Design-Space Exploration with Multi-Objective Resource-Aware Modulo Scheduling

Julian Oppermann[1(✉)], Patrick Sittel[2], Martin Kumm[3], Melanie Reuter-Oppermann[4], Andreas Koch[1], and Oliver Sinnen[5]

[1] Embedded Systems and Applications Group, Technische Universität Darmstadt, Darmstadt, Germany
{oppermann,koch}@esa.tu-darmstadt.de

[2] Circuits and Systems Group, Imperial College London, London, UK
psittel@ic.ac.uk

[3] Faculty of Applied Computer Science, University of Applied Sciences Fulda, Fulda, Germany
martin.kumm@cs.hs-fulda.de

[4] Discrete Optimization and Logistics Group, Karlsruhe Institute of Technology, Karlsruhe, Germany
melanie.reuter@kit.edu

[5] Parallel and Reconfigurable Computing Lab, University of Auckland, Auckland, New Zealand
o.sinnen@auckland.ac.nz

Abstract. Many of today's applications in parallel and concurrent computing are deployed using reconfigurable hardware, in particular field-programmable gate arrays (FPGAs). Due to the complexity of modern applications and the wide spectsrum of possible implementations, manual design of modern custom hardware is not feasible. Computer-aided design tools enable the automated transformation of high-level descriptions into hardware. However, the efficient identification of Pareto-optimal solutions to trade-off between resource utilisation and throughput is still an open research topic. Combining resource allocation and modulo scheduling, we propose a new approach for design-space exploration of custom hardware implementations. Using problem-specific rules, we are able to exclude obviously dominated solutions from the design space before scheduling and synthesis. Compared to a standard, multi-criteria optimisation method, we show the benefits of our approach regarding runtime at the design level.

1 Introduction

The use of reconfigurable platforms including field-programmable gate arrays (FPGAs) is common for hardware acceleration in the area of applied and high-performance embedded computing. Compared to costly and inflexible

© Springer Nature Switzerland AG 2019
R. Yahyapour (Ed.): Euro-Par 2019, LNCS 11725, pp. 170–183, 2019.
https://doi.org/10.1007/978-3-030-29400-7_13

application-specific integrated circuits, FPGAs provide relatively high throughput and low power implementations while enabling rapid-prototyping [4]. Due to the exponential rise of complexity of digital systems and FPGA capacity, the process of manually implementing specifications in hardware is inefficient regarding design time and quality [5]. To overcome this, high-level synthesis (HLS) can be applied to automatically transform behavioural descriptions into hardware. The three main steps of HLS are resource allocation, operation scheduling and binding [9]. In the allocation step, physical resources are determined. Using only the allocated resources, the scheduling step assigns execution times, usually in clock cycles, to every operation such that no data dependency is violated. Next, the operations are bound to specific functional units in hardware. Finally, the high-level input description is transformed into a hardware description language (HDL) representation that implements the operations, memory interfacing and data flow control at the register-transfer level.

The scheduling phase is crucial for the accelerator's performance, and is therefore typically the most time-consuming step in this process. Using conventional scheduling algorithms, the achievable throughput is reciprocally proportional to the determined schedule length (latency). The throughput can be increased by using *modulo scheduling*, which interleaves successive schedules [15]. Usually allocation, scheduling and binding are performed sequentially in order to reduce design time. This limits the number and quality of trade-off points in the design space. Detaching resource allocation from scheduling, state-of-the-art modulo schedulers only determine a single solution without providing any information about trade-offs or resource saving opportunities [3,12]. The research question of how to use resource allocation and scheduling efficiently to obtain Pareto-optimal trade-off points, remains open. Enumerating all possible allocations and scheduling each of them typically leads to prohibitively long runtimes. Fan et al. proposed cost-sensitive modulo scheduling [8] to synthesise the smallest (in terms of resource use) accelerator for a loop at a given, externally specified initiation interval. While their goal is similar to ours, they compute the number of functional units before scheduling using heuristic rules, whereas we can minimise the allocation as part of an exact scheduling formulation. We can thus handle the situation where the trivial resource allocation is infeasible for a given interval. The only published formulation that includes the minimisation of allocated resources in a modulo scheduling formulation is the one proposed by Šůcha and Hanzálek [18]. The above works did not address design-space exploration (DSE), however.

In this work, we make the following contributions. Firstly, we establish a formal definition and a framework for resource-aware modulo schedulers, discussing the necessary changes required to make existing, exact formulations suitable for multi-objective, resource-aware optimisation. Secondly, we discuss how to apply a standard method from multi-criteria optimisation, and propose a novel problem-specific approach compatible with our extended formulations. Our evaluation shows that the problem-specific approach outperforms the standard method in terms of both overall runtime and number of trade-off points.

2 Scheduling Framework

At the beginning of an HLS flow, an intermediate data flow graph (DFG) representation is constructed from the input loop description, modelling the *operations* that constitute the computation as vertices, and the data flow and other precedence relationships between the operations as edges. In contrast to general-purpose processors, HLS tools employ a spatial approach to computation, meaning that in the extreme case, an individual *operator* is instantiated for each operation. However, as each operator occupies a certain amount of an FPGA device's finite number of *resources*, an HLS compiler can choose to *share* operators among several operations in different time steps.

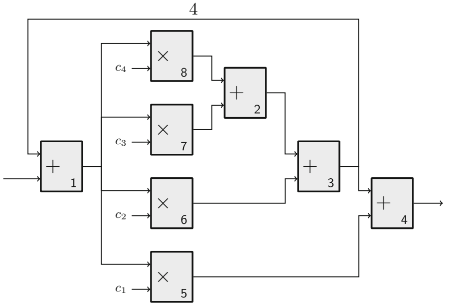

Fig. 1. Data flow graph of example.

Figure 1 shows an example DFG which we use to further illustrate the problem and introduce our notation. The DFG contains four multiplications and four additions which are represented as vertices. The result of operation 3 is delayed by four iterations (edge marked with '4') and fed back into the input of operation 1. There are different ways to schedule the execution of the operations, as illustrated in Fig. 2. Figure 2(a) shows one extreme where a separate operator is instantiated for each operation. It takes five time steps to compute one iteration, i.e. the result of operation 4, due to the data dependencies illustrated by the non-shaded parts. However, with modulo scheduling, each operator can accept new operands in each time step such that up to five iterations are processed concurrently as represented by the different colours. Since a new iteration can be initiated at operation 1 at every time step, the *initiation interval* (II) is equal to one. Figure 2(b) shows the other extreme where only one instance is used per operator type. Here, it takes eight time steps to compute one result value, but new iterations can be initiated every II = 4 time steps. Usually, several solutions exist between these extremes providing trade-offs between throughput and resource utilisation.

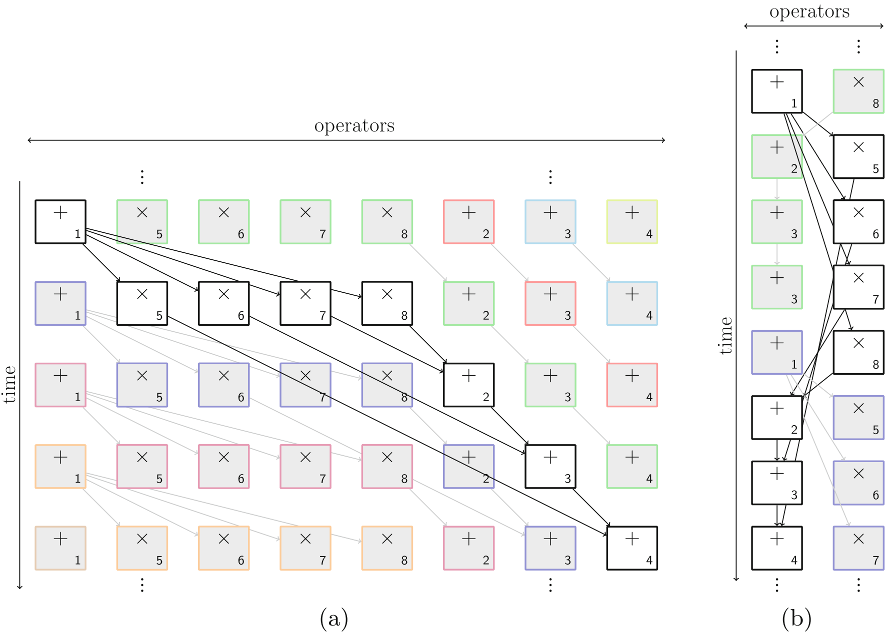

Fig. 2. Two example schedules of DFG in Fig. 1 with (a) eight parallel operators and II = 1 and (b) two parallel operators and II = 4

2.1 Formal Definitions

We now introduce the necessary terms and notations used throughout the paper, starting with the definition of the *resource-aware modulo scheduling* (**RAMS**) problem. The **target device** is abstracted to the different types of low-level resources R, and the number of elements N_{r} available of each resource $\mathsf{r} \in R$. Typical resources include lookup tables (LUTs), digital signal-processing blocks (DSPs), and memory elements such as flip-flops and on-chip block RAM. The set of **operator types** Q is derived from the HLS tool's library, which usually provides modules for the basic arithmetic/logic functionality, as well as ports to random-access memories. Each instance of operator type $\mathbf{q}$ performs a single function that takes $l_{\mathbf{q}}$ time steps to complete, and has an associated demand $n_{\mathbf{q},\mathsf{r}} \in \mathbb{N}_0$ in terms of the device's resources $\mathsf{r} \in R$. Most operator types are simple enough to implement on FPGAs to have $n_{\mathbf{q},\mathsf{r}} \ll N_{\mathsf{r}}$ regarding all resources. Therefore, it is reasonable for the HLS tool to treat them as practically unlimited, i.e. instantiate as many operators as needed. In contrast, operators whose resource demands exceed a certain threshold are candidates to be time-multiplexed by the HLS tool. Their types constitute the set of **shared operator types** $\hat{Q} \subseteq Q$. While the concrete threshold is tool-dependent, we assume that the resource demand of the multiplexing logic required for sharing is negligible in comparison to the resource demands of shared operators. Accordingly, integer addition is the canonical example for an unlimited operator

type, whereas floating-point division would be a typical shared operator type. We assume that shared operators can accept new input data, i.e. coming from a different operation, at every time step.

The sets of operations O and edges $E = \{(i \rightarrow j)\} \subseteq O \times O$ together form the **dependence graph**, which represents the semantics of the computation. In our model, each operation $i \in O$ maps to exactly one operator type. For notational convenience, we introduce the sets $O_{\mathbf{q}}$ that contain all operations using a specific operator type $\mathbf{q}$. Each dependence edge $(i \rightarrow j)$ models a precedence relationship between the operations $i, j \in O$, and is associated with two integer attributes. The delay δ_{ij} mandates additional time steps between the completion time of i and the start time of j. The distance β_{ij} expresses how many iterations later the precedence has to be satisfied. We call edges with a non-zero-distance *backedges*. The dependence graph may contain cycles that include at least one backedge. The example of Fig. 1 contains one backedge: that between operation 3 to operation 1. We denote the sum of i's operator type's latency and the edge delay as d_{ij}. In addition, we may optionally limit the maximum schedule length (latency) $U \in \mathbb{N}_0$.

A solution S to the RAMS problem consists of an **initiation interval** II^S, an allocated number of instances $a_{\mathbf{q}}^S$ for all operator types $\mathbf{q} \in Q$ that together form an **allocation** A^S, and a start time t_i^S for all operations $i \in O$, i.e. the **schedule**. Note that for all unlimited operator types $\mathbf{q'} \in Q \setminus \hat{Q}$, the allocation is fixed to $a_{\mathbf{q'}}^S = |O_{\mathbf{q'}}|$. We define the solution's utilisation of resource r as:

$$\eta_{\mathsf{r}}(A^S) = \sum_{\mathbf{q} \in Q} a_{\mathbf{q}}^S \cdot n_{\mathbf{q},\mathsf{r}} \tag{1}$$

Any **feasible** solution S must satisfy the following constraints

$$t_i^S + d_{ij} \leq t_j^S + \beta_{ij} \cdot \mathrm{II}^S \quad \forall (i \rightarrow j) \in E \tag{2}$$

$$\left|\{i \in O_{\mathbf{q}} : t_i^S \bmod \mathrm{II}^S = m\}\right| \leq a_{\mathbf{q}}^S \quad \forall \mathbf{q} \in \hat{Q} \text{ and } m \in [0, \mathrm{II}^S - 1] \tag{3}$$

$$\eta_{\mathsf{r}}(A^S) \leq N_{\mathsf{r}} \quad \forall \mathsf{r} \in R \tag{4}$$

where constraints (2) assert that all dependence edges are honoured, (3) state that no operator type shall be oversubscribed and (4) ensure that the allocation does not exceed the target device's limits.

In our setting, two **competing objectives** exist, i.e. the minimisation of the initiation interval (II), and the minimisation of the resource utilisation (RU):

$$f_{\mathrm{II}}(S) = \mathrm{II}^S \qquad\qquad f_{\mathrm{RU}}(S) = \frac{1}{|R|} \sum_{\mathsf{r} \in R} \frac{\eta_{\mathsf{r}}(A^S)}{N_{\mathsf{r}}} \tag{5}$$

As no universally applicable weighting exists, we seek to compute a set $\mathcal{S}$ of Pareto-optimal solutions with different trade-offs between the two objectives, and refer to this endeavour as the *multi-objective resource-aware modulo scheduling* (**MORAMS**) problem. A solution $S \in \mathcal{S}$ is Pareto-optimal if it is not *dominated* by any other solution, i.e. $\nexists S' \in \mathcal{S}$ with $(f_{\mathrm{II}}(S'), f_{\mathrm{RU}}(S')) < (f_{\mathrm{II}}(S), f_{\mathrm{RU}}(S))$.

2.2 Bounds

The solution space for the MORAMS problem can be confined by simple bounds derived from the problem instance.

We define the minimum allocation $A^{\perp}$ to contain $a^{\perp}_{\mathbf{q}} = 1$ instances for each shared operator type $\mathbf{q} \in Q$, and $a^{\perp}_{\mathbf{q}'} = |O_{\mathbf{q}'}|$ instances for each unlimited type $\mathbf{q}' \in Q \setminus \hat{Q}$. Note that the minimum allocation may be infeasible for any II if a MORAMS instance contains backedges, or an additional latency constraint is given. We assume that $\eta_{\mathsf{r}}(A^{\perp}) \leq N_{\mathsf{r}}$, regarding all resources r, as otherwise the problem instance is trivially infeasible.

minimise $f_{\mathrm{RU}}(X)$

subject to formulation-specific dependence constraints (→ 2)

formulation-specific constraints that ensure at most $a^{X}_{\mathbf{q}}$ operations using operator type $\mathbf{q}$ are started in each congruence class modulo II^{X} (→ 3)

$\eta_{\mathsf{r}}(A^{X}) \leq N_{\mathsf{r}} \quad \forall \mathsf{r} \in R$ (→ 4)

$a^{X}_{\mathbf{q}} \in \mathbb{N}_0$, and $a^{\perp}_{\mathbf{q}} \leq a^{X}_{\mathbf{q}} \leq a^{\top}_{\mathbf{q}} \quad \forall \mathbf{q} \in Q$

Fig. 3. Template model for resource-aware modulo scheduling

The maximum allocation $A^{\top}$ models how many operators of a particular type would fit on the device if all other operator types were fixed at their minimum allocation. Formally, we define, for each $\mathbf{q} \in \hat{Q}$:

$$a^{\top}_{\mathbf{q}} = \min \left\{ \underbrace{1}_{\text{(a)}} + \min_{\mathsf{r} \in R : n_{\mathbf{q},\mathsf{r}} > 0} \underbrace{\left\lfloor \frac{N_{\mathsf{r}} - \eta_{\mathsf{r}}(A^{\perp})}{n_{\mathbf{q},\mathsf{r}}} \right\rfloor}_{\text{(b)}}, \underbrace{|O_{\mathbf{q}}|}_{\text{(c)}} \right\} \tag{6}$$

Here, (a) represents the one $\mathbf{q}$-instance already considered in the minimum allocation, (b) models how many extra $\mathbf{q}$-instances would fit using the remaining elements of resource r, i.e. when subtracting the r-utilisation of the minimum allocation. Lastly, (c) limits the allocation to its trivial upper bound, i.e. the number of operations that use $\mathbf{q}$. For completeness, we set $a^{\top}_{\mathbf{q}'} = |O_{\mathbf{q}'}|$ for the remaining, unlimited operator types $\mathbf{q}' \in Q \setminus \hat{Q}$.

The minimum initiation interval $\mathrm{II}^{\perp}$ is usually defined (e.g. in [15]) as $\mathrm{II}^{\perp} = \max\{\mathrm{II}^{\perp}_{\mathrm{rec}}, \mathrm{II}^{\perp}_{\mathrm{res}}\}$, i.e. the maximum of the recurrence-constrained minimum II and the resource-constrained minimum II. $\mathrm{II}^{\perp}_{\mathrm{rec}}$ is induced by (2) and the recurrences (cycles) in the dependence graph, while $\mathrm{II}^{\perp}_{\mathrm{res}}$ follows from (3):

$$\mathrm{II}^{\perp}_{\mathrm{res}} = \max_{\mathbf{q} \in \hat{Q}} \left\lceil \frac{|O_{\mathbf{q}}|}{a^{\top}_{\mathbf{q}}} \right\rceil \tag{7}$$

The upper bound for the initiation interval $\mathrm{II}^{\top}$ is obtained by scheduling the instance with a non-modulo scheduler that uses heuristic resource constraints according to the minimum allocation.

3 ILP Formulations for the RAMS Problem

The template formulation in Fig. 3 illustrates how ILP-based modulo scheduling formulations can be made resource-aware with small changes. In principle, it suffices to replace formerly constant limits in the base formulation with integer decision variables modelling the allocation. For notational convenience, we consider these variables to be part of an intermediate solution X. Then, one would minimise the ILP according to the objective function $f_{\mathrm{RU}}(X)$. The specific changes required to extend state-of-the-art schedulers are described in the following.

Formulation by Eichenberger and Davidson. The formulation by Eichenberger and Davidson (abbreviated here as *ED*) limits the use of an operator (M_q, in their notation) per modulo slot only on the right-hand sides of constraints (5) [7]. Replacing M_q by the appropriate allocation variables and the objective are thus the only changes required to their model.

Formulation by Šůcha and Hanzálek. The formulation by Šůcha and Hanzálek (*SH*), is the only formulation for which a resource-aware extension was already proposed [18]. We reimplemented their unit-processing time formulation to be used in our MORAMS approach. Note though that we needed to use the weaker form of their constraints (9), i.e. before applying their Lemma 1, as otherwise the number of constraints would need to be adapted according to the dynamic values of the allocation decision variables (m_1 in their notation), which is not possible in ILPs.

Formulation by Oppermann et al. The Moovac formulation (*MV*) by Oppermann et al. was presented in two variants: Moovac-S, which is a single-II scheduler, and Moovac-I, which models the initiation interval as a decision variable [12]. The changes needed to make them resource-aware are the same for both, however. Note that the formulation, as presented in [12], does compute a *binding*, i.e. mapping of operations to concrete operators, in contrast to the ED and SH formulations, which only ensure that no more than the allocated number of operators are used in each modulo congruence class. For a fairer comparison, we adapted Šůcha and Hanzálek's idea of counting the modulo slot conflicts among the operations competing for the same shared operator type. To this end, we drop the variables r_i (in their notation) and the constraints (M3-M5), (M9) and (M11) from the formulation, and instead add the following constraints (again, in their notation):

$$\sum_{j \in L_k, i \neq j} 1 - \mu_{ij} - \mu_{ji} \ \leq a_k - 1 \ \forall i \in L_k \tag{8}$$

The binary variables μ_{ij} and μ_{ji} are both zero iff operations i and j occupy the same congruence class. The formulation can be made resource-aware by replacing the parameter a_k with the appropriate allocation variable.

4 Approaches for the MORAMS Problem

In the following, we discuss two different approaches to solve the MORAMS problem, i.e. computing a set $\mathcal{S}$ of Pareto-optimal solutions regarding $f_{\text{II}}(X)$ and $f_{\text{RU}}(X)$, with the help of the RAMS formulations described above.

4.1 ε-Approach

The ε-approach is a standard method from the multi-criteria optimisation field [6]. Its core idea, given two objectives, is to optimise for only one objective and add a constraint for the other. In order to apply the method for solving the MORAMS problem, we need to employ a RAMS formulation where *all* components of a solution are decision variables, such as the Moovac-I formulation with the extensions discussed above. The approach starts with determining an extreme point by one objective, $f_{\text{II}}(X)$ in our case, and determining the value for the other, i.e. the resource utilisation $f_{\text{RU}}(X)$. For the next iteration, a constraint forcing the resource utilisation to be less than current value *minus an* ε, is added, and the model is again solved with the II minimisation objective. We use $\varepsilon = \min_{\mathsf{r}\in R} \frac{1}{N_{\mathsf{r}}\cdot|R|}$, i.e. the smallest possible decrease in the objective value according to the device resources. This algorithm is iterated until the successively stronger ε-constraints prevent any new feasible solution to be discovered. We deviate slightly from the standard method by lexicographically minimising both the II and the resource utilisation, to ensure that we obtain the smallest possible allocation for each interval. As a bonus, we know that the II will increase in each iteration, and encode this insight in the form of a second, non-standard ε-constraint regarding $f_{\text{II}}(X)$. We only accept ILP solutions that were proven to be optimal by the solver, as suboptimal solutions could yield dominated MORAMS solutions and interfere with the convergence of the algorithm. Conversely, the returned set of solutions $\mathcal{S}$ is guaranteed to only contain Pareto-optimal solutions, thus no post-filtering is needed.

4.2 Iterative Approach

As an alternative to the ε-approach that requires the II to be a decision variable, we propose an iterative approach, in which the II is a constant for each iteration, to tackle the MORAMS problem. This approach is outlined in Algorithm 1. We choose successively larger candidate IIs from the range of possible intervals (Line 3), construct the ILP parameterised to that II, solve it with the resource utilisation objective (Line 6) and, given that the ILP solver has proven optimality, retrieve and record the solution (lines 11–13). We stop the exploration if the solver returns either no solution, or a suboptimal one, due to a violated time limit. Note that the resulting set of solutions can contain dominated solutions. While filtering out these solutions after scheduling (Line 16) is easy, significant time may be wasted in computing them. To this end, we propose two heuristic rules to skip scheduling attempts that would result in obviously dominated solutions.

Algorithm 1. Iterative approach to the RAMS problem

1: Let `ILP` be an exact modulo scheduling formulation with a candidate interval II^X (a parameter), and decision variables $a_{\mathbf{q}}^X\ \forall \mathbf{q} \in Q$ and $t_i^X\ \forall i \in O$. Consider the candidate II and the decision variables as part of an intermediate solution X.
2: $\mathcal{S} \leftarrow \emptyset$; $S^{-1} \leftarrow$ null
3: **for** $\text{II}^X \in [\text{II}^{\perp}, \text{II}^{\top}]$ **do** ▷ Iterate in ascending order
4: **if** $S^{-1} \neq$ null and $\forall \mathbf{q} \in \hat{Q} : a_{\mathbf{q}}^{S^{-1}} = \left\lceil \frac{|O_{\mathbf{q}}|}{\text{II}^X} \right\rceil$ **then**
5: **continue** with next candidate II to skip obviously dominated solutions
6: `ILP.construct(` II^X `)` ; `ILP.solveWithObjective(` $f_{\text{RU}}(X)$ `)`
7: **if** solver status is "infeasible" **then**
8: $S^{-1} \leftarrow$ null ; **continue** with next candidate II
9: **else if** solver status is not "optimal" **then**
10: **stop** exploration
11: $S \leftarrow$ new solution
12: $\text{II}^S \leftarrow \text{II}^X$; $a_{\mathbf{q}}^S \leftarrow$ `ILP.value(` $a_{\mathbf{q}}^X$ `)` $\forall \mathbf{q} \in Q$; $t_i^S \leftarrow$ `ILP.value(` t_i^X `)` $\forall i \in O$
13: $\mathcal{S} \leftarrow \mathcal{S} \cup \{S\}$; $S^{-1} \leftarrow S$
14: **if** $A^S = A^{\perp}$ **then**
15: **stop** exploration, as minimal allocation is achieved
16: **return** FilterDominatedSolutions($\mathcal{S}$)

The first rule is shown in lines 4–5. We already used the feasibility constraint in (3) to establish a static lower bound for the II. However, with knowledge of the current interval II^X, we can also use it to derive a lower bound for the allocation of each shared operator type $\mathbf{q} \in \hat{Q}$. Recall that each $\mathbf{q}$-instance can only accommodate II^X operations, which yields:

$$a_{\mathbf{q}}^X \geq \left\lceil \frac{|O_{\mathbf{q}}|}{\text{II}^X} \right\rceil \tag{9}$$

We call an allocation A^X *trivial* for an II^X if all $a_{\mathbf{q}}^X$ are equal to the right-hand side of (9).

Now, we can skip the current candidate II if the previously computed allocation $A^{S^{-1}}$ is equivalent to the trivial allocation for II^X, because it cannot be improved with respect to the previous solution. Li et al. used a similar rule to filter candidate IIs' based on the respective trivial allocations [11]. However, their definition disregards the possibility that these allocations may be infeasible, and therefore can lead to incorrectly excluded candidate IIs.

The second rule (lines 14–15) stops the exploration if the minimum allocation $A^{\perp}$ is achieved. All remaining solutions would be dominated by the current solution because the allocation cannot be improved further, and those solutions would have larger IIs. Note that both rules can only be applied if the respective minimal allocations are feasible, which may not be the case in the presence of deadlines imposed by either backedges or latency constraints.

Table 1. Complexity of problem instances

	Min	Median	Mean	Max
# operations	14	49	104	1374
# shared operations	0	4	16	416
# edges	17	81	237	4441
# backedges	0	3	23	1155

4.3 Dynamic Lower Bound for the Allocation

In order to make it easier for the ILP solver to prove that it has reached the optimal allocation for the current II, we propose to include bound (9) in the models. When using the iterative approach, we can simply add it as a linear constraint to the formulation, since II^X is a constant. For the ε-approach, (9) would be a quadratic constraint. To linearise it, we introduce binary variables II^X_π with $\mathrm{II}^X_\pi = 1 \Leftrightarrow \mathrm{II}^X = \pi$ for $\pi \in [\mathrm{II}^\perp, \mathrm{II}^\top]$, adding the following linear constraints to the formulation:

$$a^X_{\mathbf{q}} \geq \left\lceil \frac{|O_{\mathbf{q}}|}{\pi} \right\rceil \cdot \mathrm{II}^X_\pi \quad \forall \pi \in [\mathrm{II}^\perp, \mathrm{II}^\top] \tag{10}$$

5 Evaluation

We evaluated the presented MORAMS approaches on a set of **204** realistic test instances. These modulo scheduling problems were extracted from two different HLS environments: 16 instances originate from Simulink models compiled by the Origami HLS project [2], whereas 188 instances represent loops from the well-known C-based HLS benchmark suites CHStone [10] and MachSuite [16]. The latter were compiled by the Nymble C-to-hardware compiler as described in [13], using an operator library from the Bambu HLS framework [14]. Table 1 summarises the instances' complexity. Our target device was the Xilinx Zynq XC7Z020, a popular low-cost FPGA found on several evaluation boards. As resources, we model its number of lookup tables (53200), DSP slices (220), and, specifically for the C-based benchmark instances, assume the availability of up to 16 memory ports that can be used to either read from or write to an address space shared with the ARM CPU-based host system of the Zynq device.

We performed the proposed design-space exploration using Gurobi 8.1 as ILP solver on 2×12-core Intel Xeon E5-2680 v3 systems running at 2.8 GHz with 64 GiB RAM. The schedulers were allowed to use up to 8 threads, 6 hours wall-clock time and 16 GiB of memory *per instance*. We report each instance's *best* result from two runs, considering first the number of solutions, and then the accumulated runtime of the exploration.

In modulo schedulers, the II can be much lower than its latency. However, the latency should not be unbounded and there exist latency critical applications

(like in closed control loops) where a low latency is important in addition to a low II. Hence, we consider the latency as a separate user constraint. As this can significantly influence the results, we scheduled our test instances subject to three different latency constraints that cover the whole spectrum of cases: The strongest constraint is to limit the schedule length U to the length of the critical path U_{CP}. Using $\mathrm{II}^{\top}$, i.e. the length of a non-modulo schedule with heuristic resource constraints, relaxes the operations' deadlines slightly. Lastly, we adapt the loose but conservative bound U_{Im} from [12] to the maximum allocation, which by construction does not exclude any modulo schedule with minimal length.

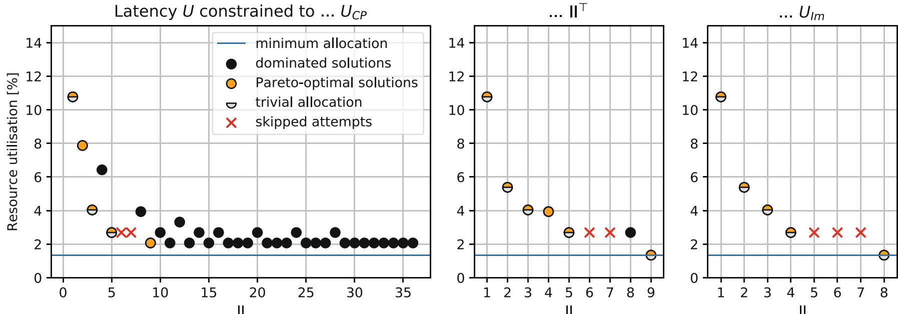

Fig. 4. Trade-off points for instance splin_pf, computed with the iterative approach

Let $\mathcal{S}_C$ be the set of solutions computed by a particular approach. We distinguish the set of Pareto-optimal solutions $\mathcal{S}$ and dominated solutions $\mathcal{S}_D$ with $\mathcal{S}_C = \mathcal{S} \cup \mathcal{S}_D$. Additionally, we define the set $\mathcal{S}_T \subseteq \mathcal{S}$ of trivial solutions, i.e. solutions with the trivial allocation for their respective II.

Figure 4 illustrates these metrics and the shape of the solution space resulting from the exploration with our iterative approach for the instance representing the Simulink model splin_pf. We picked this particular instance because it behaves differently under the three latency constraints, and showcases the effects of our heuristic rules. In the case $U = U_{\mathrm{CP}}$, many dominated solutions were computed because the minimal allocation $A^{\perp}$ was not feasible, and consequently, the early-termination rule (Lines 14–15) in Algorithm 1 was not applicable. Also, the candidate-skipping rule (Lines 4–5) was only able to skip candidate IIs 6–7. For $U = \mathrm{II}^{\top}$, the situation was significantly relaxed, as we only computed one dominated solution at $\mathrm{II} = 8$, and were able stop the exploration at $\mathrm{II} = 9$. Lastly, with $U = U_{\mathrm{Im}}$, all solutions were trivial, and no extra dominated solutions were computed. The equivalent plots for the ε-approach, which we omit here for brevity, only contain the orange-coloured Pareto-optimal solutions by construction. All approaches completed the exploration for splin_pf within three seconds of runtime.

The results of the exploration across all 204 test instances are summarised in Table 2 for the ε-approach of Sect. 4.1, as well as the iterative approach of Sect. 4.2 together with the ED, SH or MV formulations. The scheduler runtimes are accumulated in the columns "RT [h]" to give intuition into the computational effort required by the different approaches. Note that in practice, one would not need to schedule a set of instances sequentially. We then count the number of solutions in the aforementioned categories.

According to the complete exploration, the clear winner is the resource-aware ED formulation within our problem-specific, iterative approach, as it computes the most Pareto-optimal solutions (columns "$|\mathcal{S}|$") in the least amount of time (columns "RT [h]"), across all latency constraints, by a large margin. The SH formulation performs slightly better than the Moovac formulation in the MORAMS setting. We observe that for the tightest latency constraint U_{CP}, fewer trivial allocations are feasible than for the other bounds, which causes the iterative approaches to compute $|\mathcal{S}_C| \gg |\mathcal{S}|$, due to the non-applicability of the heuristic tweaks in Algorithm 1. On the other hand, the fact that $|\mathcal{S}| > |\mathcal{S}_T|$ demonstrates that only considering solutions with the trivial allocation for the respective II (e.g. as suggested in [8]) would, in general, not be sufficient to perform a complete exploration.

Table 2. Design-space exploration results for 204 instances

	$U \leq U_{\text{CP}}$				$U \leq \text{II}^{\top}$				$U \leq U_{\text{Im}}$			
Method	RT [h]	$\|\mathcal{S}_C\|$	$\|\mathcal{S}\|$	$\|\mathcal{S}_T\|$	RT [h]	$\|\mathcal{S}_C\|$	$\|\mathcal{S}\|$	$\|\mathcal{S}_T\|$	RT [h]	$\|\mathcal{S}_C\|$	$\|\mathcal{S}\|$	$\|\mathcal{S}_T\|$
ε-app	12.2	285	285	168	48.4	372	372	302	70.6	321	321	290
ED (iter)	2.4	1510	290	170	26.4	498	453	381	34.9	441	422	382
SH (iter)	16.2	1502	289	170	48.1	448	412	341	47.7	416	408	371
MV (iter)	16.0	1492	289	170	48.2	422	379	308	54.3	353	346	312

RT [h] = "total runtime in hours". $\mathcal{S}_C$, $\mathcal{S}$, $\mathcal{S}_T$ = "computed, Pareto-optimal, trivial solutions".

By design, the ε-approach computes only the Pareto-optimal solutions, regardless of the latency constraint (columns "$|\mathcal{S}_C|$" $\equiv$ "$|\mathcal{S}|$"). However, this benefit is apparently outweighed by the additional complexity introduced by modelling the II as a decision variable in the Moovac-I formulation, causing the ε-approach to be outperformed by the ED formulation.

6 Conclusion and Outlook

We presented a framework to perform a scheduler-driven design-space exploration in the context of high-level synthesis. Despite of leveraging ILP-based modulo scheduling formulations, the MORAMS problem can be tackled in a reasonable amount of time, and yields a variety of throughput vs. resource utilisation trade-off points. An open-source implementation of the proposed iterative

MORAMS approach, as well as the test instances used in the evaluation, are available as part of the HatScheT scheduling library [1].

We believe that this work can serve as the foundation for the development of heuristic approaches, as well as an environment to investigate *binding-aware* objective functions, such as register minimisation [17], or balancing the workload of the allocated operators for interconnect optimisation.

It could also be investigated, if the formulation by Eichenberger and Davidson, which already yielded the best results with our proposed, iterative approach, can be sped up further by applying a problem-reduction technique [13].

Acknowledgements. The authors would like to thank James J. Davis for providing detailed feedback regarding the clarity of this paper. The experiments for this research were conducted on the Lichtenberg high-performance computing cluster at TU Darmstadt.

References

1. HatScheT - Project Website (2019). http://www.uni-kassel.de/go/hatschet
2. Origami HLS - Project Website (2019). http://www.uni-kassel.de/go/origami
3. Canis, A., Brown, S.D., Anderson, J.H.: Modulo SDC scheduling with recurrence minimization in high-level synthesis. In: 24th International Conference on Field Programmable Logic and Applications (2014)
4. Chen, F., et al.: Enabling FPGAs in the Cloud. In: Proceedings of the 11th ACM Conference on Computing Frontiers (2014)
5. De Michell, G., Gupta, R.K.: Hardware/software co-design. Proc. IEEE **85**, 3 (1997)
6. Ehrgott, M.: Multicriteria Optimization, 2nd edn, p. 323. Springer, Berlin (2005). https://doi.org/10.1007/3-540-27659-9
7. Eichenberger, A.E., Davidson, E.S.: Efficient formulation for optimal modulo schedulers. In: Proceedings of the ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation, Las Vegas, USA (1997)
8. Fan, K., Kudlur, M., Park, H., Mahlke, S.A.: Cost sensitive modulo scheduling in a loop accelerator synthesis system. In: 38th Annual IEEE/ACM International Symposium on Microarchitecture, Barcelona, Spain (2005)
9. Gajski, D.D., Dutt, N.D., Wu, A.C., Lin, S.Y.: High-level synthesis: Introduction to Chip and System Design (2012)
10. Hara, Y., Tomiyama, H., Honda, S., Takada, H.: Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis. JIP **17**, 242–254 (2009)
11. Li, P., Zhang, P., Pouchet, L., Cong, J.: Resource-aware throughput optimization for high-level synthesis. In: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA (2015)
12. Oppermann, J., Reuter-Oppermann, M., Sommer, L., Koch, A., Sinnen, O.: Exact and practical modulo scheduling for high-level synthesis. ACM Trans. Reconfigurable Technol. Syst. **12**(2), 1–26 (2019)
13. Oppermann, J., Reuter-Oppermann, M., Sommer, L., Sinnen, O., Koch, A.: Dependence graph preprocessing for faster exact modulo scheduling in high-level synthesis. In: 28th International Conference on Field Programmable Logic and Applications, Dublin, Ireland (2018)

14. Pilato, C., Ferrandi, F.: Bambu: a modular framework for the high level synthesis of memory-intensive applications. In: 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal (2013)
15. Rau, B.R.: Iterative modulo scheduling. Int. J. Parallel Program. **24**(1), 3–64 (1996)
16. Reagen, B., Adolf, R., Shao, Y.S., Wei, G., Brooks, D.M.: MachSuite: benchmarks for accelerator design and customized architectures. In: IEEE International Symposium on Workload Characterization, Raleigh, USA (2014)
17. Sittel, P., Kumm, M., Oppermann, J., Möller, K., Zipf, P., Koch, A.: ILP-based modulo scheduling and binding for register minimization. In: 28th International Conference on Field Programmable Logic and Applications, Dublin, Ireland (2018)
18. Sucha, P., Hanzalek, Z.: A cyclic scheduling problem with an undetermined number of parallel identical processors. Comp. Opt. Appl. **48**(1), 71–90 (2011)