator variables, the achieved accuracy is particularly low for highly parallel and communication intense applications. An example of this approximation is depicted in Figure 9c. Even for this small application, the error of the approximated latency is already around 29% (2/7). Utilizing these approximations does not improve the overall exploration time as the approximation is assumed to be non-dominated with respect to the current front for nearly every found solution. Hence, the exact calculation is still executed each time. A third approximation function combines both approaches to a certain extent: resource sharing constraints are assured for computational tasks only (i.e., independent tasks mapped to the same resource are executed sequentially), the links of the hardware architecture, however, are assumed to have an unlimited bandwidth and, thus, are not prone to congestion. This way, messages that are routed over the same link of the network do not interfere with each other. This approach has two advantages. First, as the number of possible message hops generally surpasses task mapping options, many additional decision variables can be saved. Second, while all possible message collision have to be considered in the encoding, the number of actual collisions are typically lower than within tasks. This is why the accuracy does not deteriorate as much as with the former two approaches. The schedule in Figure 9d shows this approximation method. Even though the link l_1 is already occupied by message c_2 , c_1 is transferred simultaneously. Note that the error has been decreased to around 14% (1/7). In fact, if the ASP solver had selected another route for message c_2 , i.e., via r_4 , the latency approximation had the same value as the exact value.

6. Experimental Results

In this section, we will present our experimental setup for evaluating our approach. Instead of comparing our approach to the approaches discussed in Section 2, we developed our own reference DSE. We constructed our experiments in a way that we could assess the effect of our proposed contribution (i.e., the usage of safe approximations) only. Our reference DSE implies the very same scope and boundary conditions (i.e., models of computation, design time vs. run time decisions, manual vs. automated DSE) as our presented approach. That way, we can be sure that performance and quality differences stem from our contribution and not from a methodological mismatch. By strictly focusing on obtaining complete and correct Pareto sets by using approximations of the evaluation functions, a direct comparison to related approaches would be misleading as the premises of the other works are different. Even worse, as the boundary conditions would be varying, they could potentially mask the effect of our proposed solution.

First, we present a benchmark of small test instances. This series shows the absolute improvement of our approach over a traditional one not using safe approximations. The small instances consist of 11 tasks, 13 messages, and a total of 90 mapping options. Furthermore, it underpins the claim of correctness and completeness of our methodology. Second, we present a larger series of medium to large test instances where a full coverage of the search space is not viable anymore. Therefore, we generated a set of 120 specification instances with varying properties and complexities as follows. The application, hardware template and mapping options including corresponding properties (i.e., area, WCET, etc.) are created by an ASP-based specification generator [27]. For the application structure, we utilize series-parallel graphs. Each graph consists of a fixed number of series and parallel patterns that are connected with each other. As an example, the application A_1 in Figure 7 contains one parallel, while application A_2 contains one serial pattern. The heterogeneous target architecture is formed as a

regular 3×3 mesh implementing a NoC. For realizing the communication, on-chip routers are each connected to their neighbors and to one processor. Furthermore, each task is assumed to be composed of a specific mix of instruction types to model differently complex tasks. For example, a task t_1 may contain 60 integer, 30 floating point, and 10 special operations (e.g., AES) while a second task t_2 is only composed of 10 floating point and 30 integer operations. Each processor is characterized regarding general capability, performance (cycles per instruction), and energy efficiency (energy per instruction) for each instruction type. With this information for tasks and processors, mapping options as well as their corresponding energy and timing properties are generated. Processors that are not capable of executing a specific instruction type cannot be chosen as a mapping option for a task that requires this type.

As shown in Table 2, the 120 problem instances are organized into groups of ten system specifications that share common properties in the form of the same number of series and parallel patterns. Note that the number of series and parallel patterns of specifications with more than one application is depicted in total and also broken down by application in parentheses. Each application in one group has the same amount of parallelism as well as number of tasks and messages. In order to further investigate the influence of the task execution time to routing delay ratio (ERR), we additionally created three versions of each test instance totaling 360 individual medium to small instances plus three small instances. The medium ERR version is created as described above. In contrast, the high and low ERR versions have different ratios of the task execution times and routing delays of messages. In high ERR instances, the execution time is increased for each task by a factor of 10. Analogously, for low ERR, the execution time is decreased by a factor of 10. The goal of this approach is to investigate the accuracy impact of the approximation used. For higher ERRs, we expect a much higher accuracy and vice versa.

Table 2. Specification groups with its specific parameters

Group #	A	Series	Parallel	T	C
1	1	2	4	17	20
2	1	4	5	24	28
3	1	4	10	39	48
4	2	4 (2; 2)	9 (4; 5)	37	44
5	2	7 (3; 4)	10 (4; 6)	46	54
6	2	10 (5; 5)	11 (4; 7)	55	68
7	3	7 (2; 2; 3)	12 (4; 5; 3)	53	62
8	3	10 (3; 4; 3)	13 (4; 5; 4)	62	72
9	3	10 (5; 4; 3)	17 (4; 8; 5)	74	88
10	4	12 (3; 4; 2; 3)	14 (4; 3; 3; 4)	70	80
11	4	15 (5; 3; 3; 4)	20 (4; 5; 5; 6)	94	110
12	4	15 (6; 3; 4; 2)	27 (6; 6; 6; 9)	115	138

For the different tests, as described in the following, each optimization instance is executed three times with a timeout set to one hour (There is no timeout for the small instances). All optimization runs have been executed on an Intel Core i7 4470 with 32 GiB RAM and running Ubuntu 16.04. The code and raw data of the experiments conducted in here are available at [28].

6.1. Approximation Accuracy

In the first set of experiments, we investigate the accuracy of our approximation approach with respect to the exact evaluation using the SystemC simulation. To this end, the ratio of approximated to exact values is determined for each explored implementation. Hence, the accuracy for one found solution equals the approximated value divided by the exact objective value. We then calculate the mean accuracy and variance for each optimization run individually. The results for the medium to large instances are shown in the blue (dashed) box plot series in Figure 10. The boxes represent the 0.25 and 0.75 quantiles of the input data with the indication of the 0.5 quantile (i.e., median). The

lower whisker is the smallest data that is larger than the lower quartile minus one and a half times the interquartile range (IQR), i.e., the difference between upper and lower quartile. Analogously, the upper whisker represents the largest data that is smaller than the upper quartile plus one and a half times the IQR. For nearly all the optimization runs, the accuracy is higher than 90%. However, it is apparent that the more complex test instances reach a higher accuracy. A higher number of tasks that are bound to the same number of processors leads to a higher average utilization of individual processors. Available computation time has to be shared and tasks have to postpone their execution even though the required messages have been received. Thus, the start time of tasks is less dependent on the routing delay for messages such that an approximation of message transmissions does not influence the evaluation significantly.

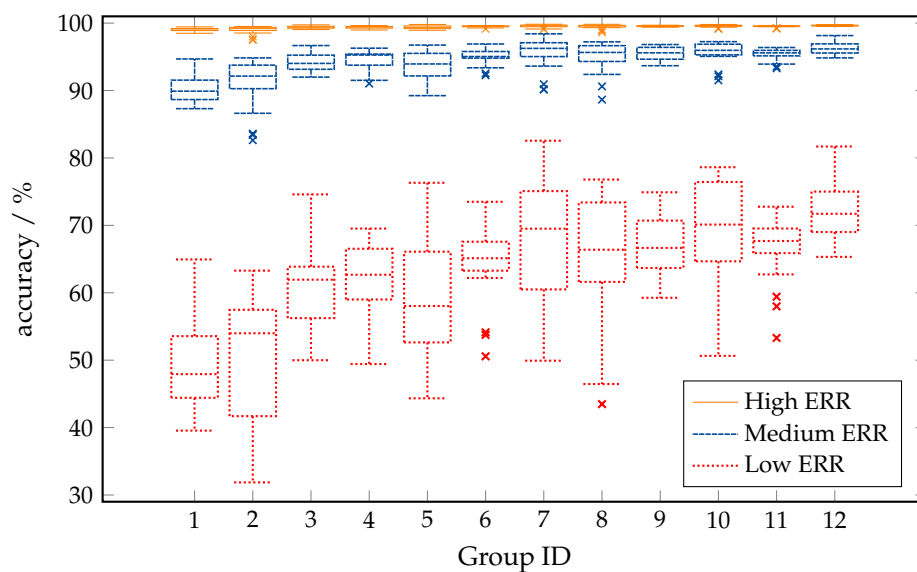


Figure 10. Latency approximation accuracy for different complexity groups and task execution time to routing delay ratios (ERRs). Blue corresponds to the original, red to a low ERR, and orange to a high ERR.

The results of the low and high ERR optimization runs are shown as the red (dotted) and orange (solid) box plot series in Figure 10, respectively. It has to be stated that the general forms are identical to the original optimization runs, i.e., less complex instances are less accurate and vice versa. However, the accuracy of our approximation approach differs significantly. While the red series (low ERR) achieves much lower accuracy, the orange series (high ERR) performs better. This behavior is expected and can be explained by the utilized latency approximation function. In high ERR problem instances, compared to the medium ERR instances, the task execution time has been increased. At the same time, the communication time has not been changed. Hence, the increase of the ERR. As a consequence, more free time slots are available on the links of the hardware platform to be used for communication messages. Thus, messages can be distributed according to more uniform patterns which leads to less congestion. Contrary, in low ERR instances, the time slots for communication are shorter and more collisions can happen on the links. As described in the previous section, congestions are not considered by the utilized approximation function, but instead are assumed to be irrelevant (cf., Figure 9d). In turn, less congestion results in higher accuracy and vice versa.

A second observation is the uneven accuracy distribution within one group of instances in the low ERR optimization runs. For instance, the accuracy for group two ranges from about 30% to nearly 65%. The reason for that behavior can be justified with the influence of the structure of a specific instance. Although the number of serial and parallel pattern is the same within each group, the execution times of tasks differ. This can lead to situations where multiple messages are sent simultaneously over the communication network. If the time slots is simultaneously low (as in the low ERR instances),

this influences the accuracy even more. Hence, the distribution follows a more uniform pattern the higher the ERR is.

Note that the accuracy results of the small test instances show the same trend as for the large ones. In detail, the accuracy of low, medium, and high ERR runs for the small instances are 0.49, 0.88, and 0.98, respectively.

6.2. Performance

While the approximation accuracy only indicates the performance of utilized approximation, we will present performance measurements of the whole approach in the following.

6.2.1. Small Instances

The small instances are explored completely. Thus, we can present absolute performance numbers for our approach compared to the traditional methodology using exact evaluations only. For evaluating the performance of our approach, we calculate the epsilon dominance [29], a binary quality indicator $\mathcal{D}_\epsilon$. For a non-dominated front A , a reference front B , and n objectives it is defined as follows:

$$\mathcal{D}_\epsilon(A, B) = \max_{b \in B} \min_{a \in A} \max_{1 \leq i \leq n} \frac{a_i}{b_i}. \quad (6)$$

In short, the epsilon dominance measures the convergence of one front to a reference front. A value of $\mathcal{D}_\epsilon(A, B) < 1$ signifies that A is dominated by the reference front (B). A value of 1 indicates that A lies directly on the reference front. As we know the true Pareto front for the small instances, a value greater than 1 is not possible.

Figure 11 depicts our experimental results and shows the quality of the obtained non-dominated fronts over time. The solid lines represent the approximation-based runs while the dashed lines represent the quality of the traditional approach. The vertical lines at the end show the overall run time of the corresponding optimization run. Essentially, there are three important observations. First, for medium and high ERR runs, our approximation-based approach converges faster than the corresponding traditional approach towards the Pareto front. The optimum is reached approximately one order of magnitude earlier (i.e., 2893 s vs. 20771 s and 2194 s vs. 23117 s) with the high ERR run having the largest gap between approximation-based and exact run. In the high ERR run, only 7183 (out of 2,103,799; 0.35%) solution candidates had to be evaluated exactly while in the medium run, due to the lower accuracy, 98,443 (out of 1,936,887; 5.08%) solutions had to be evaluated exactly. Second, in the low ERR run, our novel approach performs basically the same as the traditional approach. This becomes more obvious if we look at the high number of 929,674 (out of 931,351; 99.82%) necessary exact evaluations. Thus, although a low accuracy deteriorates the performance significantly, it is not worse than the traditional approach. Third, in the beginning of the optimization runs, the exact runs outperform the approximation-based runs. As in the beginning, nearly all solutions are better than previously found solutions, the exact evaluation has to be performed for nearly every solution candidate. Thus, in the novel approach, the approximated and exact evaluation is executed while the traditional approach only has to perform the latter step. After a few solutions have been found, the number of necessary exact evaluations decreases and the approximation-based approach starts to outperform the traditional one.

Note that due to the altered instances, the number of solutions and necessary decisions is not comparable between low, medium, and high ERR runs. The foreground solver might prune the search space earlier before the background solver is even started in some cases.

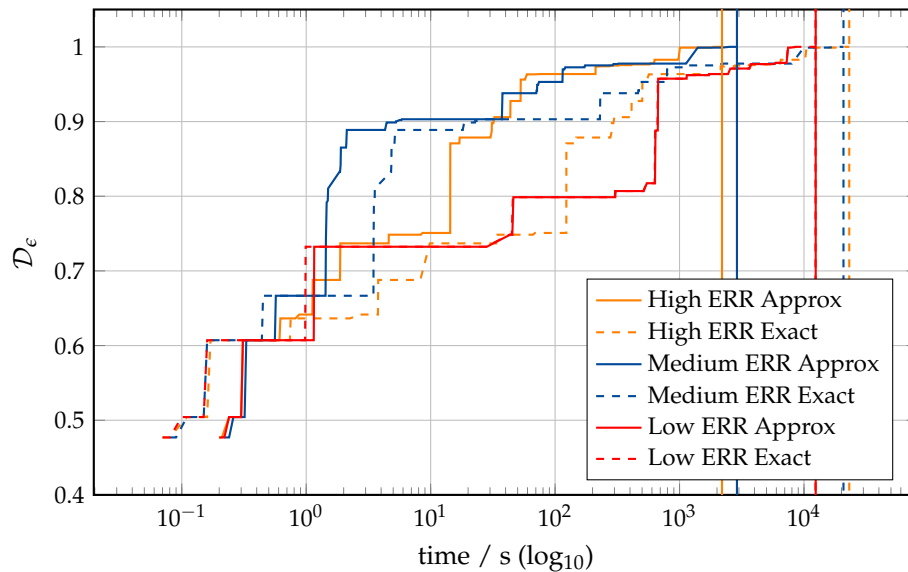


Figure 11. Epsilon dominance $\mathcal{D}_\epsilon$ over run time for the small test instances. Note that the x-axis is logarithmic.

6.2.2. Medium and Large Instances

To evaluate the performance of our approach for the medium and large instances, we performed two measurement series. The first series regards the filter ratio and its results are shown in the upper part of Table 3. The filter ratio (FR) is defined as the relative number of solutions that can be already removed safely from the search after the approximated evaluation has been executed in the background theory. Thus, it corresponds to those implementations that would have been unnecessarily evaluated. We obtained the FR for each ERR type. Note that the FR roughly correlates with the accuracy of the corresponding group. In general, the FR clearly shows that a high accuracy is imperative for our approach to be feasible. In most cases of the low ERR specifications, the FR is extremely low. For example, the average FR of 0.008 for the low ERR specification group 3 indicates that only 0.8% of all solutions can be removed from the search before the exact quality is determined. In contrast, the medium and high ERR specifications in this group achieve with 67.5% and 89.4% much better results. However, an exact correlation between accuracy and performance cannot be concluded. Exemplary, specification group 6 achieves a much higher FR for a medium ERR than for the high ERR although the accuracy is about 5% higher.

Table 3. Exploration coverage of our approach compared to the exact evaluation and filter ratio (FR) of the medium and large instances.

Group #	1	2	3	4	5	6	7	8	9	10	11	12
Filter ratio (high ERR)	0.828	0.900	0.894	0.957	0.960	0.786	0.872	0.924	0.892	0.843	0.895	0.905
Filter ratio (med ERR)	0.740	0.619	0.675	0.807	0.801	0.905	0.902	0.651	0.822	0.792	0.873	0.884
Filter ratio (low ERR)	0.049	0.093	0.008	0.044	0.009	0.016	0.170	0.055	0.077	0.063	0.138	0.274
Improvement (high ERR)	6.720	3.992	2.941	3.367	2.985	2.649	2.963	2.489	2.095	2.461	1.996	1.642
Improvement (med ERR)	3.964	2.417	2.225	3.285	2.636	2.342	2.595	2.278	2.112	2.291	1.885	1.528
Improvement (low ERR)	1.217	1.153	1.106	1.103	0.966	0.932	1.259	0.990	0.897	1.038	1.111	0.997

The second series of experiments regards the performance improvement of our approach with respect to a DSE using exact evaluations only, called reference in the following. Note that ASP solving is deterministic when supplying the solver with the same random seed. Thus, executing the same instances leads to comparable results between the reference and the approximation based approach. As typical, complex DSE problem instances impose a huge number of decision variables. Therefore, it is impossible to cover the whole design space in reasonable time. Hence, we measure the performance

gain by means of the found feasible solutions. The idea is, that if the evaluation process is faster, more time can be spent to search for new solutions in the ASP solver. The results are shown in the lower part of Table 3. Two major observations can be made from the results. First, nearly half of the low ERR specifications perform worse when compared to the reference DSE. The rest has about the same performance as the reference. The root of this behavior lies again the low accuracy and successively the low FR of the used approximation function for these instances. Therefore, for nearly all the found implementations, both the approximation and the simulation has to be performed. The additional overhead (compared to the reference) leads to a performance deterioration of up to 8% in the worst case.

The second observation is that the improvement decreases with increasing complexity of the problem instance. This can be explained by the exponential increase of decision variables that have to be explored by the underlying ASP solver. Thus, it simply takes continuously more time to find a feasible solution in the design space. That is, according to Amdahl's law, the overall improvement decreases as more time is spent in non-improvable steps.

7. Conclusions

This paper proposed a novel approach to accelerate the design space exploration of embedded systems. To this end, approximations are utilized in a way such that costly quality evaluations can be prevented for inferior system implementations. We showed that this can be achieved without sacrificing the correctness and completeness of the obtained non-dominated solutions when safe approximations are used. While our experimental results show on numerous optimization runs the ability to increase the exploration performance significantly, they also indicate that our approach does not always achieve performance gains. Especially for instances with comparatively few tasks and a good approximation accuracy, the overall performance gain is high. As expected from previous simulations, the performance gain vanishes if the approximation function cannot achieve a sufficient accuracy. As a consequence, we conclude that our novel approach is only useful if a safe approximation for complex objective functions can be defined. In fact, defining a safe approximation is the hardest challenge in our approach which is also not possible in all situations. However, our approach is even applicable if only a subset of all evaluation functions can be approximated safely. The rest of the evaluations can be performed with the exact evaluation functions. If no safe approximation can be formulated for one or more objective functions, the overall performance is not worse than a corresponding entirely exact DSE.

Finally, our approximation approach is not restricted to ASP-based DSE. It can be applied with little effort to every multi-objective optimization method that compares newly found solutions with some kind of archive of currently known non-dominated solutions. This includes many population-based meta-heuristics such as the Non-dominated Sorting Genetic Algorithm (NSGA) and Ant Colony Optimization.

Author Contributions: Conceptualization, K.N. and C.H.; methodology, K.N.; software, K.N. and B.B.; validation, K.N.; formal analysis, K.N.; investigation, K.N.; data curation, K.N. and B.B.; writing—original draft preparation, K.N. and B.B.; writing—review and editing, K.N., B.B., and C.H.; visualization, K.N.; supervision, C.H.; project administration, C.H.; funding acquisition, C.H.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the German Science Foundation (DFG) under grant HA 4463/4-1. We acknowledge financial support by the DFG and Universität Rostock/Universitätsmedizin Rostock within the funding program Open Access Publishing.

Conflicts 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. Andres, B.; Gebser, M.; Schaub, T.; Haubelt, C.; Reimann, F.; Glaß, M. Symbolic System Synthesis Using Answer Set Programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 79–91, doi:10.1007/978-3-642-40564-8_9.

2. Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; Wanko, P. Theory Solving made easy with Clingo 5. In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*; Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2016; doi:10.4230/OASIcs.ICLP.2016.2.
3. Pimentel, A.D. Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration. *IEEE Des. Test* **2017**, *34*, 77–90, doi:10.1109/MDAT.2016.2626445.
4. Thompson, M.; Pimentel, A.D. Exploiting Domain Knowledge in System-level MPSoC Design Space Exploration. *J. Syst. Archit.* **2013**, *59*, 351–360, doi:10.1016/j.sysarc.2013.05.023.
5. Lukasiewicz, M.; Glass, M.; Haubelt, C.; Teich, J. Efficient symbolic multi-objective design space exploration. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference (ASPDAC)*, Seoul, Korea, 21–24 March 2008; pp. 691–696, doi:10.1109/ASPDAC.2008.4484040.
6. Khalilzad, N.; Rosvall, K.; Sander, I. A modular design space exploration framework for multiprocessor real-time systems. In *Proceedings of the 2016 Forum on Specification and Design Languages (FDL)*, Bremen, Germany, 14–16 September 2016; pp. 1–7, doi:10.1109/FDL.2016.7880377.
7. Palermo, G.; Silvano, C.; Zaccaria, V. ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2009**, *28*, 1816–1829, doi:10.1109/TCAD.2009.2028681.
8. Silvano, C.; Fornaciari, W.; Palermo, G.; Zaccaria, V.; Castro, F.; Martinez, M.; Bocchio, S.; Zafalon, R.; Avasare, P.; Vanmeerbeeck, G.; et al. MULTICUBE: Multi-objective Design Space Exploration of Multi-core Architectures. In *Proceedings of the 2010 IEEE Computer Society Annual Symposium on VLSI*, Lixouri, Greece, 5–7 July 2010; pp. 488–493, doi:10.1109/ISVLSI.2010.67.
9. Rojas-Gonzalez, S.; Jalali, H.; Nieuwenhuyse, I.V. A stochastic-kriging-based multiobjective simulation optimization algorithm. In *Proceedings of the 2018 Winter Simulation Conference (WSC)*, Gothenburg, Sweden, 9–12 December 2018; pp. 2155–2166, doi:10.1109/WSC.2018.8632322.
10. Zhang, J.; Ma, Y.; Yang, T.; Liu, L. Estimation of the Pareto front in stochastic simulation through stochastic Kriging. *Simul. Model. Pract. Theory* **2017**, *79*, 69–86, doi:10.1016/j.simpat.2017.09.006.
11. Onnebrink, G.; Hallawa, A.; Leupers, R.; Ascheid, G.; Shaheen, A.U.D. A Heuristic for Multi Objective Software Application Mappings on Heterogeneous MPSoCs. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC)*, Tokyo, Japan, 21–24 January 2019; pp. 609–614, doi:10.1145/3287624.3287651.
12. Liu, H.; Diakonikolas, I.; Petracca, M.; Carloni, L. Supervised design space exploration by compositional approximation of Pareto sets. In *Proceedings of the 48th Design Automation Conference, DAC 2011*, San Diego, CA, USA, 5–10 June 2011; pp. 399–404, doi:10.1145/2024724.2024818.
13. Sengupta, A.; Sedaghat, R.; Zeng, Z. Rapid design space exploration by hybrid fuzzy search approach for optimal architecture determination of multi objective computing systems. *Microelectron. Reliab.* **2011**, *51*, 502–512, doi:10.1016/j.microrel.2010.08.003.
14. Schwarzer, T.; Falk, J.; Müller, S.; Letras, M.; Heidorn, C.; Wildermann, S.; Teich, J. Compilation of Dataflow Applications for Multi-Cores Using Adaptive Multi-Objective Optimization. *ACM Trans. Des. Autom. Electron. Syst.* **2019**, *24*, 29:1–29:23, doi:10.1145/3310249.
15. Aguilar, M.A.; Aggarwal, A.; Shaheen, A.; Leupers, R.; Ascheid, G.; Castrillon, J.; Fitzpatrick, L. Work-in-progress: multi-grained performance estimation for MPSoC compilers. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis For Embedded Systems*, Seoul, Korea, 15–20 October 2017; doi:10.1145/3125501.3125521.
16. Schürmans, S.; Onnebrink, G.; Leupers, R.; Ascheid, G.; Chen, X. Frequency-Aware ESL Power Estimation for ARM Cortex-A9 Using a Black Box Processor Model. *ACM Trans. Embed. Comput. Syst.* **2016**, *16*, 26:1–26:26, doi:10.1145/2987375.
17. Abraham, S.G.; Rau, B.R.; Schreiber, R. Fast design space exploration through validity and quality filtering of subsystem designs. In *HP Laboratories Technical Reports (HPL-2000-98)*, Hewlett-Packard Company: Palo Alto, CA, USA, 2000.
18. Mohanty, S.; Prasanna, V.K.; Neema, S.; Davis, J. Rapid Design Space Exploration of Heterogeneous Embedded Systems Using Symbolic Search and Multi-granular Simulation. In *Proceedings of the LCTES/SCOPES 2002*, Berlin, Germany, 19–21 June 2002; pp. 18–27, doi:10.1145/513829.513835.
19. Ascia, G.; Catania, V.; Nuovo, A.G.D.; Palesi, M.; Patti, D. Efficient design space exploration for application specific systems-on-a-chip. *J. Syst. Archit.* **2007**, *53*, 733–750, doi:10.1016/j.sysarc.2007.01.004.

20. Piscitelli, R.; Pimentel, A.D. Design space pruning through hybrid analysis in system-level design space exploration. In Proceedings of the 2012 Design, Automation & Test in Europe (DATE), Dresden, Germany, 12–16 March 2012; pp. 781–786, doi:10.1109/DATE.2012.6176600.
21. Herrera, F.; Sander, I. Combining analytical and simulation-based design space exploration for time-critical systems. In Proceedings of the 2013 Forum on specification and Design Languages, Paris, France, 24–26 September 2013.
22. Singh, A.K.; Das, A.; Kumar, A. RAPIDITAS: RAPId Design-Space-Exploration Incorporating Trace-Based Analysis and Simulation. In Proceedings of the 2013 Euromicro Conference on Digital System Design, Los Alamitos, CA, USA, 4–6 September 2013; pp. 836–843, doi:10.1109/DSD.2013.93.
23. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Resource and Throughput Aware Execution Trace Analysis for Efficient Run-Time Mapping on MPSoCs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2016**, *35*, 72–85, doi:10.1109/TCAD.2015.2446938.
24. Neubauer, K.; Haubelt, C.; Wanko, P.; Schaub, T. On Leveraging Approximations for Exact System-level Design Space Exploration: Work-in-progress. In Proceedings of 2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Turin, Italy, 30 September–5 October 2018; pp. 15:1–15:2, doi:10.1109/CODESISSS.2018.8525974.
25. Bomanson, J.; Gebser, M.; Janhunen, T.; Kaufmann, B.; Schaub, T. Answer set programming modulo acyclicity. *Fundam. Inform.* **2016**, *147*, 63–91, doi:10.3233/FI-2016-1398.
26. Neubauer, K.; others. Exact multi-objective design space exploration using ASPmT. In Proceedings of the 2018 Design, Automation & Test in Europe (DATE), Dresden, Germany, 19–23 March 2018; pp. 257–260, doi:10.23919/DATE.2018.8342014.
27. Neubauer, K.; Haubelt, C.; Wanko, P.; Schaub, T. *Systematic Test Case Instance Generation for the Assessment of System-level Design Space Exploration Approaches*; Proceedings of MBMV; Universitätsbibliothek Tübingen: Tübingen, Germany, 2018; doi:10.15496/publikation-25685.
28. Neubauer, K. MDPI Electronics Data. Available online: <https://gitlab.amd.e-technik.uni-rostock.de/open-source-es/mdpi-electronics-data> (accessed on 26 June 2020).
29. Laumanns, M.; Thiele, L.; Deb, K.; Zitzler, E. Combining Convergence and Diversity in Evolutionary Multiobjective Optimization. *Evol. Comput.* **2002**, *10*, 263–282, doi:10.1162/106365602760234108.



© 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/>).