Optimal Software Pipelining under Resource Constraints

Dirk Fimmel & Jan Müller

SFB 358-A1-1/99

Optimal Software Pipelining under Resource Constraints

Dirk Fimmel Jan Müller
Department of Electrical Engineering
Dresden University of Technology

7 October 1999

Abstract

In this paper we present a novel approach to find an optimum loop scheduling under consideration of limited resources. The initiation interval λ is assumed to be a rational number. Our approach is formulated as a single optimization problem that can be solved using integer linear programming. The objective is to minimize the initiation interval, while both numerator and denominator of λ can be incorporated as variables of the optimization problem. The resources (functional units) may have a pipeline architecture; the approach also supports heterogeneous functional units.

1 Introduction

Software pipelining is a well-studied scheduling method to optimally exploit processor resources [1, 2, 6, 29, 12, 14, 16, 18, 19, 26, 31, 32]. Several methods stemming from VLSI design [5, 15, 21, 28] are readily applicable to software pipelining problems. The past research in the field of software pipelining can be roughly divided into heuristic approaches, and approaches using integer linear programming (ILP) formulations.
As an advantage, the methods based on heuristics can be quite fast, but they are not always guaranteed to produce an optimal scheduling. To the contrary, integer linear programming (ILP) approaches always find the optimal solution of the problem. Compared to heuristic methods, ILP algorithms are slower, such that the user has to decide whether to have an optimal solution and to put up with a longer compilation time, or to get the compilation result fast and to cope with a slower program.
Allan e.a. [2], and Rau e.a. [25] summarize a number of heuristic methods in overview papers. The important approach of decoupling dependence constraints and resource constraints (so-called decomposed software pipelining) was introduced by Gasperoni and Schwiegelshohn [14], and similarly, by Wang and Eisenbeis [32, 33]. These approaches first produce an unconstrained scheduling, and then try to restructure the result with constraints, using classical graph algorithms. Calland e.a. [6] exploit circuit retiming methods [21] to improve this restructuring, they apply integer linear programming for a partial solution of their algorithm. Wang and Parhi [31] found heuristics to improve the lower bound on the initiation interval. Vegdahl [30] proposed an extensive search through all possible schedules. This guarantees to find the minimum λ, yet with the cost of possibly enumerating a huge number of schedules.
The above-mentioned algorithms, however, are limited to integer initiation intervals.

Quinton found an approach with a rational λ, though in the field of array design [24]. The optimizations by Lavery e.a. using loop unrolling result in non-integral λ, too [20]. Another approach with a rational initiation interval, yet without regarding the resource constraints, was proposed by van Dongen [29].

Govindarajan e.a. [16, 17] and Hwang e.a. [18] formulate integer linear programming (ILP) algorithms, assuming a given integer λ. Several optimizations are performed with increasing λ, starting with the lower bound, until a valid solution is found. Hwang e.a. discuss several extensions of their approach, e.g. to handle bus and register constraints. Altman e.a. [3, 4] discuss resource constraints for pipelines having structural hazards, in conjunction with an ILP approach. Gebotys [15] presents an in-depth analysis of ILP formulations based on polyhedron theory [27]. The proposed methods concern problems of system design, but are similarly useful for software pipelining problems. With a more efficient formulation of the modulo scheduling solution space, Eichenberger e.a. [7] show that ILP algorithms can compete with heuristics in terms of compilation time. They also incorporate the assignment of instructions to functional units, yet only with one unit type.

Most stimulating to our research was the paper by Feautrier [12] where an ILP approach to software pipelining with a rational initiation interval $\lambda = A/D$ is derived. The resource constraints are translated into bi-linear inequalities, which become linear when λ is fixed, such that the determination of λ is still decoupled from the optimization process. The algorithm starts by guessing D, and then iteratively searches for A until a valid solution is found.

In this paper we overcome some of the limitations of the cited papers. We avoid the decoupling of the optimization from the search for an optimum initiation interval λ, showing that both tasks can be combined. Our ILP approach assumes a rational $\lambda = A/D$, where the computation of A and D is an integral part of the optimization. Especially for small initiation intervals, a rational λ can yield significant speed-up of the code. Moreover we offer a way to cope with a flexible assignment of operations to functional units, which is important for the use of processors having heterogeneous units. In formulation of resource constraints, we focus our attention to an optimum exploitation of functional units; though one could as well involve bus and register resources [15, 18]. The treatment of functional units with a computation time exceeding the initiation interval is also discussed. We present problem formulations for relevant optimization tasks, together with complexity estimations.

The paper is structured as follows. The next section describes the problem formulation: loop representation, resource and causality constraints. The resource constraints are discussed for a single functional unit, and for multiple functional units. After constructing the constraints we simplify them, and try to find a corresponding linear expression for use in the linear program. In Section 3 we describe the objective of the scheduling, which is again modified for the ILP. The necessary estimation of parameter bounds is given in Section 4, thereupon follows the formulation of the optimization problems and their complexities. The 6th section illustrates the approach by some example loops. In Section 7 we discuss extensions of our approach, such as the flexible assignment of functional units, and the treatment of D as a variable, which is followed by the Conclusions.

2 Problem description

2.1 Loop representation

In this paper we restrict our attention to single loops:

$$\begin{array}{ll} \text{for } i = 1 \text{ to } N & \\ S_j: \quad y_j[i] = F_j(\cdots, y_k[i - d^l_{jk}], \cdots), \quad 1 \leq j, k \leq M, 1 \leq l \leq M_{jk}, & \\ \text{end for,} & \end{array} \tag{1}$$

where F_j are single valued operations; we use indices i for iterations, j and k for statements (assignments), the argument y_k may appear l-fold.
We suppose that the loop has a single assignment form, i.e. each instance of a variable y_j is assigned only once in the loop. The single assignment form may result from application of an array dataflow analysis and a renaming of variables [11]. Then all data dependencies are flow dependencies which can be represented in a reduced dependence graph (RDG). The statements form the M nodes $S_j \in \mathcal{S}$ of the RDG $\langle \mathcal{S}, \mathcal{E} \rangle$. Directed edges $(S_j, S_k) \in \mathcal{E}$ are data dependencies weighted by the dependence distances d^l_{jk}. The weight of an edge $e \in \mathcal{E}$ is called $d(e)$, the source of this edge $\sigma(e)$ and the sink $\delta(e)$.
Our aim is to find a D-periodic scheduling of the following form:

$$\Theta_j(i) = \left\lfloor \frac{Ai + B_j}{D} \right\rfloor, \quad 1 \leq j \leq M, \tag{2}$$

where $A, D, B_j \in \mathbb{Z}$ and A and D are relatively prime. The quotient A/D is called *initiation interval* λ, and B_j/D are individual time offsets for each statement S_j.

2.2 Resource constraints

2.2.1 Functional units

At first we have to introduce a bunch of variables describing functional units and relationships between them. A set of functional units $u_l \in \mathcal{U}$ available for the computation of operations F_j is supposed to be given. Each functional unit u_l is associated with a latency $L_{l,j} \in \mathbb{Z}$, defining the time between the start of the computation and the availability of the result, and with an offset time $O_{l,j} \in \mathbb{Z}$, defining the minimum time between the start of two computations if u_l has a pipeline architecture. Latency $L_{l,j}$ and offset $O_{l,j}$ of u_l may depend on operation F_j if u_l is able to perform different operations. Furthermore, the number of instances of u_l is limited to U_l. Finally we introduce a function $f\colon \mathbb{Z} \to \mathbb{Z}$, expressing that operation F_j is evaluated by functional unit u_l if $f(j) = l$. In this section we assume that the assignment $f(j) = l$, $1 \leq j \leq M$, is fixed, whereas the more general case of assigning a functional unit u_l to each operation F_j by the linear program is discussed in Section 7.

2.2.2 Single functional unit

We now draw up a resource constraint for operations requiring the same (single) resource. This constraint is simplified afterwards, and then linearized for use in a linear program.

The resource constraint results from the limitation of functional units u_l used for the computation of operations F_j. Suppose that two operations F_j and F_k share a functional unit u_l. Then a conflict free use of u_l implies that F_j can only be scheduled after the offset time of F_k has passed by, or vice versa:

$$\forall i,i' \in \mathbb{Z}. \left\lfloor \frac{Ai+B_j}{D} \right\rfloor \geq \left\lfloor \frac{Ai'+B_k}{D} \right\rfloor + O_{l,k} \vee \left\lfloor \frac{Ai'+B_k}{D} \right\rfloor \geq \left\lfloor \frac{Ai+B_j}{D} \right\rfloor + O_{l,j}. \tag{3}$$

Throughout the paper, we assume that the upper loop bound, N, is sufficiently large.

Theorem 1 *(3) is equivalent to:*

$$\forall i,i' \in \mathbb{Z}. \frac{Ai+B_j}{D} \geq \frac{Ai'+B_k}{D} + O_{l,k} \vee \frac{Ai'+B_k}{D} \geq \frac{Ai+B_j}{D} + O_{l,j}. \tag{4}$$

For the sake of completeness the proofs of all theorems are presented in the Appendix. Replacing i and i' by $x = i - i'$ in (4) leads to:

$$\forall x \in \mathbb{Z}. Ax + B_j - B_k \geq D\, O_{l,k} \vee -Ax + B_k - B_j \geq D\, O_{l,j}.$$

Now we observe that the first inequality is only valid if $x > (B_k - B_j)/A$ since $D\, O_{l,k} > 0$. Considering the integer constraint of x, we have $x \geq \lfloor (B_k - B_j)/A \rfloor + 1$. The same discussion leads to $x \leq \lfloor (B_k - B_j)/A \rfloor$ for the second inequality. Since only the extremal cases are of interest in both inequalities, it is sufficient to examine the first and second inequalities only for $x = \lfloor (B_k - B_j)/A \rfloor + 1$ and $x = \lfloor (B_k - B_j)/A \rfloor$, respectively:

$$\begin{aligned} A(\lfloor (B_k - B_j)/A \rfloor + 1) + B_j - B_k &\geq D\, O_{l,k}, \\ -A(\lfloor (B_k - B_j)/A \rfloor) + B_k - B_j &\geq D\, O_{l,j}. \end{aligned} \tag{5}$$

Next, we introduce a decomposition of B_j, B_k modulo A: $B_j = \lfloor B_j/A \rfloor A + b_j$ and $B_k = \lfloor B_k/A \rfloor A + b_k$. Then (5) can be rewritten as:

$$\begin{aligned} A(\lfloor B_k/A \rfloor - \lfloor B_j/A \rfloor + \lfloor (b_k - b_j)/A \rfloor + 1) + \lfloor B_j/A \rfloor A + b_j - \lfloor B_k/A \rfloor A - b_k &\geq D\, O_{l,k}, \\ -A(\lfloor B_k/A \rfloor - \lfloor B_j/A \rfloor + \lfloor (b_k - b_j)/A \rfloor) + \lfloor B_k/A \rfloor A + b_k - \lfloor B_j/A \rfloor A - b_j &\geq D\, O_{l,j}, \end{aligned}$$

which leads to:

$$\begin{aligned} A(\lfloor (b_k - b_j)/A \rfloor + 1) + b_j - b_k &\geq D\, O_{l,k}, \\ -A\lfloor (b_k - b_j)/A \rfloor + b_k - b_j &\geq D\, O_{l,j}. \end{aligned} \tag{6}$$

From $-A + 1 \leq b_k - b_j \leq A - 1$ follows that $\lfloor (b_k - b_j)/A \rfloor \in \{-1, 0\}$, more precisely: $\lfloor (b_k - b_j)/A \rfloor = \begin{cases} 0, & b_k \geq b_j \\ -1, & b_k < b_j \end{cases}$. Introduction of a binary variable $\alpha_{jk} \in \{0,1\}$ allows to rewrite (6) as follows:

$$\begin{aligned} A\alpha_{jk} + b_j - b_k &\geq D\, O_{l,k}, \\ -A(\alpha_{jk} - 1) + b_k - b_j &\geq D\, O_{l,j}. \end{aligned} \tag{7}$$

Note that if (7) is true, there is exactly one integer solution for α_{jk}.

Since we want to use A as a variable in linear programming, we have to linearize both $A\alpha_{jk}$ and the modulo decomposition of B_j. To this end we need an upper bound A^{max} of A, i.e. $A \leq A^{max}$. The bounds on the variables are estimated in Section 4.
For the linearization of $A\alpha_{jk}$ we propose two approaches.

First, the term $A\alpha_{jk}$ can be replaced by a new variable $A_{jk} \in \mathbb{Z}$, where the following constraints have to be satisfied for A_{jk}:

$$\begin{array}{rcl} 0 & \leq A_{jk} \leq & A, \\ A-(1-\alpha_{jk})A^{max} & \leq A_{jk} \leq & \alpha_{jk}A^{max}. \end{array} \tag{8}$$

The second way, which makes the introduction of A_{jk} superfluous, is based on an observation of all solutions of (7). Then we have:

$$\left.\begin{array}{rl} b_j - b_k \geq D\,O_{l,k}, & (\star 1) \\ A + b_k - b_j \geq D\,O_{l,j}, & (\star 2) \end{array}\right\} b_k < b_j, (\alpha_{jk}=0),$$
$$\left.\begin{array}{rl} A + b_j - b_k \geq D\,O_{l,k}, & (\star 3) \\ b_k - b_j \geq D\,O_{l,j}, & (\star 4) \end{array}\right\} b_k \geq b_j, (\alpha_{jk}=1).$$

Note that $(\star 2)$ and $(\star 3)$ are fulfilled if $b_k \geq b_j$ and $b_k < b_j$, respectively, since $A \geq D\,O_{l,j}$. For $(\star 1)$ and $(\star 4)$ we introduce "correcting terms" $\alpha_{jk}C_0$ and $(1-\alpha_{jk})C_0$, such that the inequalities are always satisfied if $b_k \geq b_j$ and $b_k < b_j$, respectively. Now we can write:

$$\begin{array}{rcl} b_j - b_k & \geq & D\,O_{l,k} - \alpha_{jk}C_0, \\ A + b_k - b_j & \geq & D\,O_{l,j}, \\ A + b_j - b_k & \geq & D\,O_{l,k}, \\ b_k - b_j & \geq & D\,O_{l,j} - (1-\alpha_{jk})C_0. \end{array} \tag{9}$$

For the estimation of the constant C_0 we consider its extremal values. On the one hand we have $b_k - b_j \geq -A+1 \geq -A^{max}+1$, and on the other hand $D\,O_{l,j} \leq A \leq A^{max}$. Hence, we get $C_0 \geq 2A^{max}-1$.
Comparison of (9) to (7) and (8) yields 4 versus 6 inequalities, and in (7) and (8) an additional integer variable is necessary. Yet, the large coefficient C_0 of variable α_{jk} in (9) leads to an ill-conditioned linear program.

In order to linearize the decomposition of B_j, we introduce new variables $\underline{B}_j, p_j \in \mathbb{Z}$, such that $\underline{B}_j = p_jA = \lfloor B_j/A \rfloor A$ and $B_j = \underline{B}_j + b_j$. Again, an upper bound p^{max} of p_j, i.e. $p_j \leq p^{max}$ is necessary. Then $\underline{B}_j$ can be expressed using $p^{max}+1$ binary variables $\beta_{jk} \in \{0,1\}$ in the following way:

$$\begin{array}{rcll} 0 & \leq \underline{B}_j \leq & p^{max}A, & \\ kA-(1-\beta_{jk})kA^{max} & \leq \underline{B}_j \leq & kA+(1-\beta_{jk})(p^{max}-k)A^{max}, & 0 \leq k \leq p^{max}, \\ \sum\limits_{k=0}^{p^{max}} \beta_{jk} & = 1. & & \end{array} \tag{10}$$

Variable p_j can be expressed by $p_j = \sum\limits_{k=0}^{p^{max}} k\beta_{jk}$ but is not needed further. For $\beta_{jk}=1$ we have $kA \leq \underline{B}_j \leq kA$, i.e. $\underline{B}_j = kA$. For $\beta_{jk}=0$ the left hand side is $kA - kA^{max} \leq 0$, and the right hand side is $kA+(p^{max}-k)A^{max} \geq kA+(p^{max}-k)A = p^{max}A$. Hence, we have again $0 \leq \underline{B}_j \leq p^{max}A$.

2.2.3 Multiple functional units

If we consider multiple functional units being able to perform an operation, we have to deduce new resource constraints, which are then simplified and linearized in this subsection.

Suppose that the i-th iteration of statement S_j is performed by functional unit u_l. Then u_l is busy in the interval $[\tau_j(i), \tau_j(i)+O_{l,j}-1]$. This leads to the following resource constraint in the case of U_l available functional units u_l:

$$\forall t \in \mathbb{Z}.\forall u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t - O_{l,j} + 1 \leq \lfloor (Ai + B_j)/D \rfloor \leq t\}| \leq U_l, \tag{11}$$

i. e. the number of functional units being busy at a time must not exceed the number of available units. The assignment $f(j) = l$ is assumed to be fixed in this section, as already mentioned before.
Omitting the floor operator in (11) leads to:

$$\forall t \in \mathbb{Z}.\forall u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t - O_{l,j} + 1 \leq (Ai + B_j)/D < t + 1\}| \leq U_l. \tag{12}$$

Theorem 2 *(12) is equivalent to:*

$$\underset{0 \leq t' \leq A-1}{\forall \mathrm{t}' \in \mathbb{Z}.}\ \forall u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t' - D\,O_{l,j} \leq Ai + B_j < t'\}| \leq U_l. \tag{13}$$

Using the decomposition $B_j = \lfloor B_j/A \rfloor A + b_j$, (13) can be transformed into:

$$\underset{0 \leq t' \leq A-1}{\forall \mathrm{t}' \in \mathbb{Z}.}\ \forall u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t' - D\,O_{l,j} \leq Ai + b_j < t'\}| \leq U_l.$$

Since all parts of the inequality are integers, we can write:

$$\underset{0 \leq t' \leq A-1}{\forall \mathrm{t}' \in \mathbb{Z}.}\ \forall u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t' - D\,O_{l,j} \leq Ai + b_j \leq t' - 1\}| \leq U_l. \tag{14}$$

To further restrict this expression, we now want to estimate values of i in (14). We have $i \leq (t' - b_j - 1)/A \leq (A - 2)/A$, and hence $i \leq \lfloor (A - 2)/A \rfloor = 0$. On the other hand we have $i \geq (t' - D\,O_{l,j} - b_j)/A \geq (-A - A + 1)/A$, which leads to $i \geq \lceil (-2A + 1)/A \rceil = -1$. Combining both results we get $i \in \{-1, 0\}$. This allows to rewrite (14) as follows:

$$\underset{0 \leq t' \leq A-1}{\forall \mathrm{t}' \in \mathbb{Z}.}\forall u_l \in \mathcal{U}.|\{S_j|f(j) = l \wedge t' - DO_{l,j} \leq b_j \leq t' - 1 \vee t' - DO_{l,j} \leq b_j - A \leq t' - 1\}| \leq U_l. \tag{15}$$

Next, we introduce decision variables $\gamma_{j,t'} \in \{0, 1\}$ such that:

$$\gamma_{j,t'} = \begin{cases} 1 & \text{if } t' - D\,O_{l,j} \leq b_j \leq t' - 1 \vee t' - D\,O_{l,j} \leq b_j - A \leq t' - 1, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Using variables $\gamma_{j,t'}$, (15) can be transformed into:

$$\underset{0 \leq t' \leq A-1}{\forall \mathrm{t}' \in \mathbb{Z}.}\ \forall u_l \in \mathcal{U}.\ \sum_{\substack{j=1 \\ f(j)=l}}^{M} \gamma_{j,t'} \leq U_l. \tag{17}$$

For the determination of $\gamma_{j,t'}$, we observe two properties of (16). First, $b_j - A \leq t' - 1$ is always true. Second, we have $t' - D\,O_{l,j} \leq t' - 1 < A + t' - D\,O_{l,j}$. This allows to rewrite (16) as follows:

$$\gamma_{j,t'} = \begin{cases} 0 & \text{if } b_j \leq t' - D\,O_{l,j} - 1 \vee t' \leq b_j \leq A + t' - D\,O_{l,j} - 1, \\ 1 & \text{otherwise.} \end{cases}$$

We encode the "$\vee$" operator by another binary variable $\nu_{j,t'} \in \{0,1\}$, where $b_j \leq t' - D\,O_{l,j} - 1$ is considered if $\nu_{j,t'} = 0$ and $t' \leq b_j \leq A + t' - D\,O_{l,j} - 1$ if $\nu_{j,t'} = 1$. (16) can then be formulated in the following way:

$$\begin{aligned} &b_j \leq t' - D\,O_{l,j} - 1 + (\gamma_{j,t'} + \nu_{j,t'})C_1, \\ &b_j \geq t' - (\gamma_{j,t'} + (1 - \nu_{j,t'}))C_2, \\ &b_j \leq A + t' - D\,O_{l,j} - 1 + (\gamma_{j,t'} + (1 - \nu_{j,t'}))C_3. \end{aligned} \tag{18}$$

To determine the constants C_1, C_2 and C_3, we consider extremal cases:

$$\begin{array}{llll} A - 1 \leq -A - 1 + C_1 & \rightarrow\ C_1 \geq 2A & \rightarrow\ C_1 = 2A^{max}, \\ 0 \geq A - 1 - C_2 & \rightarrow\ C_2 \geq A - 1 & \rightarrow\ C_2 = A^{max} - 1, \\ A - 1 \leq A - A - 1 + C_3 & \rightarrow\ C_3 \geq A & \rightarrow\ C_3 = A^{max}. \end{array}$$

The inequalities of (18) force $\gamma_{j,t'} = 1$ in the "if" case of (16). The "otherwise" case is more relaxed, yet only as long as the optimality of the initiation interval A/D is ensured. Consideration of $t' \geq A$ does not affect the solution since we have $\gamma_{j,t'} = \gamma_{j,t'-A} = 1$ for all $t' \in [A, b_j + D\,O_{l,j}]$. For $t' \geq b_j + D\,O_{l,j} + 1$ the first inequality of (18) is always true, which leads to $\gamma_{j,t'} = 0$.

Remark: Let us now consider a special case, supposing that D and $f(j)$ are fixed. Then $D\,O_{l,j}$ is fixed as well, which allows a reformulation of (17). We introduce a binary decomposition of $b_j = \sum_{k=0}^{A^{max}-1} k\epsilon_{j,k}$ as follows:

$$\sum_{k=0}^{A^{max}-1} \epsilon_{j,k} = 1, \quad \epsilon_{j,k} \in \{0,1\}, \quad \sum_{k=0}^{A^{max}-1} k\epsilon_{j,k} \leq A - 1. \tag{19}$$

Furthermore we need a binary decomposition of $c_j = A - (b_j + D\,O_{l,j}) = \sum_{k=0}^{D\,O_{l,j}-1} k\chi_{j,k}$:

$$\sum_{k=0}^{D\,O_{l,j}-1} \chi_{j,k} = 1, \quad \chi_{j,k} \in \{0,1\}, \quad A - \sum_{k=1}^{A^{max}-1} k\epsilon_{j,k} + \sum_{k=0}^{D\,O_{l,j}-1} (k+1)\chi_{j,k} \geq 1 + D\,O_{l,j}. \tag{20}$$

The coefficient $(k+1)$ forces $\chi_{j,0} = 1$ for $A - \sum_{k=1}^{A^{max}-1} k\epsilon_{j,k} = D\,O_{l,j}$. Now we can express (16) as follows:

$$\gamma_{j,t'} = \sum_{k=t'-D\,O_{l,j}}^{t'-1} \epsilon_{j,k} + \sum_{k=t'}^{D\,O_{l,j}-1} \chi_{j,k}.$$

Variable $\gamma_{j,t'}$ is not needed further since we can replace (17) by:

$$\underset{0 \leq t' \leq A-1}{\forall \mathrm{t}' \in \mathbb{Z}.}\ \forall u_l \in \mathcal{U}.\ \sum_{\substack{j=1 \\ f(j)=l}}^{M} \left(\sum_{k=t'-D\,O_{l,j}}^{t'-1} \epsilon_{j,k} + \sum_{k=t'}^{D\,O_{l,j}-1} \chi_{j,k} \right) \leq U_l. \tag{21}$$

Higher values of c_j are suppressed by the optimality demand of the linear program. Finally, we point out that in the case of a fixed λ it is sufficient to decompose b_j, as described in (19). Then (21) can be replaced by:

$$\underset{0 \leq t' \leq A-1}{\forall \mathrm{t}' \in \mathbb{Z}.}\ \forall u_l \in \mathcal{U}.\ \sum_{\substack{j=1 \\ f(j)=l}}^{M} \left(\sum_{k=t'-D\,O_{l,j}}^{t'-1} \epsilon_{j,k} + \sum_{k=A+t'-D\,O_{l,j}}^{A+t'-1} \epsilon_{j,k} \right) \leq U_l,$$

which is equivalent to the formulation in [12]. ■

2.3 Causality constraint

After having derived constraints due to limitations of functional units, i.e. of the target machine, we now discuss restrictions resulting from the program sequence itself, namely from dependencies between loop variables.
A scheduling is called causal if all data dependencies of the RDG are satisfied. For our assumption of only flow dependencies which are uniform (see Section 2.1), the causality constraint can be formulated as follows:

$$\forall e \in \mathcal{E}.\ \underset{1 \le i \le N}{\forall \mathrm{i} \in \mathbb{Z}}\ .\left\lfloor \frac{Ai + Ad(e) + B_{\delta(e)}}{D} \right\rfloor \ge \left\lfloor \frac{Ai + B_{\sigma(e)}}{D} \right\rfloor + L_{f(\sigma(e)),\sigma(e)}. \tag{22}$$

Theorem 3 *For a sufficiently large N, (22) is equivalent to:*

$$\forall e \in \mathcal{E}. Ad(e) + B_{\delta(e)} - B_{\sigma(e)} \ge D\, L_{f(\sigma(e)),\sigma(e)}. \tag{23}$$

3 Objective of the scheduling

For the completion of the linear program we still lack an objective function to be minimized. In this section we show how to deduce an appropriate function.
The aim of loop scheduling is usually to minimize latency L. Here L is given by:

$$L = \max_{1 \le j \le M} \max_{1 \le i \le N} (\tau_j(i) + L_{f(j),j}) - \min_{1 \le j \le M} \min_{1 \le i \le N} \tau_j(i).$$

Since $\tau_j(i)$ is increasing monotonically in i, we can write:

$$L = \max_{1 \le j \le M} (\lfloor (AN + B_j)/D \rfloor + L_{f(j),j}) - \min_{1 \le j \le M} \lfloor (A + B_j)/D \rfloor.$$

Omitting the brackets leads to:

$$DL = A(N-1) + \max_{1 \le j \le M} (B_j + DL_{f(j),j}) - \min_{1 \le j \le M} B_j. \tag{24}$$

The term $\min\limits_{1 \le j \le M} B_j$ can be omitted, since a retiming is always possible such that $\underset{1 \le j \le M}{\exists j.}\ B_j = 0$, and the optimality of (24) forces exactly this case. The term $\max\limits_{1 \le j \le M} (B_j + DL_{f(j),j})$ can be replaced by an integer variable $z \in \mathbb{Z}$, where:

$$z \ge B_j + DL_{f(j),j}, \quad 1 \le j \le M. \tag{25}$$

Again, a minimum L forces $z = \max\limits_{1 \le j \le M} (B_j + DL_{f(j),j})$.
In (24) only the factor DL is non-linear, if we suppose that $f(j)$ is fixed. To circumvent this non-linearity we introduce an upper bound D^{max} of D, i.e. $D \le D^{max}$.
A straightforward way is now to solve D^{max} problems with $D = k$, $1 \le k \le D^{max}$, by choosing L as the objective function, and to select the minimum L of all problems.
If we want to restrict the effort to a single problem, we have to introduce a binary decomposition of D:

$$D = \sum_{k=1}^{D^{max}} k\delta_k, \quad \sum_{k=1}^{D^{max}} \delta_k = 1, \quad \delta_k \in \{0, 1\}. \tag{26}$$

This allows to replace (24) by:

$$A(N-1)+z-(1-\delta_k)C_8 \leq kL \leq A(N-1)+z+(1-\delta_k)C_9, \quad 1 \leq k \leq D^{max}. \qquad (27)$$

Consideration of extreme cases leads to:

$$\begin{array}{ll} A(N-1)+(p^{max}+1)A-1+D\max\limits_{1\leq j\leq M} L_{f(j),j}-C_8 \leq 0 & \rightarrow C_8 \geq (N+p^{max})A^{max}-1+D^{max}\max\limits_{1\leq j\leq M} L_{f(j),j}, \\ kL^{max} \leq A(N-1)+C_9 & \rightarrow C_9 \geq D^{max}L^{max}-(N-1), \end{array}$$

where $L^{max} = A^{max}(N+p^{max})-1+\max\limits_{1\leq j\leq M} L_{f(j),j}$.
Both constants C_8 and C_9 depend on the size parameter N, which leads to very ill-conditioned linear programs.
A possibility to overcome this problem is to relax (24) by omitting the term $\max\limits_{1\leq j\leq M}(B_j + DL_{f(j),j})$. This leads to a relative error:

$$\left|\frac{L-z}{L}\right| \leq \left|1-\frac{A(p^{max}+1)-1+\max\limits_{1\leq j\leq M} L_{f(j),j}}{A(N+p^{max})-1+\max\limits_{1\leq j\leq M} L_{f(j),j}}\right|,$$

which can be neglected if the number of iterations, N, is sufficiently large. The minimization of L is now equivalent to the minimization of the initiation interval $\lambda = A/D, \lambda \in \mathbb{R}$, which is the usual objective in software pipelining. The initiation interval λ can be expressed as follows:

$$A-(1-\delta_k)C_{10} \leq k\lambda \leq A+(1-\delta_k)C_{11}, \quad 1 \leq k \leq D^{max}, \qquad (28)$$

where the constants C_{10} and C_{11} are estimated as:

$$\begin{array}{lll} A-C_{10} \leq 1 & \rightarrow & C_{10} \geq A^{max}-1, \\ kA/D \leq A+C_{11} & \rightarrow & C_{11} \geq (D^{max}-1)A^{max}. \end{array}$$

To obtain an integer initiation interval as the objective function, $\lambda \in \mathbb{R}$ is then replaced by $\Lambda \in \mathbb{Z}$, where $\Lambda = G\lambda$ and $G = lcm(\{1, \cdots, D^{max}\})$ (lcm abbreviates least common multiple). (28) has to be replaced by:

$$G(A-(1-\delta_k)C_{10}) \leq k\Lambda \leq G(A+(1-\delta_k)C_{11}), \quad 1 \leq k \leq D^{max}.$$

The formulation of the objective function does not ensure that A and D are relatively prime. To cope with this problem, the following theorem can be applied.

Theorem 4 *Suppose that $H = gcd\{A, D\}$. Then we have:*

$$\forall i \in \mathbb{Z}. \left\lfloor \frac{Ai+B_j}{D} \right\rfloor = \left\lfloor \frac{A'i+B'_j}{D'} \right\rfloor, \qquad (29)$$

where $A' = A/H$, $D' = D/H$ and $B'_j = \lfloor B_j/H \rfloor$.

The compliance with resource and causality constraints is ensured since the proofs of Theorems 1 - 3 do not require that A and D are relatively prime in directions (4) $\rightarrow$ (3), (13) $\rightarrow$ (12) and (23) $\rightarrow$ (22).
Remark: A corollary of Theorem 4 is the fact that it is sufficient
either to consider only those D such that $\forall_{D<D'\leq D^{max}} D' \in \mathbb{Z}.\ D' \not\equiv 0 \pmod{D}$ in (26) and (28),
or to choose a fixed $D = lcm\{1, \cdots, D^{max}\}$.
■

4 Estimation of bounds

As the reader will have noticed, we have used bounds on several variables in the last sections, which are now to be estimated.
We start at determining an upper bound λ^{seq} and two lower bounds λ^{res}, and λ^{par} of the initiation interval λ.
An upper bound λ^{seq} of λ is given when the loop body of one iteration is executed sequentially, i.e. $\lambda^{seq} = \sum_{j=1}^{M} L_{f(j),j}$.
The maximum utilization of resources gives rise to a lower bound λ^{res} of λ. This bound can be computed as $\lambda^{res} = \max_{u_l \in \mathcal{U}} \sum_{\substack{j=1 \\ f(j)=l}}^{M} O_{l,j}/U_l$.
Another lower bound λ^{par} of λ arises from assuming unlimited resources which results in the maximum parallelism of the loop. It is well known that $\lambda^{par} = \max_{\mathcal{C} \subseteq \mathcal{E}} \{L(\mathcal{C})/d(\mathcal{C})\}$, where $\mathcal{C}$ is a cycle in $\langle \mathcal{V}, \mathcal{E} \rangle$, $d(\mathcal{C}) = \sum_{e \in \mathcal{C}} d(e)$ and $L(\mathcal{C}) = \sum_{e \in \mathcal{C}} L_{f(\sigma(e)),\sigma(e)}$.

Now we are able to estimate A^{max} and D^{max}.
Following the discussion in [12], a suitable estimation would be

$$D^{max} = \max\{den(\lambda^{res}), \max_{\substack{\mathcal{C} \subseteq \mathcal{E} \\ L(\mathcal{C})/d(\mathcal{C}) = \lambda^{par}}} d(\mathcal{C})\},$$

where $den(\lambda^{res})$ is the denominator of λ^{res}. We refer to the discussion on the impact of a large D in Section 7.
The upper bound A^{max} of A is simply $A^{max} = \lambda^{seq} D^{max}$ if D is treated as a variable, and otherwise $A^{max} = \lambda^{seq} D$. Since λ^{seq} is a very pessimistic assumption, one can try to solve the linear program with integer λ ($D = 1$) first, and use the bound $A^{max} = \lambda_{D=1} D^{max}$ afterwards. The usage of a lower bound A^{min} of A given by $A^{min} = \max\{\lceil \lambda^{par} \rceil, \lceil \lambda^{res} \rceil\}$, or $A^{min} = D \max\{\lceil \lambda^{par} \rceil, \lceil \lambda^{res} \rceil\}$, if D is fixed, respectively, may reduce the solution time of the linear program. The lower bound of A can be further improved by methods presented in [31].
The value of p^{max} can be established by application of a list scheduling [19] as proposed in [18]. A rough estimation is to use $p^{max} = \max_{e \in \mathcal{E}} d(e)$, whereas experimental results show that $p^{max} = 1$ suffices in most cases.

5 Linear program

In this section we summarize necessary constraints for different problem formulations and present some complexity measures. In Table 1 the constraints of the linear programs are collected. The complexity of the linear programs can be estimated using the number of variables in Table 1, and the number of inequalities/equations in Table 2.
We introduce the set $\mathcal{V}_l = \{i | f(i) = l \wedge u_l \in \mathcal{U}\}$ to give complexity measures in a more compact form.

U_l	D	object.	constraints	number of variables integer	binary
1	fixed	min A	9 + 10 + 23	$2M+1$	$\sum_{u_l \in \mathcal{U}} \lvert\mathcal{V}_l\rvert(\lvert\mathcal{V}_l\rvert - 1) + M(p^{max} + 1)$
1	free	min λ	9 + 10 + 23 + 26,28	$2M+3$	$\sum_{u_l \in \mathcal{U}} \lvert\mathcal{V}_l\rvert(\lvert\mathcal{V}_l\rvert - 1) + M(p^{max} + 1) + D^{max}$
≥ 2	fixed	min A	19,20,21 + 10 + 23	$2M+1$	$\sum_{j=1}^{M} DO_{f(j),j} + MA^{max} + M(p^{max}+1)$
≥ 2	free	min λ	17,18+10+23+26,28	$2M+3$	$2MA^{max} + M(p^{max} + 1) + D^{max}$

Table 1: Constraints of the linear programs

constraint	number of inequalities/equations
9	$4 \sum_{u_l \in \mathcal{U}} \lvert\mathcal{V}_l\rvert(\lvert\mathcal{V}_l\rvert - 1)$
10	$M(2(p^{max} + 1) + 1)$
17	$\lvert\mathcal{U}\rvert A^{max}$
18	$3MA^{max}$
19	3
20	2
21	$\lvert\mathcal{U}\rvert A^{max}$
23	$\lvert\mathcal{E}\rvert$
26	2
28	$2D^{max}$

Table 2: Number of inequalities/equations

6 Experimental results

Example 1 (Program: WDF)
This program represents a second order wave digital filter [13]. One adder and one multiplication

for $i = 1$ to N
S_1 $a[i] = X[i] + e[i-1]$
S_2 $b[i] = a[i] - g[i-1]$
S_3 $c[i] = b[i] + e[i]$
S_4 $d[i] = \gamma_1 * b[i]$
S_5 $e[i] = d[i] + e[i-1]$
S_6 $f[i] = \gamma_2 * b[i]$
S_7 $g[i] = f[i] + g[i-1]$
S_8 $Y[i] = c[i] - g[i]$
end for

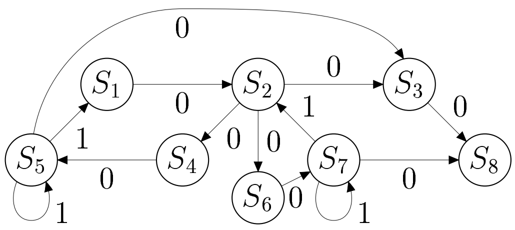

Figure 1: Program code and RDG of Example 1

unit with an evaluation time of one and three cycles, respectively, are supposed to be available. Analyzing the parallelism of the algorithm and the resource utilization we have the lower bounds $\lambda^{par} = 6$, *and* $\lambda^{res} = 6$, *respectively. The solution of the linear program for* $D = 1$ *with* $A^{max} = 12$ *and* $p^{max} = 1$ *leads to the optimum* $\lambda = 8$ *after* 1.8 *sec of CPU time on a SUN SPARCstation 10.*

We get the following scheduling functions:

$$\tau_1(i) = 8i + 0,\quad \tau_2(i) = 8i + 2,\quad \tau_3(i) = 8i + 7,\quad \tau_4(i) = 8i + 3,$$
$$\tau_5(i) = 8i + 6,\quad \tau_6(i) = 8i + 6,\quad \tau_7(i) = 8i + 9,\quad \tau_8(i) = 8i + 11.$$

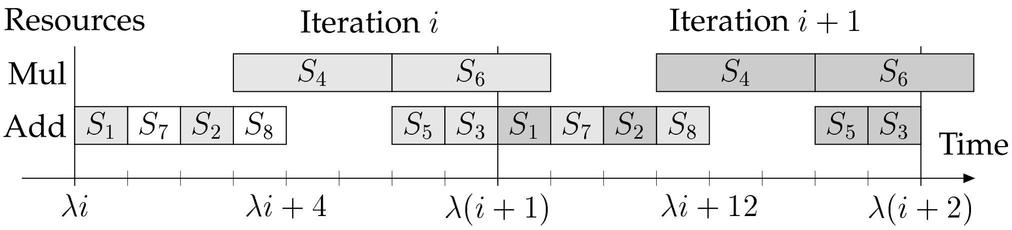

Figure 2: Scheduling of Example 1; different grey levels represent different iterations

Example 2 (Program: Calland [6])

for $i = 1$ to N
S_1 $a[i]{=}c[i-1]$
S_2 $b[i]{=}a[i-2] * d[i-1]$
S_3 $c[i]{=}b[i] + 1$
S_4 $d[i]{=}f[i-1]/3$
S_5 $e[i]{=}sin(f[i-2])$
S_6 $f[i]{=}log(b[i] + e[i])$
end for

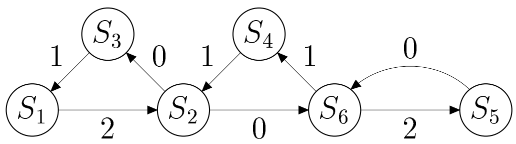

Figure 3: Program code and RDG of Example 2

In this example two identical ALUs are given, and the evaluation times for the operations S_j are $L_{1,1} = 1$, $L_{1,2} = 4$, $L_{1,3} = 2$, $L_{1,4} = 6$, $L_{1,5} = 10$, and $L_{1,6} = 14$. The lower bounds of the initiation interval λ are $\lambda^{par} = 12$ and $\lambda^{res} = 37/2$. Following Remark 2 it is sufficient to solve the problem for $D = 2$. With $p^{max} = 2$ and $A^{max} = 74$, the optimal solution of $\lambda = 37/2$ is found in 38 seconds. We get the following scheduling functions:

$$\tau_1(i) = \lfloor (37i + 23)/2 \rfloor,\quad \tau_2(i) = \lfloor (37i + 11)/2 \rfloor,\quad \tau_3(i) = \lfloor (37i + 19)/2 \rfloor,$$
$$\tau_4(i) = \lfloor (37i + 25)/2 \rfloor,\quad \tau_5(i) = \lfloor (37i + 0)/2 \rfloor,\quad \tau_6(i) = \lfloor (37i + 20)/2 \rfloor.$$

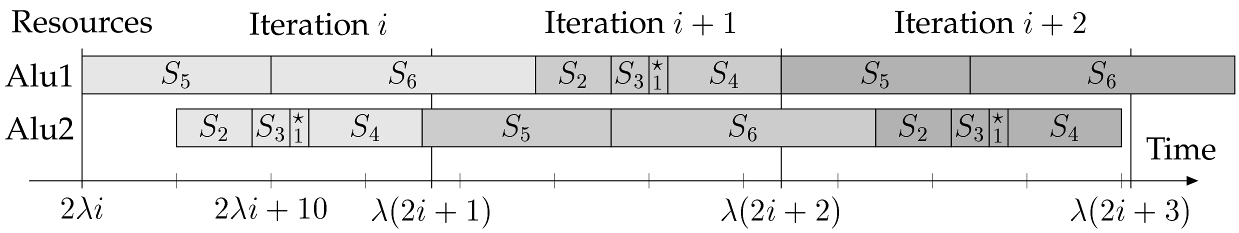

Figure 4: Scheduling of Example 2

Example 3 (Program: van Dongen [29])

for $i = 1$ to N
S_1 $\quad a[i] = h[i-1] + R[i]$
S_2 $\quad b[i] = a[i-1] + a[i]$
S_3 $\quad c[i] = b[i-1] + b[i]$
S_4 $\quad d[i] = c[i-1] + c[i]$
S_5 $\quad e[i] = d[i-2] + d[i-1]$
S_6 $\quad f[i] = e[i-2] + e[i-1]$
S_7 $\quad g[i] = f[i] + X[i]$
S_8 $\quad h[i] = g[i] + Y[i]$
S_9 $\quad i[i] = j[i-1] + d[i]$
S_{10} $\quad j[i] = i[i] + Z[i]$
end for

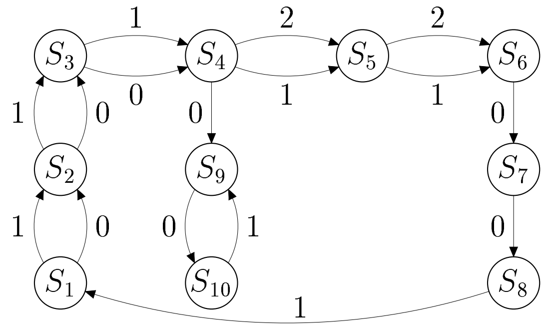

Figure 5: Program code and RDG of Example 3

We consider two given adders which are supposed to have a pipeline architecture. The evaluation of an addition takes two cycles, and an adder can start the next evaluation after one cycle. The initiation interval is bound to $\lambda^{res} = 5$, and $\lambda^{par} = 16/3$, respectively. Thus we have $\lambda \geq 16/3$ and $D^{max} = 3$.
First we solve the problem for $D = 1$ with $A^{max} = 20$ and $p^{max} = 2$. We get the solution $\lambda = 6$ after 2.4 seconds. With the new upper bound $A^{max} = 6$, we solve the problem for $D = 2$, and $D = 3$, respectively. Each problem requires approximately 18 seconds, and the solutions are $\lambda = 11/2$ for $D = 2$, and $\lambda = 17/3$ for $D = 3$. We observe that none of the selection strategies for D^{max} discussed in [12] would lead to the optimal solution of $\lambda = 11/2$. The resulting scheduling functions are:

$$\begin{array}{lll} \tau_1(i) = \lfloor (11i+1)/2 \rfloor, & \tau_2(i) = \lfloor (11i+5)/2 \rfloor, & \tau_3(i) = \lfloor (11i+9)/2 \rfloor, \\ \tau_4(i) = \lfloor (11i+13)/2 \rfloor, & \tau_5(i) = \lfloor (11i+7)/2 \rfloor, & \tau_6(i) = \lfloor 11i/2 \rfloor, \\ \tau_7(i) = \lfloor (11i+4)/2 \rfloor, & \tau_8(i) = \lfloor (11i+8)/2 \rfloor, & \tau_9(i) = \lfloor (11i+17)/2 \rfloor, \\ \tau_{10}(i) = \lfloor (11i+21)/2 \rfloor. & & \end{array}$$

If we consider D as a variable where $D \in \{2, 3\}$, a solution time of several minutes is required. Although the number of variables and constraints increases only slightly, the condition of the linear program deteriorates drastically.

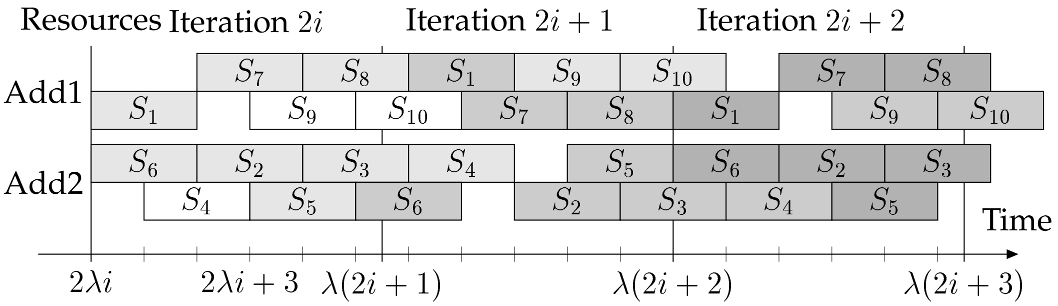

Figure 6: Scheduling of Example 3

7 Discussion and Extensions

In this section, we discuss some aspects of our approach, and outline some extensions, thus rounding off the presentation of our paper.
As a first extension we want to discuss the assignment of operations to *functional units of different types*. Suppose that several functional units $u_l \in \mathcal{U}_j$, where $\mathcal{U}_j \subseteq \mathcal{U}$, are able to evaluate operation F_j. We express the assignment of a functional unit u_l to operation F_j by binary variables $\phi_{l,j} \in \{0,1\}$, where $\sum\limits_{u_l \in \mathcal{U}_j} \phi_{l,j} = 1$ and $\phi_{l,j} = 1 \rightarrow f(j) = l$.
In the case of a fixed D the term $D\, L_{f(\sigma(e)),\sigma(e)}$ in (23) has to be replaced by $\sum\limits_{u_l \in \mathcal{U}_{\sigma(e)}} \phi_{l,\sigma(e)} L_{l,\sigma(e)}$, and the term $D\, O_{l,j}$ in (9), (18) and (20) by $\sum\limits_{u_l \in \mathcal{U}_j} \phi_{l,j} O_{l,j}$. The bounds $D\, O_{l,j}$ of the summation symbols in (21) and (20) have to be replaced by $\max\limits_{u_l \in \mathcal{U}_j} D\, O_{l,j}$.
The treatment of D *as a free parameter* requires more effort, since the products $D\, O_{l,j}$ in (9) and $D\, L_{f(\sigma(e)),\sigma(e)}$ in (23) result in non-linear constraints. These constraints can be linearized in the following way:

$$\left.\begin{array}{rcl} b_j - b_k & \geq & i \sum\limits_{u_l \in \mathcal{U}_k} O_{l,k} - (1-\delta_i) D^{max} \max\limits_{u_l \in \mathcal{U}_k} O_{l,k} - \alpha_{jk} C, \\ A + b_k - b_j & \geq & i \sum\limits_{u_l \in \mathcal{U}_j} O_{l,j} - (1-\delta_i) D^{max} \max\limits_{u_l \in \mathcal{U}_j} O_{l,j}, \\ A + b_j - b_k & \geq & i \sum\limits_{u_l \in \mathcal{U}_k} O_{l,k} - (1-\delta_i) D^{max} \max\limits_{u_l \in \mathcal{U}_k} O_{l,k}, \\ b_k - b_j & \geq & i \sum\limits_{u_l \in \mathcal{U}_j} O_{l,j} - (1-\delta_i) D^{max} \max\limits_{u_l \in \mathcal{U}_j} O_{l,j} - (1-\alpha_{jk}) C, \end{array}\right. \quad 1 \leq i \leq D^{max}. \tag{9*}$$

$$\forall e \in \mathcal{E}. A d(e) + B_{\delta(e)} - B_{\sigma(e)} \geq i \sum_{u_l \in \mathcal{U}_{\sigma(e)}} L_{f(\sigma(e)),\sigma(e)} - (1-\delta_i) D^{max} \max_{u_l \in \mathcal{U}_{\sigma(e)}} L_{f(\sigma(e)),\sigma(e)}, \quad 1 \leq i \leq D^{max}. \tag{23*}$$

For most problems (that we encountered), only small values of D are reasonable, e. g. $D \in \{2,3\}$. A drawback of a large D is that the code size is growing linearly in D. Moreover the relative improvement of $\lambda_{D^{max}}$, which is the optimal value of λ for $D \leq D^{max}$, is less than $1/(D^{max}\lambda_{D^{max}})$. For practical reasons we recommend to keep D fixed, since the linear problem is significantly more complex with a free D than for a fixed value of D.
Next, we discuss the treatment of functional units $u_l \in \mathcal{U}$ that have an *offset time* $O_{l,j}$ *exceeding the initiation interval* λ, i.e. $D\, O_{l,j} > A$. Then for operation j a number of $\underline{O}_{l,j} = \lfloor D\, O_{l,j}/A \rfloor$ units of type u_l is used permanently. The resource constraint has to be applied to $o_{l,j} \equiv D\, O_{l,j} \pmod{A}$. This can be formulated introducing binary variables $\mu_{j,k} \in \{0,1\}$, similarly to the decomposition of B_j in (10):

$$\begin{array}{rl} -(1-\mu_{j,k})C_{12} & \leq D\, O_{l,j} - kA - o_{l,j} \leq \ (1-\mu_{j,k})C_{13}, \qquad 0 \leq k \leq \lfloor D^{min} O_{l,j}/A^{min} \rfloor, \\ 0 & \leq o_{l,j} \leq A - 1, \\ \sum\limits_{k=0}^{\lfloor D^{min} O_{l,j}/A^{min} \rfloor} & \mu_{j,k} = 1, \end{array} \tag{30}$$

where $C_{12} \geq (k+1)A^{max} - 1 - D^{min} O_{l,j}$ and $C_{13} \geq D^{max} O_{l,j}$.
Equations (17) and (18) have to be replaced by:

$$\forall t' \in \mathbb{Z}.\ \forall u_l \in \mathcal{U}.\ \sum_{\substack{j=1 \\ f(j)=l}}^{M} (\gamma_{j,t'} + \sum_{k=0}^{\lfloor D^{min} O_{l,j}/A^{min} \rfloor} k \mu_{j,k}) \leq U_l. \quad {\scriptstyle 0 \leq t' \leq A-1} \tag{17*}$$

$$\begin{aligned} b_j &\le t' - D\,o_{l,j} - 1 + (\gamma_{j,t'} + \mu_{j,t'})C_1, \\ b_j &\ge t' - (\gamma_{j,t'} + (1-\mu_{j,t'}))C_2, \\ b_j &\le A + t' - D\,o_{l,j} - 1 + (\gamma_{j,t'} + (1-\mu_{j,t'}))C_3. \end{aligned} \tag{18*}$$

Another yet open problem is that of *mapping instructions to particular functional units*. In the case of overlapping instructions needing the same unit type, it is not always possible to map each instruction to the same unit in each iteration. Because of the different mapping, different code has to be generated for successive iterations, which is similar to partial loop unrolling. This increases the code size, and may cause cache misses of the instruction cache. To cope with this problem, we would have to regard the code size, the cache size, the costs (delays) of cache misses, etc., and find a trade-off between the costs of and the benefits from larger code. However, the cache sizes of modern processors are large enough to keep larger loops, such that - even with very large loops - the delays due to cache misses are very small compared to the loop delay. These problems are discussed in detail in [4], and are quite similar to the problem of mapping variables to registers, as discussed in [8, 9, 10, 22, 23, 34, 35].
We finish this section by proposing a *fixed assignment* for the case of multiple functional units, which circumvents the above-mentioned problems. Up to now, an operation is allowed to switch functional units periodically. If we suppress this behaviour, which simplifies the resulting program code, we can drastically simplify the description of the resource constraints. Suppose that operation F_j requires functional unit u_l, i.e. $f(j) = l$. Then we introduce binary variables $\varphi_{j,i} \in \{0,1\}$ such that $\varphi_{j,i} = 1$ if F_j is evaluated on the i-th instance of u_l. The uniqueness of selection is ensured by:

$$\sum_{i=1}^{N_{u(j)}} \varphi_{j,i} = 1.$$

Even in the case of multiple functional units, we can apply (9) by ensuring its validity only in the case of $\varphi_{j,i} = \varphi_{k,i} = 1$:

$$\left.\begin{array}{rcl} b_j - b_k & \ge & D\,O_{l,k} - (\alpha_{jk} + 2 - \varphi_{j,i} - \varphi_{k,i})C, \\ A + b_k - b_j & \ge & D\,O_{l,j} - (2 - \varphi_{j,i} - \varphi_{k,i})D\,O_{l,j}, \\ A + b_j - b_k & \ge & D\,O_{l,k} - (2 - \varphi_{j,i} - \varphi_{k,i})D\,O_{l,k}, \\ b_k - b_j & \ge & D\,O_{l,j} - (1 - \alpha_{jk} + 2 - \varphi_{j,i} - \varphi_{k,i})C, \end{array}\right. \quad 1 \le i \le N_{u(j)}. \tag{9**}$$

Application of this approximation to Example 3 leads to the optimum $\lambda = 11/2$, requiring a solution time of only 38 seconds for $D = 6$, $A^{max} = 120$ and $p^{max} = 2$.

8 Conclusions

In this paper we developed an integer linear program (ILP) formulation to solve the software pipelining problem under resource constraints. We constructed resource constraints for single functional units (i.e. operations require the same resource), and for multiple functional units (several functional units are able to perform an operation), following the classification introduced by Feautrier [12]. The causality constraint we deduced from flow dependencies of the program sequence in the usual manner. In our ILP approach a rational $\lambda = A/D$ is assumed, where A and D are free parameters of the linear program. For larger problems it is recommended to keep D fixed, either using the least common multiple of the preferred values of D in a single problem, or running several optimizations with different values for D. We illustrated the validity of our approach by results from several examples.

References

[1] A. Aiken, A. Nicolau, and S. Novack. Resource-constrained software pipelining. *IEEE Transactions on Parallel and Distributed Systems*, 6(12), December 1995.

[2] V.H. Allan, R.B. Jones, R.M. Lee, and S.J. Allan. Software pipelining. *ACM Computing Surveys*, 27(3):367–432, September 1995.

[3] E.R. Altman. *Optimal Software Pipelining with Function Unit and Register Constraints*. PhD thesis, McGill University, Montreal, Quebec, Canada, October 1995.

[4] E.R. Altman, R. Govindarajan, and G.R. Gao. Scheduling and mapping: software pipelining in the presence of structural hazards. *ACM SIGPLAN Notices: Conference on Programming Language Design and Implementation PLDI*, 30(6):139–150, June 1995.

[5] A. Bachmann, M. Schöbinger, and L. Thiele. Synthesis of domain specific multiprocessor systems including memory design. In *Int. Workshop on VLSI and Signal Processing*, pages 417–425, 1993.

[6] P.-Y. Calland, A. Darte, and Y. Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Transactions on Parallel and Distributed Systems*, 9(1):24 – 35, 1998.

[7] A.E. Eichenberger and E.S. Davidson. Efficient formulation for optimal modulo schedulers. In *ACM SIGPLAN'97 Conference on Programming Language Design and Implementation PLDI*, pages 194–205, Las Vegas, Nevada, June 1997.

[8] A.E. Eichenberger, E.S. Davidson, and S.G. Abraham. Minimizing register requirements of a modulo schedule via optimum stage scheduling. *International Journal of Parallel Programming*, 24(2):103–132, April 1996.

[9] C. Eisenbeis, S. Lelait, and B. Marmol. The meeting graph: a new framework for loop register allocation. Technical Report RR-2758, INRIA, Institut National de Recherche en Informatique et en Automatique, 1995.

[10] C. Eisenbeis and A. Sawaya. Optimal loop parallelization under register constraints. Technical Report RR-2781, INRIA, Institut National de Recherche en Informatique et en Automatique, 1996.

[11] P. Feautrier. Dataflow analysis of scalar and array references. *International Journal of Parallel Programming*, 20(1):23–53, February 1991.

[12] P. Feautrier. Fine-grain scheduling under resource constraints. In *Proc. 7th Int. Workshop on Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Science, pages 1–15, Ithaca, New York, 1994. Springer-Verlag.

[13] A. Fettweis. Wave digital filters: Theory and practice. *Proc. of the IEEE*, 74:270 – 327, 1986.

[14] F. Gasperoni and U. Schwiegelshohn. Generating close to optimum loop schedules on parallel processors. *Parallel Processing Letters*, 4(4):391–403, 1994.

[15] C.H. Gebotys. *VLSI Design Methodologies for Digital Signal Processing Architectures*, chapter Synthesizing Optimal Application-Specific DSP Architectures, pages 43–92. Kluwer Academic Publishers, Boston/Dordrecht/London, M.A. Bayoumi edition, 1994.

[16] R. Govindarajan, E.R. Altman, and G.R. Gao. A framework for resource-constrained rate-optimal software pipelining. In *CONPAR/VAPP*, Linz, Austria, 1994.

[17] R. Govindarajan, E.R. Altman, and G.R. Gao. Minimizing register requirements under resource-constrained rate-optimal software pipelining. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 85–94, San Jose, California, November 30–December 2, 1994. ACM SIGMICRO and IEEE Computer Society TC-MICRO.

[18] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu. A formal approach to the scheduling problem in high-level synthesis. *IEEE Transactions on Computer-Aided Design*, 10(4):464–474, 1991.

[19] M.S. Lam. Software pipelining; an effective scheduling technique for VLIW machines. In *SIGPLAN'88 Conference on Programming Languages, Design and Implementation PLDI*, pages 318–328, Atlanta, GA, USA, 1988.

[20] D.M. Lavery and W.W. Hwu. Unrolling-based optimizations for modulo scheduling. In *Proceedings of the 28th annual international symposium on Microarchitecture*, page 327, 1995.

[21] C.E. Leiserson and J.B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[22] S. Lelait, G.R. Gao, and C. Eisenbeis. A new fast algorithm for optimal register allocation in modulo scheduled loops. Research Report 3337, INRIA, Rocquencourt, January 1998.

[23] Q. Ning and G.R. Gao. A novel framework of register allocation for software pipelining. In *Proceedings of the 20th Annual Symposium on Principles of Programming Languages POPL93*, pages 29–42, Charleston, SC, USA, 1993.

[24] P. Quinton. *Automata Networks in Computer Science*, chapter The Systematic Design of Systolic Arrays, pages 229–260. Manchester University Press, F. Fogelman and Y. Robert and M. Tschuente edition, 1987.

[25] B.A. Rau and J.A. Fisher. Instruction-level parallel processing: History, overview and perspective. *Journal of Supercomputing*, 7(1):9–50, 1993.

[26] B.R. Rau and C.D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In *Proc. of the 14th Annual Workshop of Microprogramming*, pages 183–198, 1981.

[27] A. Schrijver. *Theory of Integer and Linear Programming*. John Wiley & Sons, New York, 1986.

[28] M. Schwiegershausen, M. Schönfeld, and P. Pirsch. Mapping complex image processing algorithms onto heterogenous multiprocessors regarding architecture dependent performance parameters. In *Proc. Int. Workshop of Algorithms and Parallel VLSI Architectures III*, pages 353–364, 1994.

[29] V.H. van Dongen, R.G. Gao, and Q. Ning. A polynomial time method for optimal software pipelining. *Lecture Notes in Computer Science*, 634:613–624, 1992.

[30] S.R. Vegdahl. A dynamic-programming technique for compacting loops. In *Proceedings of the 25th Annual International Symposium on Microarchitecture, Portland, Oregon, USA*, pages 180–188, December 1-4 1992.

[31] C.-Y. Wang and K.K. Parhi. Resource-constrained loop list scheduler for DSP algorithms. *Journal of VLSI Signal Processing*, 11:75–96, 1995.

[32] J. Wang and C. Eisenbeis. Decomposed software pipelining. In *Proceedings of IFP WG 10.3 Working Conference on Architectures and Compilation Techniques*, pages 3–15, January 1993.

[33] J. Wang, C. Eisenbeis, M. Jourdan, and B. Su. Decomposed software pipelining: A new perspective and a new approach. *International Journal of Parallel Programming*, 22(3):357–379, 1994.

[34] J. Wang, A. Krall, and M.A. Ertl. Decomposed software pipelining with reduced register requirement. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques PACT'95*, pages 70–75. IFIP, June 1995.

[35] J. Wang, A. Krall, and M.A. Ertl. Register requirement for exploiting loops' maximum instruction-level parallelism. In *Proceedings of the 4rd International Conference for Young Computer Scientists*, pages 277–280, Beijing, China, July 1995.

Appendix

In this Appendix we document the proofs of Theorems 1, 2, 3, and 4. For convenience, we added the respective theorems and equations on top of the proofs.

$$\forall i, i'. \left\lfloor \frac{Ai + B_j}{D} \right\rfloor \geq \left\lfloor \frac{Ai' + B_k}{D} \right\rfloor + O_{l,k} \vee \left\lfloor \frac{Ai' + B_k}{D} \right\rfloor \geq \left\lfloor \frac{Ai + B_j}{D} \right\rfloor + O_{l,j}. \tag{3}$$

Theorem 1 *(3) is equivalent to:*

$$\forall i, i'. \frac{Ai + B_j}{D} \geq \frac{Ai' + B_k}{D} + O_{l,k} \vee \frac{Ai' + B_k}{D} \geq \frac{Ai + B_j}{D} + O_{l,j}. \tag{4}$$

Proof: We consider only the first inequality; the second inequality has to be treated in a similar way.
(4) $\rightarrow$ (3): Clearly, it is:

$$(Ai + B_j)/D \geq (Ai' + B_k)/D + O_{l,k} \geq \lfloor (Ai' + B_k)/D \rfloor + O_{l,k}.$$

Since the right hand side is integer, we can write:

$$\lfloor (Ai + B_j)/D \rfloor \geq \lfloor (Ai' + B_k)/D \rfloor + O_{l,k}.$$

(3) $\rightarrow$ (4): We prove this direction by showing that the contrary of (4) leads to a contradiction to (3):

$$\exists i, i'.(Ai + B_j)/D < (Ai' + B_k)/D + O_{l,k} \rightarrow \exists j, j'.\lfloor (Aj + B_j)/D \rfloor < \lfloor (Aj' + B_k)/D \rfloor + O_{l,k}.$$

We choose j' such that $(Aj' + B_k)/D$ is integer, which is always possible since A and D are relatively prime. Then we have:

$$(A(i + j' - i') + B_j)/D < (Aj' + B_k)/D + O_{l,k} = \lfloor (Aj' + B_k)/D \rfloor + O_{l,k}.$$

Again, the right hand side is integer, and we can write:

$$\lfloor (A(i + j' - i') + B_j)/D \rfloor < \lfloor (Aj' + B_k)/D \rfloor + O_{l,k}.$$

We finish the proof by setting $j = i + j' - i'$, which produces the contradiction to (3).
■

$$\forall t \in \mathbb{Z}.\forall u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t - O_{l,j} + 1 \leq (Ai + B_j)/D < t + 1\}| \leq U_l. \tag{12}$$

Theorem 2 *(12) is equivalent to:*

$$\underset{0 \leq t' \leq A-1}{\forall t' \in \mathbb{Z}.} \ \forall u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t' - D\, O_{l,j} \leq Ai + B_j < t'\}| \leq U_l. \tag{13}$$

Proof: (12) $\rightarrow$ (13): Suppose that:

$$\underset{0 \leq t' \leq A-1}{\exists t' \in \mathbb{Z}.} \ \exists u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t' - D\, O_{l,j} \leq Ai + B_j < t'\}| > U_l.$$

Then we have:

$$\exists t \in \mathbb{Z}.|\{(S_j, i')|f(j) = l \wedge t - O_{l,j} + 1 \leq (Ai' + B_j)/D < t + 1\}| > U_l,$$

where $Dt + D = kA + t'$, $i' = i + k$ and $k \in \mathbb{Z}$. Pairs (t, k) always exist since A and D are relatively prime.
(13) $\rightarrow$ (12): Again we show that the contrary of (12) leads to a contradiction to (13):

$$\exists t \in \mathbb{Z}.\exists u_l \in \mathcal{U}.|\{(S_j, i)|f(j) = l \wedge t - O_{l,j} + 1 \leq (Ai + B_j)/D < t + 1\}| > U_l.$$

Then:

$$\exists t' \in \mathbb{Z}.|\{(S_j, i')|f(j) = l \wedge t' - D\, O_{l,j} \leq Ai' + B_j < t'\}| > U_l,$$

where $Dt + D = \lfloor (Dt + D)/A \rfloor A + t'$ and $i' = i - \lfloor (Dt + D)/A \rfloor$.
■

$$\forall e \in \mathcal{E}. \underset{1 \leq i \leq N}{\forall i \in \mathbb{Z}} . \left\lfloor \frac{Ai + Ad(e) + B_{\delta(e)}}{D} \right\rfloor \geq \left\lfloor \frac{Ai + B_{\sigma(e)}}{D} \right\rfloor + L_{f(\sigma(e)),\sigma(e)}. \tag{22}$$

Theorem 3 *For a sufficiently large N, (22) is equivalent to:*

$$\forall e \in \mathcal{E}. Ad(e) + B_{\delta(e)} - B_{\sigma(e)} \geq D\, L_{f(\sigma(e)),\sigma(e)}. \tag{23}$$

Proof: (23) $\rightarrow$ (22): Adding Ai to both sides of (23) leads to:

$$\forall e \in \mathcal{E}.\ \forall \underset{1\leq i\leq N}{\mathrm{i} \in \mathbb{Z}}\ .\frac{Ai + Ad(e) + B_{\delta(e)}}{D} \geq \frac{Ai + B_{\sigma(e)}}{D} + L_{f(\sigma(e)),\sigma(e)} \geq \left\lfloor \frac{Ai + B_{\sigma(e)}}{D} \right\rfloor + L_{f(\sigma(e)),\sigma(e)}.$$

Since the right hand side is integer, (22) follows.
(22) $\rightarrow$ (23): We consider the contradictions to both expressions:

$$Ad(e) + B_{\delta(e)} - B_{\sigma(e)} < D\, L_{f(\sigma(e)),\sigma(e)} \rightarrow \exists i \in \mathbb{Z}. \left\lfloor \frac{Ai + Ad(e) + B_{\delta(e)}}{D} \right\rfloor < \left\lfloor \frac{Ai + B_{\sigma(e)}}{D} \right\rfloor + L_{f(\sigma(e)),\sigma(e)}.$$

At first we get:

$$\forall i \in \mathbb{Z}. \frac{Ai + Ad(e) + B_{\delta(e)}}{D} < \frac{Ai + B_{\sigma(e)}}{D} + L_{f(\sigma(e)),\sigma(e)}.$$

Then we choose $i \in \mathbb{Z}$ such that $(Ai + B_{\sigma(e)})/D$ is integer, which can always be found since A and D are relatively prime. This leads to:

$$\frac{Ai + Ad(e) + B_{\delta(e)}}{D} < \frac{Ai + B_{\sigma(e)}}{D} + L_{f(\sigma(e)),\sigma(e)} = \left\lfloor \frac{Ai + B_{\sigma(e)}}{D} \right\rfloor + L_{f(\sigma(e)),\sigma(e)},$$

and hence:

$$\left\lfloor \frac{Ai + Ad(e) + B_{\delta(e)}}{D} \right\rfloor \leq \frac{Ai + Ad(e) + B_{\delta(e)}}{D} < \left\lfloor \frac{Ai + B_{\sigma(e)}}{D} \right\rfloor + L_{f(\sigma(e)),\sigma(e)}.$$

■

Theorem 4 *Suppose that $H = gcd\{A, D\}$. Then we have:*

$$\forall i \in \mathbb{Z}. \left\lfloor \frac{Ai + B_j}{D} \right\rfloor = \left\lfloor \frac{A'i + B'_j}{D'} \right\rfloor, \tag{29}$$

where $A' = A/H$, $D' = D/H$ and $B'_j = \lfloor B_j/H \rfloor$.

Proof: We have:

$$\forall i \in \mathbb{Z}. \lfloor (Ai + B_j)/D \rfloor = \lfloor (A'i + B_j/H)/D' \rfloor = \lfloor (A'i + \lfloor B_j/H \rfloor + h)/D' \rfloor,$$

where $h \in [0, (H-1)/H]$, i.e. $h \leq (H-1)/H$.
Suppose that the contrary applies:

$$\exists i \in \mathbb{Z}. \lfloor (A'i + \lfloor B_j/H \rfloor + h)/D' \rfloor > \lfloor (A'i + \lfloor B_j/H \rfloor)/D' \rfloor.$$

Then we have:

$$\begin{array}{rcl} \lfloor (A'i + \lfloor B_j/H \rfloor + h)/D' \rfloor & \geq & \lfloor (A'i + \lfloor B_j/H \rfloor)/D' \rfloor + 1 \\ (A'i + \lfloor B_j/H \rfloor + h)/D' & \geq & \lfloor (A'i + \lfloor B_j/H \rfloor)/D' \rfloor + 1 \\ (A'i + \lfloor B_j/H \rfloor)/D' - \lfloor (A'i + \lfloor B_j/H \rfloor)/D' \rfloor & \geq & 1 - h/D' \end{array}.$$

With $(D'-1)/D' \geq (A'i + \lfloor B_j/H \rfloor)/D' - \lfloor (A'i + \lfloor B_j/H \rfloor)/D' \rfloor$ follows $(D'-1)/D' \geq 1 - h/D'$, and hence $h \geq 1$ which is a contradiction to $h \leq (H-1)/H$.
■