Modulo Scheduling with Rational Initiation Intervals in Custom Hardware Design

Patrick Sittel [*,‡], John Wickerson [*], Martin Kumm [†], and Peter Zipf [‡]
[*]Imperial College London, [†]University of Applied Sciences Fulda, [‡]University of Kassel

Abstract– In modulo scheduling, the number of clock cycles between successive inputs (the *initiation interval, II*) is traditionally an *integer*, but in this paper, we explore the benefits of allowing it to be a *rational* number. This rational II can be interpreted as the *average* number of clock cycles between successive inputs. As the minimum rational II can be less than the minimum integer II, this translates to higher throughput. We formulate rational-II modulo scheduling as an integer linear programming (ILP) problem that is able to find latency-optimal schedules for a fixed rational II. We have applied our scheduler to a standard benchmark of hardware designs, and our results demonstrate a significant speedup compared to state-of-the-art integer-II and rational-II formulations.

I. Introduction

Scheduling, the task of mapping operations to clock cycles while respecting resource constraints and maximising throughput, is an important stage in hardware synthesis. Particularly high throughput can be achieved by interleaving schedules of successive samples, as obtained using *modulo scheduling* [1], [2], [3]. In modulo scheduling, the computation's latency can exceed the *initiation interval* (II), which is the number of clock cycles between successive inputs.

In traditional modulo scheduling, the II is always an integer [2]. In this work, we explore the consequences of allowing *rational* IIs, such as $\frac{3}{2}$. The idea of a rational II is not new – it has been proposed by Fimmel and Müller in the domain of VLIW architectures [4]. Our work lifts several restrictions that limit the applicability of the Fimmel–Müller approach (Section III) and is also the first to explore rational IIs in the context of hardware design. The rough idea is to allow the number of clock cycles between successive inputs to vary, then to reinterpret the II as the *average* of these numbers. For example, in a situation where the minimum *integer* II is 2 (i.e., a new sample can be inserted every two clock cycles) there might be another solution where the II alternates between 1 and 2. This means that two samples can begin processing every three cycles, which can be interpreted as a *rational* II of $\frac{3}{2}$.

A hardware implementation using this smaller, rational II would show significant speedup, since throughput is the reciprocal of the II. Additionally, recent work has shown that only a very small and sparsely distributed number of hardware/throughput trade offs are possible using integer-II modulo scheduling [5]. Adding implementations that can only be found using rational-II modulo scheduling provides more fine-grained control over the design space.

In this paper, we present a novel integer linear programming (ILP) formulation of rational-II modulo scheduling, significantly improving on the state-of-the-art (Section IV). We show that existing approaches fail to find any rational II in most cases, whereas our algorithm can identify schedules with the minimal rational II, leading to a functional unit utilisation of close to 100%.

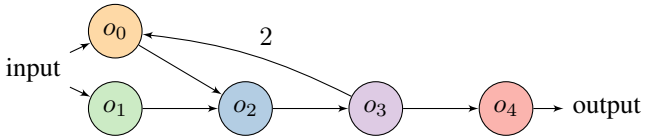

Fig. 1. Example data-flow graph.

We evaluate the performance of our approach compared to state-of-the-art modulo schedulers (Section V). Across a standard benchmark set, we found that 35% of the scheduling problems stand to benefit from a rational II, and of these, the average speedup is 1.26×. Post-routing synthesis results suggest that the additional control logic to support rational IIs is negligible.

II. Motivating Example

Consider the example data-flow graph (DFG) shown in Figure 1. It consists of five vertices of the same resource type r, which has a latency of one cycle. The edge from o_3 to o_0 is labelled with a *dependence distance* of two, to indicate a *recurrence*: operation o_0 on sample n depends on the result of operation o_3 on sample $n-2$. The other edges implicitly have a dependence distance of zero.

The maximum throughput achieved using modulo scheduling depends both on recurrences and on resource constraints. This example has one recurrence where the dependence distance is two and the latency is three cycles, so the II must not be less than $\frac{3}{2}$. This is called the *recurrence-constrained minimum II* [2], written $\mathrm{II}^{\perp}_{\mathrm{rec}}$. In general, we have

$$\mathrm{II}^{\perp}_{\mathrm{rec}} = \max_{i \in \text{ recurrences}} (\text{latency}_i / \text{distance}_i) \tag{1}$$

where latency_i and distance_i give the latency and distance of the i^{th} recurrence.

Moreover, because there are five r-operations, the II must also not be less than $5/\mathrm{FUs}(r)$, where $\mathrm{FUs}(r)$ is the number of functional units that can execute operations of type r. This is called the *resource-constrained minimum II*, written $\mathrm{II}^{\perp}_{\mathrm{res}}$. In general, we have

$$\mathrm{II}^{\perp}_{\mathrm{res}} = \max_{r \in \text{ resources}} (\#r / \mathrm{FUs}(r)) \tag{2}$$

where $\#r$ is the number of type r operations in the DFG.

The ideal performance of our proposed approach on Figure 1 compared to optimal integer-II solutions, over all possible resource allocations, is shown in Figure 2. In all cases except $\mathrm{FUs}(r) = 1$, our proposed approach leads to a significantly improved throughput, reaching 33% when $\mathrm{FUs}(r) \in \{4, 5\}$.

978-1-7281-4123-7/20/$31.00 © 2020 IEEE

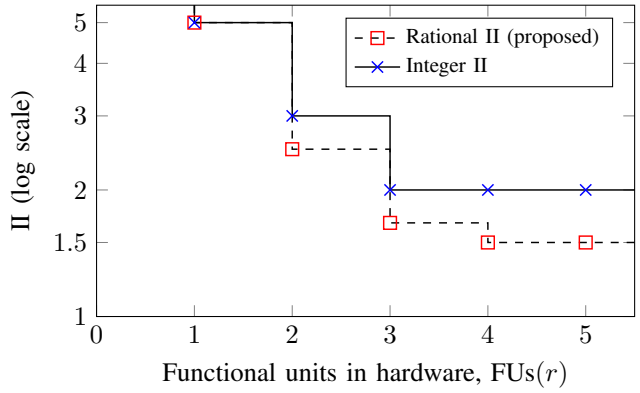

Fig. 2. Performance comparison for the DFG of Fig. 1.

To be more concrete, Table I shows one possible outcome of scheduling our example graph when $\mathrm{FUs}(r) = 3$, using both integer and rational IIs. In the integer-II case, one sample is inserted every two clock cycles. In the rational-II case, three samples are inserted every five clock cycles. This results in the integer-II schedule requiring 20 clock cycles to process 9 samples, but the rational-II schedule requiring only 18. Note that in the integer-II schedule, FU3 is idle half of the time, whereas the rational-II schedule keeps all functional units 100% utilised. It is also interesting that in the rational-II schedule, all operations on one sample can be performed by the same FU (see the thick borders in Table I), but note that this is only one of many possible resource bindings.

III. Related Work

Modulo scheduling is a multi-objective optimization problem, e.g., minimizing both II and latency, that has been formulated as an ILP problem [1], [3]. Optimal modulo scheduling can be very time-consuming, so heuristic algorithms have also been proposed, often based on systems of difference constraints [2], [6].

Fimmel and Müller have investigated the benefits of using rational IIs in modulo scheduling for VLIW architectures [4]. However, their formulation has some drawbacks that we address in this paper. First, their formulation only applies when $\mathrm{II}^{\perp}_{\mathrm{res}} < \mathrm{II}^{\perp}_{\mathrm{rec}}$, an assumption that actually rarely holds in our benchmarks, thus restricting the applicability of their approach. Second, their formulation determines a dynamic schedule, where each operation's start time is not determined until runtime, and thus incurs a significant hardware overhead.

Rational-II scheduling somewhat resembles partially unrolling the DFG then applying ordinary integer-II scheduling. However, that approach is less effective than ours because it is not always applicable when $\mathrm{II}^{\perp}_{\mathrm{rec}} < \mathrm{II}^{\perp}_{\mathrm{res}}$, it requires the user to determine the unrolling factor manually, and the ILP problem tends to be harder to solve, which means fewer solutions are found. See Section V for an experimental comparison.

IV. Rational-II Scheduling

We now describe our approach for rational-II modulo scheduling under resource constraints using ILP. All constants and variables used are listed in Table II.

TABLE I

Integer-II and rational-II schedules for the example graph when $\mathrm{FUs}(r) = 3$. The tables assign each operation (for the first 9 samples) to a clock cycle and a functional unit (FU). We write $n{:}o_i$ for operation o_i on sample n. The thick borders show the schedule for the fourth sample. The ▷ symbol indicates a clock cycle that accepts a new sample; note the non-uniform sample-insertion rate in the rational-II case.

(a) Integer II = 2

clock cycle		FU1	FU2	FU3
0	▷	0:o_0	0:o_1	
1		0:o_2		
2	▷	1:o_0	1:o_1	
3		1:o_2	0:o_3	
4	▷	2:o_0	2:o_1	0:o_4
5		2:o_2	1:o_3	
6	▷	3:o_0	3:o_1	1:o_4
7		3:o_2	2:o_3	
8	▷	4:o_0	4:o_1	2:o_4
9		4:o_2	3:o_3	
10	▷	5:o_0	5:o_1	3:o_4
11		5:o_2	4:o_3	
12	▷	6:o_0	6:o_1	4:o_4
13		6:o_2	5:o_3	
14	▷	7:o_0	7:o_1	5:o_4
15		7:o_2	6:o_3	
16	▷	8:o_0	8:o_1	6:o_4
17		8:o_2	7:o_3	
18				7:o_4
19			8:o_3	
20				8:o_4

(b) Rational II = $\frac{5}{3}$

	FU1	FU2	FU3
▷	0:o_1		
	0:o_0		
▷	0:o_2	1:o_1	
	0:o_3	1:o_0	
▷	0:o_4	1:o_2	2:o_1
▷	3:o_1	1:o_3	2:o_0
	3:o_0	1:o_4	2:o_2
▷	3:o_2	4:o_1	2:o_3
	3:o_3	4:o_0	2:o_4
▷	3:o_4	4:o_2	5:o_1
▷	6:o_1	4:o_3	5:o_0
	6:o_0	4:o_4	5:o_2
▷	6:o_2	7:o_1	5:o_3
	6:o_3	7:o_0	5:o_4
▷	6:o_4	7:o_2	8:o_1
		7:o_3	8:o_0
		7:o_4	8:o_2
			8:o_3
			8:o_4

A. Integer and Rational Minimum II

In state-of-the-art modulo scheduling, the recurrence-constrained minimum II and the resource-constrained minimum II, as defined in (1) and (2), provide a lower bound for the II. The *integer* minimum II is defined as

$$\mathrm{II}^{\perp}_{\mathbb{N}} = \max(\lceil \mathrm{II}^{\perp}_{\mathrm{res}} \rceil, \lceil \mathrm{II}^{\perp}_{\mathrm{rec}} \rceil) \ . \tag{3}$$

The *rational* minimum II, on the other hand, avoids the ceiling functions and can be determined as

$$\mathrm{II}^{\perp}_{\mathbb{Q}} = \max(\mathrm{II}^{\perp}_{\mathrm{res}}, \mathrm{II}^{\perp}_{\mathrm{rec}}) \ . \tag{4}$$

It follows that $\mathrm{II}^{\perp}_{\mathbb{N}} = \lceil \mathrm{II}^{\perp}_{\mathbb{Q}} \rceil$, and hence that rational-II schedules will always attain a throughput that is at least as good as integer-II schedules.

When $\mathrm{II}^{\perp}_{\mathbb{Q}}$ is already an integer, we have $\mathrm{II}^{\perp}_{\mathbb{Q}} = \mathrm{II}^{\perp}_{\mathbb{N}}$, so switching to rational-II scheduling cannot improve throughput (speedup = 1). This situation can be identified quickly before scheduling, and standard integer-II algorithms can be applied. On the other hand, the *maximum* speedup is obtained when $\mathrm{II}^{\perp}_{\mathbb{Q}} = 1 + \epsilon$ for small, positive ϵ. In this case, the speedup is $\frac{\lceil 1+\epsilon \rceil}{1+\epsilon}$, which tends towards 2. In summary, we have

$$1 \leq \text{speedup} < 2 \ . \tag{5}$$

In our experiments (Section V), we observe that potential speedups are widely spread from 1 up to 1.98.

TABLE II
CONSTANTS (TOP) AND VARIABLES (BOTTOM) FOR RESOURCE-CONSTRAINED RATIONAL-II MODULO SCHEDULING

Constant / Variable	Meaning
O	Set of operations in the DFG
E	Set of edges in the DFG
$d_{ij} \in \mathbb{N}_{\geq 0}$	Dependence distance on edge $o_i \to o_j$
R	Set of resource-constrained operation types (e.g., add, mult)
$\check{O} \subseteq O$	Set of resource-constrained operations
$\check{O}_r \subseteq \check{O}$	Set of resource-constrained operations of type $r \in R$, i.e., $\bigcup_{r \in R} \check{O}_r = \check{O}$
$\text{FUs}(r) \in \mathbb{N}_{\geq 1}$	No. of allowed hardware instances of resource type $r \in R$
$D_i \in \mathbb{N}_{\geq 0}$	Latency of operation $o_i \in O$
$M \in \mathbb{N}_{\geq 1}$	No. of cycles before the rational II modulo schedule repeats
$S \in \mathbb{N}_{\geq 1}$	No. of samples inserted every M cycles
$0 \leq s \leq S-1$	Range of sample indices
$\text{II}_{\mathbb{Q}} = \frac{M}{S}$	Rational initiation interval
$L \in \mathbb{N}_{\geq 0}$	Maximal latency constraint
$t_{i,s} \in \mathbb{N}_{\geq 0}$	Start time of operation o_i on sample s
t_v	Virtual node
$b_{i,s,\tau}$	True iff τ is the start time of operation $o_i \in \check{O}$ on sample s
$\langle \text{II}_0 \ ... \ \text{II}_{S-1} \rangle$	Latency sequence
$I_s \in \mathbb{N}_{\geq 0}$	Insertion time of sample s

B. Prerequisites

We consider the input to be a DFG (O, E) where operations $o_i \in O$ are connected by directed edges $(o_i, o_j) \in E$ that have a latency D_i. We write $\check{O}_r$ for the set of operations that require resource type r (adder, multiplier, etc.). The number of available functional units of type r is $\text{FUs}(r)$. As in state-of-the-art integer-II modulo scheduling formulations, we consider the II to be a constant input to the ILP problem, as calculated using (4). We write II in the form $\text{II} = \frac{M}{S}$, where M is the number of cycles before the insertion sequence repeats, and S is the number of samples inserted every M cycles.

Each operation o_i gets assigned S different clock cycles, $t_{i,0}, \ldots, t_{i,S-1}$, where $t_{i,s}$ holds the clock cycle in which operation o_i is operating on sample s. Similar to Eichenberger et al. [1], the binary variable $b_{i,s,\tau}$ models whether operation o_i of sample s is scheduled in clock cycle τ. We assume that the maximum allowed latency L is provided by the user. It follows that it is sufficient to build a schedule based only on the first P clock cycles, where $P = M+L$. This is because we can assume, without loss of generality, that the sample with index 0 will arrive at clock cycle 0, hence that the sample with index $S-1$ will arrive before clock cycle M, and hence that the sample with index $S-1$ will complete before clock cycle $M+L$.

C. Sequential Sample Insertion

We assign every sample s an insertion time I_s modulo M. For our motivating example in Table I where $\text{II} = \frac{5}{3}$, we would have $I_0 = 0$, $I_1 = 2$, and $I_2 = 4$. This means that for all $n \geq 0$, we have sample $3n$ inserted at cycle $5n$, sample $3n+1$ inserted at cycle $5n+2$, and sample $3n+2$ inserted at cycle $5n+4$. We fix the first insertion time to 0.

The repeating sequence of insertions lets us calculate the latency in clock cycles between successive samples. For this, we adopt the concept of latency sequences [7], which take the form

$$\langle \text{II}_0 \ \text{II}_1 \ ... \ \text{II}_{S-1} \rangle \tag{6}$$

where

$$\text{II}_s = \begin{cases} I_{s+1} - I_s & \text{if } s < S-1 \\ M - I_s & \text{if } s = S-1 \end{cases} . \tag{7}$$

The sample insertion times from the motivating example lead to a latency sequence of $\langle 2 \ 2 \ 1 \rangle$. This yields a modulo-5 schedule where new samples will be inserted in cycles $\{0, 2, 4, 5, 7, ...\}$. Note that integer IIs correspond to latency sequences of length 1, such as $\langle 3 \rangle$.

D. Causality

The introduction of latency sequences requires us to revisit the causality constraint used in integer-II scheduling. A typical causality constraint [1], [3] is

$$t_i + D_i - d_{i,j} \cdot \text{II} \leq t_j \qquad \forall (o_i \to o_j) \in E \tag{8}$$

which expresses that the start time of operation o_j (which is given by t_j) must not precede the end time of operation o_i from $d_{i,j}$ samples ago (which is given by $t_i + D_i - d_{i,j} \cdot \text{II}$). Here, the dependence distance (algorithmic delay) $d_{i,j}$ is multiplied by II because this is the number of cycles between successive samples.

As an example, consider the integer-II schedule from Table I(a), and the edge from o_3 to o_0 in Figure 1. We have $t_3 = 3$, $t_0 = 0$, $D_3 = 1$, $d_{3,0} = 2$, and $\text{II} = 2$, so (8) holds in this instance.

However, the introduction of latency sequences means that the number of cycles between successive samples can vary, depending on the sample index, s. Assuming a latency sequence $\langle \text{II}_0 \ \text{II}_1 \ ... \ \text{II}_{S-1} \rangle$, the number of cycles between sample s and sample $s-d$ can be calculated as

$$\Delta_s(d) = \sum_{n=1}^{d} \text{II}_{(s-n) \bmod S} . \tag{9}$$

Starting at sample s, the calculation steps backwards through the latency sequence, adding up the last d latencies. Thus, the causality constraint becomes

$$t_{i,s} + D_i - \Delta_s(d_{i,j}) \leq t_{j,s} \quad \forall (o_i, o_j) \in E, \forall s < S . \tag{10}$$

As an example, consider the rational-II schedule from Table I(b), and the edge from o_3 to o_0. When $s = 0$, we have $t_{3,s} = 3$, $t_{0,s} = 1$, $D_3 = 1$, $d_{3,0} = 2$, and $\Delta_s(2) = 3$, so (10) holds in this instance. It also holds for $s = 1$ and $s = 2$. However, with an *alternative* latency sequence $\langle 1 \ 1 \ 3 \rangle$, obtained by shifting the FU2 column in Table I(b) up by one cycle and the FU3 column up by two cycles, we would get $\Delta_s(2) = 2$, and hence (10) would be violated – the third sample is being inserted too soon.

E. ILP Formulation

Since the II is an input to the ILP, the objective of the ILP is to minimize the latency of each sample. Following Cong and Zhang [8], we add a 'virtual' node t_v. By adding constraints to the ILP, we ensure that t_v is scheduled in a cycle after all nodes are finished ($t_{i,s} + D_i$) processing. Minimising the start time of the virtual node is then the same as minimising latency for all samples.

The overall problem of rational-II modulo scheduling is now formulated as follows:

$$\min(t_v)$$

subject to

$$\textbf{D1}: \quad t_{i,s} + D_i - \Delta_s(d_{i,j}) \leq t_{j,s} \qquad \forall (o_i, o_j) \in E, \forall s < S$$

$$\textbf{D2}: \quad t_{i,0} + D_i \leq t_v \qquad \forall o_i \in O$$

$$\textbf{D3}: \quad t_v \leq L$$

$$\textbf{M1}: \quad I_0 = 0$$

$$\textbf{M2}: \quad I_s \leq I_{s+1} \qquad \forall s < S-1$$

$$\textbf{M3}: \quad t_{i,s+1} - I_{s+1} = t_{i,s} - I_s \qquad \forall s < S-1$$

$$\textbf{R1}: \quad \sum_{0 \leq \tau < P} \tau \cdot b_{i,s,\tau} = t_{i,s} \qquad \forall r : \forall o_i \in \check{O}_r, \forall s < S$$

$$\textbf{R2}: \quad \sum_{0 \leq \tau < P} b_{i,s,\tau} = 1 \qquad \forall r : \forall o_i \in \check{O}_r, \forall s < S$$

$$\textbf{R3}: \quad \sum_{0 \leq s < S} \sum_{\substack{0 \leq \tau < P \\ \tau \bmod M = m}} b_{i,s,\tau} \leq \mathrm{FUs}(r) \qquad \forall r : \forall o_i \in \check{O}_r, \forall m < M$$

D1 enforces the causality constraint introduced in (9). The latency of each sample is constrained by the user-specified value L, which can be seen in **D2** and **D3**.

The modulo constraints **M1**–**M3** enable sequential IIs. **M1** ensures that the schedule starts in the first clock cycle. **M2** prevents any samples being inserted after their successor. **M3** expresses that each of the S samples follows the same schedule (and hence that every sample is fully processed within L cycles of its insertion), except that each schedule is offset by that sample's insertion time. This constraint is one that would not be enforced if we used partial unrolling and integer-II scheduling (see discussion in Section III). It is a useful constraint because it reduces the search space yet remains satisfiable by all the solvable scheduling problems we have encountered. In future work, we plan to investigate relaxing this constraint, so different samples can follow different schedules. This may enable better schedules to be found, at the cost of further complicating the ILP.

To enforce that the number of FUs used does not exceed $\mathrm{FUs}(r)$ we use binary variables. The Boolean value $b_{i,s,\tau}$ in **R1** is true if and only if $t_{i,s} = \tau$. This is ensured by **R2**, which allows one (and only one) $b_{i,s,\tau}$ to be true for each $t_{i,s}$. This information is then used in **R3** to make sure that the upper limit $\mathrm{FUs}(r)$ for each resource type r is respected. The inner sum in **R3** adds all occupations of resource r in clock cycle $m \bmod M$. This is done for all samples, thus preventing the scheduling of more than $\mathrm{FUs}(r)$ operations of resource r in any clock cycle.

V. Experiments

We evaluated the proposed rational-II modulo scheduling approach on a set of fourteen test instances from digital signal processing and embedded computing, as listed in the first column of Table III. The vanDongen benchmark was used by Fimmel and Müller [4]; we include it because it is the only example we could find where their assumption of $\mathrm{II}^{\perp}_{\mathrm{rec}} > \mathrm{II}^{\perp}_{\mathrm{res}}$ can actually be met. Ten of the remaining benchmarks are the same test instances used by Sittel et al. [18]; the remaining three (gen, srg [10] and cholesky [17]) are new. The source code of all our benchmarks is available at [19].

Our proposed formulation was implemented in the open-source `HatScheT` library [20]. *ScaLP* was used to generate the ILP and Gurobi 8.1 (single thread mode) was used as solver [21]. All problems were solved on a server system with an Intel Xeon E5-2650v3 2.3 GHz CPU with 128 GB RAM. The hardware description after scheduling was generated using [19] which uses *FloPoCo* [22] for VHDL generation. The examined hardware implementations were synthesized, placed and routed for a Xilinx Virtex7 xc7v2000t g1925-2G targeting 250 MHz using Vivado v2018.1.

First, we analyse the potential speedup for rational-II scheduling by evaluating $\mathrm{II}^{\perp}_{\mathrm{rec}}$ and $\mathrm{II}^{\perp}_{\mathrm{res}}$ for all possible resource allocations (#FUs) for each problem. Every operation of the same type is implemented in hardware using homogeneous FUs. The results of this experiment are displayed in Table III. To provide a complexity overview, the number of operations (#ops) and the $\mathrm{II}^{\perp}_{\mathrm{rec}}$ of the DFGs are given. For each benchmark, we enumerate all possible resource allocations (#allocs). The 'avg. $\mathrm{II}^{\perp}_{\mathrm{res}}$' column reports the average value of $\mathrm{II}^{\perp}_{\mathrm{res}}$ over all of these allocations. We then report how many of the possible resource allocations lead to $\mathrm{II}^{\perp}_{\mathrm{res}} > \mathrm{II}^{\perp}_{\mathrm{rec}}$. For example, in test instances biquad and lms, we find that $\mathrm{II}^{\perp}_{\mathrm{rec}}$ always dominates $\mathrm{II}^{\perp}_{\mathrm{res}}$, and since $\mathrm{II}^{\perp}_{\mathrm{rec}}$ is an integer in both cases, no speedup can be obtained using rational-II scheduling.

We then report how many of the remaining resource allocations have a minimum II that is not an integer (column 'rational II'). For example, test instance dlms has $\mathrm{II}^{\perp}_{\mathrm{res}} > \mathrm{II}^{\perp}_{\mathrm{rec}}$ in three out of its 15 possible resource allocations, but still the minimum II in each case is an integer. This can be explained by the fact that the resource type with the largest number of operations is `mult`, with five instances. No allocation can lead to a rational II between 4 and 5 and, thus, no speedup can be obtained using rational-II scheduling. Note that this can always be determined quickly before attempting scheduling and an integer-II scheduler can be used instead. In all other cases, there exist resource allocations where the minimum II is not an integer. On average, 35% of all resource allocations show speedup potential for rational-II scheduling (see bottom row of Table III). Of those, the average potential speedup is 1.26×. In the larger models (sam, cholesky), the maximum speedup possible was 1.98×. In larger benchmarks, potential speedups are tightly distributed within the possible range we derived in Section IV A

We solved the scheduling problems using three approaches besides our own: (1) Fimmel and Müller's rational-II formulation [4], (2) the Moovac integer-II formulation [3], and (3) Moovac after partially unrolling the problem having identified

TABLE III
RATIONAL-II MODULO SCHEDULING: SPEEDUP THAT CAN BE POTENTIALLY OBTAINED (IF SCHEDULES CAN BE FOUND)

	DFG properties		Allocation info (sweep over all possible resource allocations)				Potential speedup	
instance	#ops	$\mathrm{II}^{\perp}_{\mathrm{rec}}$	#allocs	avg. $\mathrm{II}^{\perp}_{\mathrm{res}}$	#($\mathrm{II}^{\perp}_{\mathrm{res}} > \mathrm{II}^{\perp}_{\mathrm{rec}}$)	rational II	avg.	max.
vanDongen [9]	10	5.33	10	2.93	1	9 (90%)	1.13×	1.13×
dlms [10]	16	4	15	2.71	3	0 (0%)	–	–
gen [10]	15	1	15	2.71	14	7 (47%)	1.3×	1.6×
gm [11]	16	1	24	3.04	23	5 (21%)	1.47×	1.67×
hilbert [12]	14	1	18	2.42	17	3 (17%)	1.33×	1.33×
lms [10]	15	18	15	2.71	0	0 (0%)	–	–
linear phase [13]	29	1	91	4.11	90	71 (78%)	1.25×	1.87×
srg [10]	17	1	8	2.29	7	1 (13%)	1.5×	1.5×
sam [14]	121	1	1770	6.77	1769	1403 (79%)	1.21×	1.97×
biquad [15]	14	10	16	2.69	0	0 (0%)	–	–
rgb [16]	24	1	64	3.07	63	7 (11%)	1.5×	1.5×
spline [16]	26	1	64	3.78	63	26 (41%)	1.3×	1.75×
ycbcr [16]	22	1	32	2.78	31	3 (9%)	1.5×	1.5×
cholesky [17]	266	1	113386	9.31	113385	100235 (88%)	1.15×	1.98×
average	43.2	3.4	–	3.66	–	– (**35**%)	**1.26**×	**1.49**×

TABLE IV
RATIONAL II SCHEDULER COMPARISON LIMITING SOLVING TIME TO 60 SECONDS FOR EACH PROBLEM

		F.–M. [4]			Moovac [3]			unroll+Moovac			prop. ILP			speedup w.r.t.	
instance	#allocs	avg. II	solved	opt.	avg. II	solved	opt.	avg. II	solved	opt.	avg. II	solved	opt.	F.–M.	Moovac
vanDongen	9	5.4	9	9	6	9	9	5.3	9	9	5.3	9	9	1.02×	1.13×
gen	7	2.3	7	7	2.3	7	7	1.8	7	7	1.8	7	7	1.3×	1.3×
gm	5	2	5	5	2	5	5	1.4	5	5	1.4	5	5	1.5×	1.5×
hilbert	3	2	3	3	2	3	3	1.5	3	3	1.5	3	3	1.3×	1.3×
linear phase	71	3.7	71	10	3.3	71	68	3.2**	24	22	3.0**	43	3	1.2×	1.1×
srg	1	2	1	1	2	1	1	1.3	1	1	1.3	1	1	1.5×	1.5×
sam	100*	–	0	0	2.9	20	19	–	0	0	2.9**	1	1	–	1.0×
rgb	7	2	7	7	2	7	7	1.3	7	5	1.3	7	7	1.5×	1.5×
spline	26	2.4	26	7	2.4	26	26	2.2**	15	8	2.1**	20	11	1.2×	1.2×
ycbcr	3	2	3	3	2	3	3	1.3	3	3	1.3	3	3	1.5×	1.5×
cholesky	100*	–	0	0	–	0	0	–	0	0	–	0	0	–	–
total	332	–	132	52	–	152	148	–	74	65	–	98	49	–	–

* For the larger benchmarks, a random subset of all possible resource allocations was chosen.
** To provide a fair average speedup comparison, $\mathrm{II}^{\perp}_{\mathbb{N}}$ was used as a fallback whenever no result was found.

S and M. For each experiment, a solver timeout of 60 seconds was used. These results are shown in Table IV. Benchmarks dlms, lms and biquad do not appear in this table because there were no allocations with a non-integer minimum II.

Compared to the existing approaches, we could identify rational-II schedules with a speedup between 1.1× and 1.5× on average across all benchmarks except vanDongen, sam and cholesky. In all cases where a solution was obtained, the optimal II could be identified. The vanDongen benchmark was chosen by Fimmel and Müller to motivate their approach. For one resource allocation, our formulation was able to find a better II than theirs. The benchmarks spline and linear phase could be solved completely by Fimmel–Müller and Moovac. But, the II found by Fimmel–Müller is worse on average. We observed that the proposed approach tends to timeout whenever S becomes larger than four, blowing up the number of variables in the ILP problem significantly. This can be detected before scheduling and a heuristic approach could identify smaller values for S such that the problem is more likely to be solvable. Moovac was able to identify 20 modulo schedules for sam, the proposed ILP found one solution and the other approaches timed out in all cases. All examined schedulers failed to find a modulo schedule for cholesky. Regarding II, the unrolling approach performs the same as the ILP formulation, but far fewer solutions were found (only 74 compared to 98), especially for the larger models. However, a larger proportion of schedules ($65/74$ instead of $49/98$) were proven to be optimal with regard to latency.

To understand the possible hardware overhead (after place and route), we studied all 71 resource allocations of the linear phase benchmark model. We define the resource usage, RU, of an implementation as $RU = slicesUsed + N \cdot DSPsUsed$, where N denotes the slice-to-DSP ratio of the given FPGA device; in our experiments, $N = 142$. The Pareto frontier for II and resource usage is shown in Figure 3. The Pareto-optimal implementations that were found using integer-IIs are

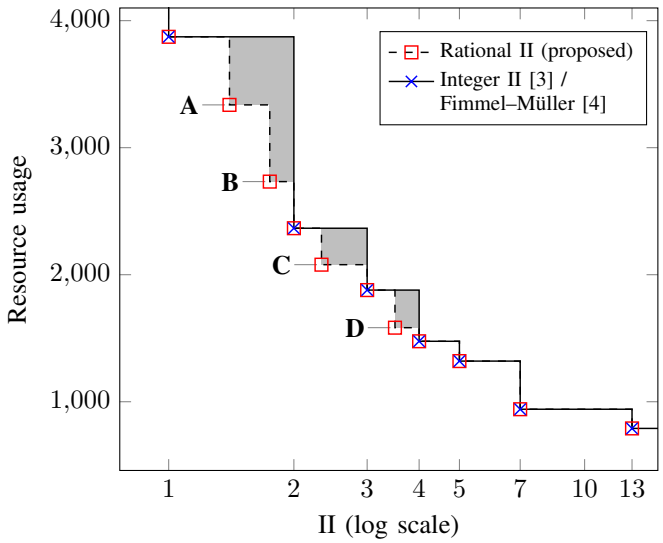

Fig. 3. Pareto-optimal implementations of the linear phase benchmark after synthesizing all possible allocations.

displayed as crosses. New Pareto-optimal designs that are revealed using our proposed rational-II formulation (labelled **A** to **D**) could only be found using our formulation. Note that all implementations are able to support the demanded 250 MHz, and implementing rational-II scheduling does not affect the operating frequency of the final hardware. Note also that, at least on this benchmark, rational-II scheduling does not change the fact that the best possible II is still 1 (top-left point), and the best possible resource usage is still provided by the bottom-right point; the value here is the finer-grained control over the design-space exploration. This 'fine-grained control' can be quantified by taking the area (highlighted in grey) between the two Pareto frontiers.

VI. Conclusion and Outlook

We present a novel ILP formulation that is able to determine optimal rational IIs whenever the number of operations in the DFG is less than about 100. Compared to state-of-the-art methods, we achieve throughput improvements of up to 1.5×. We show that in 35% of the encountered scheduling problems, speedups of 1.26× on average and up to 1.98× are possible.

To solve larger problems, heuristics for adapting S will be required. One solution could be the adaptation of iterative modulo scheduling from integer-II modulo scheduling to the rational-II case. Gradually increasing the II, such an algorithm could make repeated attempts as a fallback strategy for complex scheduling problems. The idea is that when the II is larger, the scheduling constraints are easier to satisfy.

Finally, Pareto frontiers can be improved using our approach, thus enabling a more fine-grained control over the design-space. In addition, the theoretical analysis of the minimum II in combination with synthesis results from Section V indicate that it is possible to identify resource allocations that lead to the Pareto frontier before scheduling and synthesis. We envision to reduce overall design time for multi-objective optimisation in custom hardware design by our approach significantly.

Acknowledgements

This work was carried out while the first author visited Imperial College, supported by the UK EPSRC (grant EP/P010040/1). We also acknowledge the financial support of grant ZI 762/5-1 from the German Research Foundation (DFG) and grant EP/R006865/1 from the EPSRC.

References

[1] A. E. Eichenberger and E. S. Davidson, "Efficient Formulation for Optimal Modulo Schedulers," *ACM SIGPLAN*, 1997.
[2] A. Canis, S. D. Brown, and J. H. Anderson, "Modulo SDC Scheduling with Recurrence Minimization in High-level Synthesis," in *24th Int. Conf. on Field Programmable Logic and Applications*, IEEE, 2014.
[3] J. Oppermann, A. Koch, M. Reuter-Oppermann, and O. Sinnen, "ILP-based Modulo Scheduling for High-level Synthesis," in *Int. Conf. on Compliers, Architectures, and Sythesis of Embedded Systems*, IEEE, 2016.
[4] D. Fimmel and J. Müller, "Optimal Software Pipelining under Resource Constraints," *Int. Journal of Foundations of Computer Science*, 2001.
[5] J. Oppermann, P. Sittel, M. Kumm, M. Reuter-Oppermann, A. Koch, and O. Sinnen, "Design-Space Exploration with Multi-Objective Resource-Aware Modulo Scheduling," in *25th European Conference on Parallel Processing*, 2019.
[6] Z. Zhang and B. Liu, "SDC-based Modulo Scheduling for Pipeline Synthesis," in *Int. Conf. on Comp.-Aided Design*, 2013.
[7] J. H. Patel and E. S. Davidson, "Improving the Throughput of a Pipeline by Insertion of Delays," in *ACM SIGARCH Computer Architecture News*, 1976.
[8] J. Cong and Z. Zhang, "An Efficient and Versatile Scheduling Algorithm Based on SDC Formulation," in *43rd ACM/IEEE Design Automation Conf.*, IEEE, 2006.
[9] V. H. Van Dongen, G. R. Gao, and Q. Ning, "A Polynomial Time Method for Optimal Software Pipelining," in *Parallel Processing: CONPAR 92VAPP V*, Springer, 1992.
[10] U. Meyer-Baese, A. Vera, A. Meyer-Baese, M. Pattichis, and R. Perry, "Discrete Wavelet Transform FPGA Design using MatLab/Simulink," in *Ind. Comp. Analyses, Wavelets, Unsupervised Smart Sensors, and Neural Networks*, 2006.
[11] D. Goodman and M. Carey, "Nine Digital Filters for Decimation and Interpolation," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 1977.
[12] M. Kumm and M. S. Sanjari, "Digital Hilbert Transformers for FPGA-based Phase-locked Loops," in *Int. Conf. on Field Programmable Logic and Applications*, IEEE, 2008.
[13] D. Shi and Y. J. Yu, "Design of Linear Phase FIR Filters with High Probability of Achieving Minimum Number of Adders," *Trans. on Circuits and Systems I: Regular Papers*, 2011.
[14] H. Samueli, "An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Powers-of-two Coefficients," *Trans. on Circuits and Systems*, 1989.
[15] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 2007.
[16] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. John Wiley & Sons, 2011.
[17] "Cholesky Decomposition," 2011. http://www.alterawiki.com/wiki/Floating-point_Matrix_Inversion_Example.
[18] P. Sittel, M. Kumm, J. Oppermann, K. Möller, P. Zipf, and A. Koch, "ILP-based Modulo Scheduling and Binding for Register Minimization," in *28th Int. Conf. on Field Programmable Logic and Applications*, IEEE, 2018.
[19] "Origami HLS." http://www.uni-kassel.de/go/origami.
[20] P. Sittel, J. Oppermann, M. Kumm, A. Koch, and P. Zipf, "HatScheT: A Contribution to Agile HLS," in *Int. Workshop on FPGAs for Software Programmers*, VDE, 2018.
[21] P. Sittel, T. Schönwälder, M. Kumm, and P. Zipf, "ScaLP: A Light-Weighted (MI)LP Library," in *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 2018.
[22] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, July 2011.