Lower Bounds on the Iteration Time and the Initiation Interval of Functional Pipelining and Loop Folding

Abstract—**The performance of pipelined datapath implementations is measured basically by three parameters: the clock cycle length, the initiation interval between successive iterations (inverse of the throughput) and the iteration time (turn-around time). In this paper we present a new method for computing performance bounds of pipelined implementations:**

- **Given an iterative behavior, a set of resource constraints and a target initiation interval, we derive a lower bound on the iteration time achievable by any pipelined implementation.**
- **Given an iterative behavior and a set of resource constraints, we derive a lower bound on the initiation interval achievable by any pipelined implementation.**

The method has a low complexity and it handles behavioral specifications containing loop statements with inter-iteration data dependency and timing constrains.

1. Introduction

High level synthesis refers to the design process which transforms a behavioral specification of a digital system into a register transfer level (RTL) structure. Two fundamental steps in high level synthesis are scheduling and allocation. Scheduling assigns circuit operations to control steps under resource constraints (e.g. functional units, registers and buses) and/or performance constraint, while allocation assigns operations and data transfers to resources to realize the datapath. Synthesis of efficient circuits for real-time digital signal processing (DSP) applications is becoming a more challenging and crucial task, because most applications require higher sample rates and higher sample processing speed. These applications are often recursive or iterative and their behavioral descriptions consist of an infinite loop statement. In order to synthesize a high throughput circuit, a scheduler should exploit all the potential concurrence between the loop body operations. A way to exploit this parallelism is to *pipeline* (overlap) the execution of successive iterations. This technique is called *loop folding* in the general case, and it is called *functional*

pipelining (functional pipelined datapath) when there are no data dependencies between different iterations of the algorithm. Figure 1 illustrates the pipelining approach where a loop instance considered as a task is split into five subtasks ST_i. Each subtask ST_i corresponds to a set of operations executed in parallel.

The performance of a pipelined datapath is measured basically by three values [1]: the clock cycle length, the initiation interval and the iteration time. The *initiation interval* corresponds to the number of clock cycles separating the initiation of successive instances of the loop, it is the inverse of the throughput. The *iteration time* corresponds to the number of clock cycles necessary to execute one instance inside the pipeline (turn-around time), it measures the sample processing speed which is usually critical.

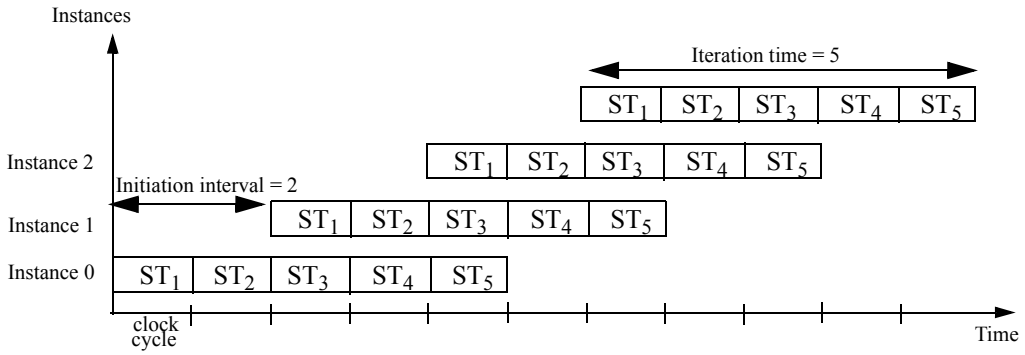


Figure 1. Space-time diagram of the pipelined schedule

The problem of determining the optimum initiation interval and the problem of determining the optimum iteration time for a given initiation interval are both NP-complete [8, 38] when there are resource constraints and inter-iteration data dependencies. The main objective of pipelined scheduling heuristics developed in the literature [1-11] is to find schedules with low initiation interval and low iteration time.

In this paper we address the two following lower-bound problems:

Problem 1: Given a cyclic data flow graph representing a loop statement, a set of resource constraints and a target initiation interval, we derive a lower bound on the iteration time achievable by any pipelined implementation.

Problem 2: Given a cyclic data flow graph and a set of resource constraints, we derive a lower bound on the initiation interval achievable by any pipelined implementation.

2. Motivations and previous works

The motivations for developing performance estimators in high-level synthesis area have been discussed extensively in [14,15,17,22]. Here we present three main motivations related to pipelined designs:

• Speed-up the space solution exploration. To achieve high performance pipelined designs, behavioral optimizations and dataflow-based transformations such as common subexpression elimination, associativity-commutativity algebraic transformations [28, 32-35], loop unfolding [29- 31] and retiming [13], are often necessary. To find the best set of transformations and the best order requires the analysis of a large number of solutions. An efficient method to compute exact lower-bound performance allows to speed-up this exploration by detecting and removing solutions which theoretically cannot achieve the target performance under a given resource constraints.

• Evaluation of the quality of a pipelined solution produced by a heuristic. By comparing the exact lower-bound performance to the performance of the heuristic solution, we get the maximal distance between the heuristic solution and the optimum solution. A small distance indicates the good quality of the heuristic solution.

• Performance improvement of scheduling heuristics. The general framework used by resource-constraint pipelined scheduling heuristics [1-10] is composed of the following steps:

1. Fix the initiation interval to its tight lower bound;
2. For the target initiation interval, fix the iteration time to its tight lower bound;
3. Find a pipelined schedule with the current initiation interval and iteration time;
4. If no feasible schedule found and no time out, then increment the iteration time, and go to Step 3. Otherwise, increment the initiation interval, and go to Step 2.

Step3 is the most time consuming step since it is repeated many times and it is NP-complete. The use of tight lower bounds in steps 1 and 2 reduces considerably this time.

The majority of previous studies related to lower-bound performance estimation are restricted to non-pipelined designs [14-20], and only few works have dealt with pipelined designs [22, 23, 28]. Jain *et al.* [22] have addressed the problem of area-delay prediction in pipelined and non-pipelined designs. For pipelined designs, they have presented a simple algorithm to compute a lower bound on the product of functional unit requirements with the initiation interval for acyclic

data flow graphs. Although a lower bound on the product of these two parameters (number of functional units and initiation interval) permits to make design space exploration, this exploration is still restricted compared to the case where a lower bound on each parameter is given separately. Hu *et al.* [23] have presented a method to compute lower bounds on the iteration time under resource constraints but it is also restricted to acyclic data flow graphs. In [28], Potkonjak and Rabaey studied the relation between retiming and functional pipelining, and they gave a new polynomial algorithm for determining an upper bound on the throughput. However, the algorithm does not take into account the resource constraints. Other polynomial algorithms have been proposed [24-27] for determining an upper bound on the throughput and computing optimum rate schedules but they assume an unlimited number of resources.

Comparatively to the above mentioned works, the method presented here differs in both the scope and the techniques used. It computes exact lower bounds on the iteration time and on the initiation interval for an iterative behavior under resource constraints. The iterative behavior may contain inter-iteration and intra-iteration precedence constraints, and user-specified min-max timing constraints between pairs of operations. It may also contain bounded nested loops; in such case all inner loops will be totally unfolded. The used technique is based on integer programming constraint relaxations; the proof that this technique has a minimal complexity is established. The method solves also the dual problem: lower bound on the resource requirement to achieve a target iteration time and/or a target initiation interval. In this paper we restrict resource type to functional units. Registers and buses can be handled similarly by adding to the data flow graph a set of fictive operations representing data memorizations and data transfers.

The rest of this paper is organized in the following way. Section 3 gives the necessary background. Section 4 describes the constraint graph model. Section 5 gives the general formulation for the minimum iteration time problem, then a relaxation of this problem, and the proof that the relaxed problem can be solved optimally in polynomial time by the proposed method. Lower-bound algorithms for the iteration time and for the initiation interval are given in Section 6 and Section 7, respectively. Experimental results are presented in Section 8.

3. Background

C*yclic data flow graph* (DFG) is a well known model used to capture loop behavior. A cyclic DFG $G = (O, E, w, d)$ is a node-weighted and edge-weighted directed graph, where $O = \{o_i\}$ is the set of nodes representing atomic operations, E is the set of edges. An edge $(o_i \rightarrow o_j)$ corresponds to data precedence relationship between operations o_i and o_j. The integer edge weight $w(o_i \rightarrow o_j)$ means that the data produced by operation o_i in any iteration k of the loop will be used by operation o_j in iteration $(k + w(o_i \rightarrow o_j))$. The value $d(o_i)$ is the duration of operation o_i in clock cycles. Figure 2(a) shows a cyclic DFG example, where normal arcs correspond to intra-iteration precedence constraints and they have the implicit weight zero, and dashed arcs correspond to inter-iteration precedence constraints. This example will be used throughout the paper.

Definition: Let $s(o_j, k)$ be the starting execution time of operation o_j of the *k*-th iteration of the loop. The schedule s is said to be a *pipelined schedule* with an initiation interval II if it satisfies:

$$s(o_j, k) = s(o_j, 0) + k \cdot II, \qquad \forall o_j \in O, \forall k \in N \tag{1}$$

and $$s(o_j, w(o_i \rightarrow o_j)) \geq s(o_i, 0) + d(o_i), \qquad \forall (o_i \rightarrow o_j) \in E \tag{2}$$

Equation (1) expresses the schedule periodicity, and Equation (2) expresses intra-iteration and inter-iteration precedence constraints between operations. The iteration time (IT) of the schedule s is defined as:

$$IT = \underset{i}{Max} \{s(o_i, 0) + d(o_i)\} + 1 \tag{3}$$

As all iterations have the same schedule, they have the same iteration time. Figure 3 shows a pipelined schedule of the cyclic DFG of Figure 2(a).

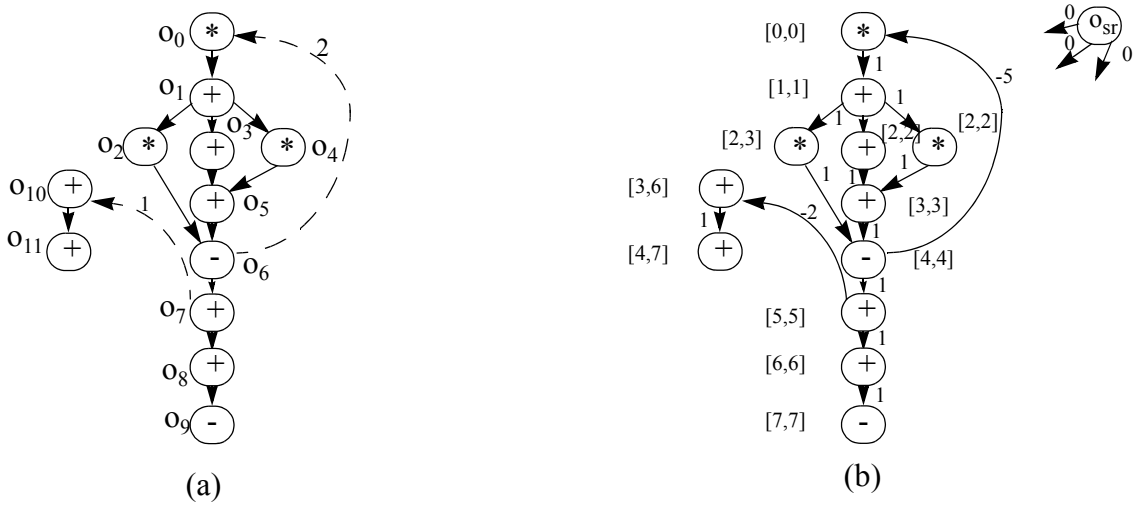


Figure 2. (a) A cyclic DFG example. (b) The constraint graph corresponding for $II = 3$, and the operation mobility-intervals.

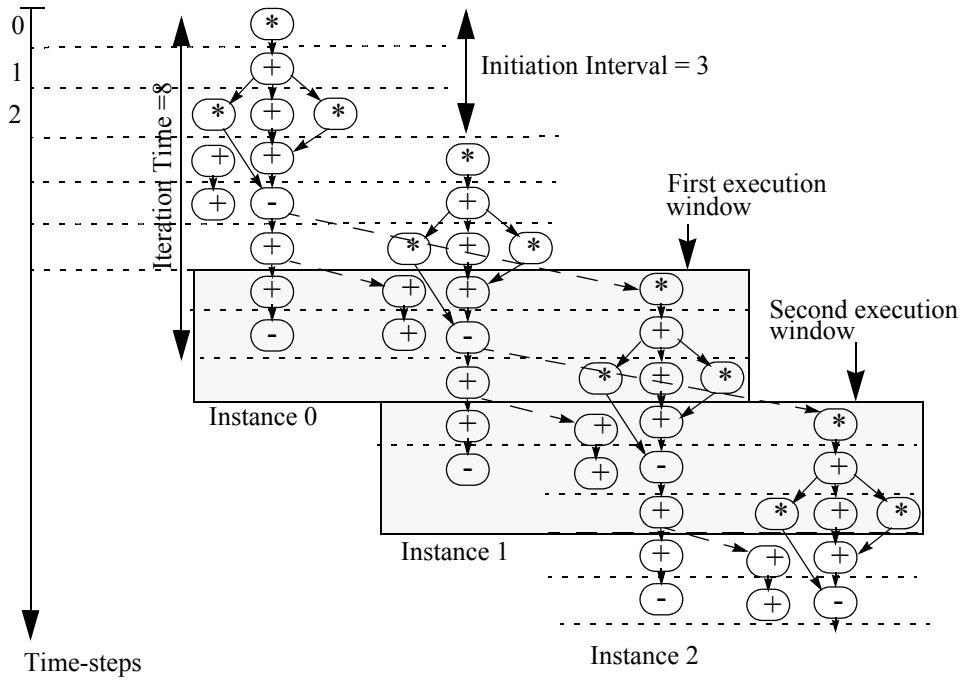


Figure 3. A pipelined schedule of the cyclic DFG of Figure 2(a).

Initial lower bound on the initiation interval: The presence of loops in a cyclic DFG, called *loop carried dependencies* (LCD), limits the minimal value of the initiation interval. The lower bound value due to LCD, denoted II_{LDC}, is defined as follows [21]:

$$II_{LDC} = \underset{c_i}{Max} \left\{ \left\lceil \frac{D_{c_i}}{L_{c_i}} \right\rceil \right\} \tag{4}$$

where c_i is a loop in the DFG, D_{c_i} and L_{c_i} are respectively the sum of operation durations and the sum of arc weights in c_i.

Resource constraints limit also the minimal value of the initiation interval. This minimal value due to resource constraints, denoted II_{RC}, is given by [1]:

$$II_{RC} = \underset{r}{Max} \left\{ \left\lceil \frac{|T_r| \cdot \alpha_r}{M_r} \right\rceil \right\} \tag{5}$$

where r is a functional unit type, M_r is the available number of functional units of type r, T_r is the set of operations in the cyclic DFG executed by functional units of type r, and α_r is the duration in clock cycles of type r functional unit.

Theorem 1*: The initiation interval II of a feasible pipelined schedule must satisfy:*

$$II \geq Max\{II_{RC}, II_{LDC}\}$$

4. The Constraint Graph

Both intra-iteration and inter-iteration precedence constraints in a cyclic DFG will be transformed into timing constraints between operations of the *first loop iteration*. Combining Equations (1) and (2), we obtain:

$$s(o_j, 0) \geq s(o_i, 0) + d(o_i) - w(o_i \rightarrow o_j) \cdot II \quad , \qquad \forall (o_i \rightarrow o_j) \in E \tag{6}$$

Equation (6) expresses timing constraint between operations of the same initial loop iteration.

Definition: Let be $G = (O, E, w, d)$ a cyclic DFG. The *constraint graph CG* associated to *G* is an edge-weighted and directed graph defined by $CG = (O \cup \{o_{sr}\}, E \cup E_{sr}, \hat{w})$. Where O and E are the same sets of nodes and edges as in *G,* and o_{sr} is a source node connected to all nodes in O by the set of edges E_{sr}. The weights of edges $\hat{w}$ are defined as follows:

$$\hat{w}(o_i \rightarrow o_j) = d(o_i) - w(o_i \rightarrow o_j) \cdot II \quad , \qquad \forall (o_i \rightarrow o_j) \in E$$

$$\hat{w}(o_{sr} \rightarrow o_i) = 0 , \qquad \forall (o_{sr} \rightarrow o_i) \in E_{sr}$$

A positive (resp. negative) value of $\hat{w}(o_i \rightarrow o_j)$ means that operation o_j of the first iteration must be scheduled no sooner than $|\hat{w}(o_i \rightarrow o_j)|$ time-steps after (resp. before) operation o_i of the first iteration. Figure 2(b) shows the constraint graph corresponding to the cyclic DFG of Figure 2(a) for an initiation interval equal to 3. In the rest of the paper, we will use the constraint graph model instead of the DFG, and we assume that all multi-cyclic operations are broken into multiple uni-cycle operations related by timing constraints.

We denote by τ_i^s the ¨as soon as possible¨ starting time of operation o_i in the first iteration. The value of τ_i^s corresponds to the longest-path weight in the constraint graph from the source node o_{sr} to node o_i. The minimal number of time steps necessary to execute any iteration inside the pipeline corresponds to the critical path length in the constraint graph, denoted CP, and it is equal to:

$$CP = Max_i\{\tau_i^s + d(o_i)\} \qquad (7)$$

We denote by τ_i^l the ¨as late as possible¨ starting time of operation o_i of the first loop instance such that the schedule length does not exceed CP. The interval $[\tau_i^s, \tau_i^l]$ is called *mobility-interval* of o_i.

5. ILP formulation of the minimum iteration time problem and its relaxation

In this section, we give first an ILP formulation for the *minimum iteration time problem*: find the minimum value of the iteration time, given a constraint graph, a fixed number of functional units and a target initiation interval. Then we construct a relaxation of this problem and showing that the relaxed problem can be solved optimally in polynomial time by a greedy algorithm.

Notations:

- $GC = (O \cup \{o_{sr}\}, E \cup E_{sr}, \hat{w})$ is a constraint graph,
- τ_i^s is the earliest starting time of operation o_i of the first iteration, without resource constraints.
- τ_i^l is the latest starting time of operation o_i of the first iteration, without resource constraints.

- $M = \{1, 2, \ldots, m\}$ is the set of types of functional units,
- M_r is the number of functional units of type r,
- $T_r \subset O$ is the set of operations to be executed by functional units of type r,
- CP is the critical path length in the constraint graph,
- II is the initiation interval,
- IT is the iteration time,
- z is the additional number of time steps necessary to execute any loop iteration inside the pipeline. $z = IT - CP$.

5.1. ILP formulation of the minimum iteration time problem (P)

The system P formulates this NP-complete problem. A similar formulation is given in [12].

$$
P\left\{\begin{array}{ll}
\text{Minimize } z \text{ subject to:} & (8)\\
\sum_{t=0}^{IT-1} x_{i,t} = 1 & \forall o_i \in O \quad (9)\\
\sum_{k=0}^{\lfloor (IT-e-1)/II \rfloor} \sum_{o_i \in T_r} x_{i,e+k\cdot II} \le M_r & \forall e \in \{0, 1, \ldots, II-1\}, \forall r \in M \quad (10)\\
\tau_j - \tau_i \ge \hat{w}(o_i \rightarrow o_j) & \forall (o_i \rightarrow o_j) \in E \quad (11)\\
\tau_i < CP + z & \forall o_i \in O \quad (12)\\
\text{where } \tau_i = \sum_{t=0}^{IT-1} t\hat{x}_{i,t} &
\end{array}\right.
$$

$x_{i,t}$ is a binary variable, equals 1 if operation o_i of the first iteration is scheduled in control step t, zero otherwise. The objective function (8) states that we minimize the iteration time $IT = CP + z$ of the schedule. Constraint (9) states that each operation o_i should be scheduled. Constraint (10) expresses the resource constraints: due to the pipelining, operations in control steps $e + k \cdot II$, for $k = 0, 1, 2, \ldots$, are executed simultaneously and cannot share the same functional units (see Figure 4(a)). Constraint (11) expresses the timing constraints between operations. Constraint (12) states that all operations must be scheduled before control-step $(CP + z)$.

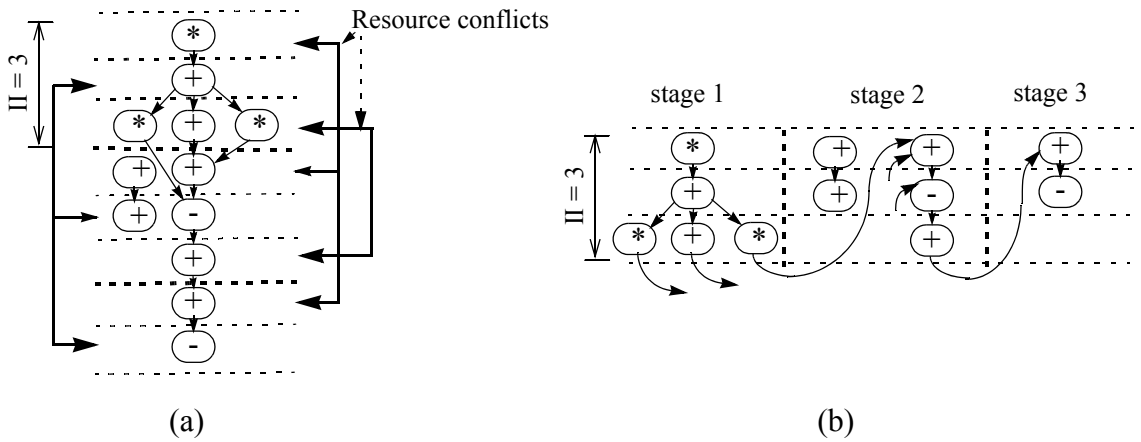


Figure 4. Two equivalent views of a pipelined schedule: (a) single-iteration view, (b) execution-window view.

5.2. The minimum iteration time relaxed problem

5.2.1 The relaxation problem ($\hat{PR}$)

The relaxation consists of replacing constraint (11) by a less strong constraint. The constraint is formalized by Equation (13). It defines the control-steps where operations must be scheduled: due to timing constraints between operations, an operation o_i cannot start before τ_i^s and must start no later than $\tau_i^l + z$. The relaxation problem is:

$$\hat{PR}\begin{cases} \text{Minimize } z \text{ subject to:} & \\ \sum_{t=0}^{IT-1} x_{i,t} = 1 & \forall o_i \in O \\ \sum_{k=0}^{\lfloor (IT-e-1)/II \rfloor} \sum_{o_i \in T_r} x_{i,e+k\cdot II} \le M_r & \forall e \in \{0, 1, \ldots, II-1\}, \forall r \in M \\ \tau_i^s \le \tau_i \le \tau_i^l + z & \forall o_i \in O \\ \tau_i < CP + z & \forall o_i \in O \\ \text{where } \tau_i = \sum_{t=0}^{IT-1} t x_{i,t} & \end{cases} \quad (13)$$

5.2.2 Formulation of the relaxation problem based on execution-window's properties (*PR*)

The *execution-window* of a pipelined schedule is the repetitive pattern of the same size as the initiation interval that appears after a certain number of scheduling steps. In this section we give a second formulation of the relaxation problem $\hat{PR}$ based on the execution-window' properties.

The execution window starting at control step number $((\lceil IT/II \rceil - 1) \cdot II)$ is called the *first execution-window* (see Figure 3). A feasible pipelined schedule should satisfy the three following necessary conditions:

C1. Each operation o_i occurs one and only one time inside the first execution window.

C2. If an operation o_i of the first iteration is scheduled into control-step τ_i, then there is one occurrence of o_i scheduled into control-step $(\tau_i \ mod \ II)$ relatively to the beginning of the first execution window (see Figure 4). More generally, if o_i of the first iteration is scheduled inside the control-step interval $[\tau_i^s, \tau_i^l + z]$, for any positive integer *z*, then there is one occurrence of o_i scheduled, relatively to the beginning of the first window, inside the domain $D_i(z)$ defined by:

$$D_i(z) = \begin{cases} [0, II-1], & if \ (\tau_i^l + z - \tau_i^s) \geq II - 1 \\ [\tau_i^s \ mod \ II, \tau_i^s \ mod \ II + (\tau_i^l + z - \tau_i^s)], & if \ (\tau_i^s \ mod \ II + \tau_i^l + z - \tau_i^s) \leq II - 1 \\ [\tau_i^s mod \ II, II-1] \cup [0, (\tau_i^l + z) mod \ II], & otherwise \end{cases} \quad (14)$$

The values $D_i(0)$ for the illustrative example are given in Table 1 and represented in Figure 5(a).

C3. Resource constraints must be satisfied for every control step in the first execution window.

The following system *PR* is equivalent to $\hat{PR}$ and it formalizes the conditions C1, C2 and C3.

$$PR: \begin{cases} \text{Minimize } z \text{ subject to:} & (15) \\ \sum_{t=0}^{II-1} y_{i,t} = 1 \qquad \forall o_i \in O & (16) \\ \sum_{t=0}^{II-1} t y_{i,t} \in D_i(z) \qquad \forall o_i \in O & (17) \\ \sum_{o_i \in T} y_{i,t} \leq M_r \qquad \forall t \in \{0, \ldots, II-1\}, \forall r \in M & (18) \end{cases}$$

where $y_{i,t}$ equals to 1 if an occurrence of operation o_i is scheduled into control step t, zero otherwise. The objective function (15) states that we minimize the iteration time $IT = CP + z$. Constraints (16), (17) and (18) correspond to C1, C2 and C3, respectively.

The system *PR* can be solved optimally in $O(|O|^3)$ using the sub-interval method [23]. We propose a new method which solves the same system optimally in $O(|O|^2)$. The next section presents the theoretical framework of this method.

Table 1: Initial domains of operations for $II = 3$

	o_0	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}	o_{11}
$[\tau_i^s, \tau_i^l]$	[0, 0]	[1, 1]	[2, 3]	[2, 2]	[2, 2]	[3, 3]	[4, 4]	[5, 5]	[6, 6]	[7, 7]	[3, 6]	[4, 7]
$D_i(0)$	[0, 0]	[1, 1]	[0, 0] ∪ [2, 2]	[2, 2]	[2, 2]	[0, 0]	[1, 1]	[2, 2]	[0, 0]	[1, 1]	[0, 2]	[1, 2] ∪ [0, 0]

5.3. Transformation of PR to PRU(S)

In the system, *PR* the initial scheduling domains $\{D_i(0)\}$ of operations could be the union of two disjoint intervals (Equation 14). The goal of the proposed transformation is to construct a new scheduling problem *PRU(S)* equivalent to *PR* where the scheduling domains of operations are single intervals.

PRU(S) is obtained by generating for each operation $o_i \in O$ a number S of operations $o_{i,j}$, $j = 0, 1, \ldots, S-1$, and associating to $o_{i,j}$ an initial single interval domain $[\tau_{i,j}^s, \tau_{i,j}^l]$ defined by:

$$[\tau_{i,j}^s, \tau_{i,j}^l] = [\tau_{i,j-1}^s + II, \tau_{i,j-1}^l + II] \tag{19}$$

with
$$[\tau_{i,0}^s, \tau_{i,0}^l] = \begin{cases} [\tau_i^s \bmod II, \tau_i^s \bmod II + (\tau_i^l - \tau_i^s)], & \text{if } (\tau_i^l - \tau_i^s) \le II - 1 \\ [\tau_i^s \bmod II, \tau_i^s \bmod II + II - 1], & \text{otherwise} \end{cases} \tag{20}$$

Figure 5 illustrates this unfolding like-transformation.

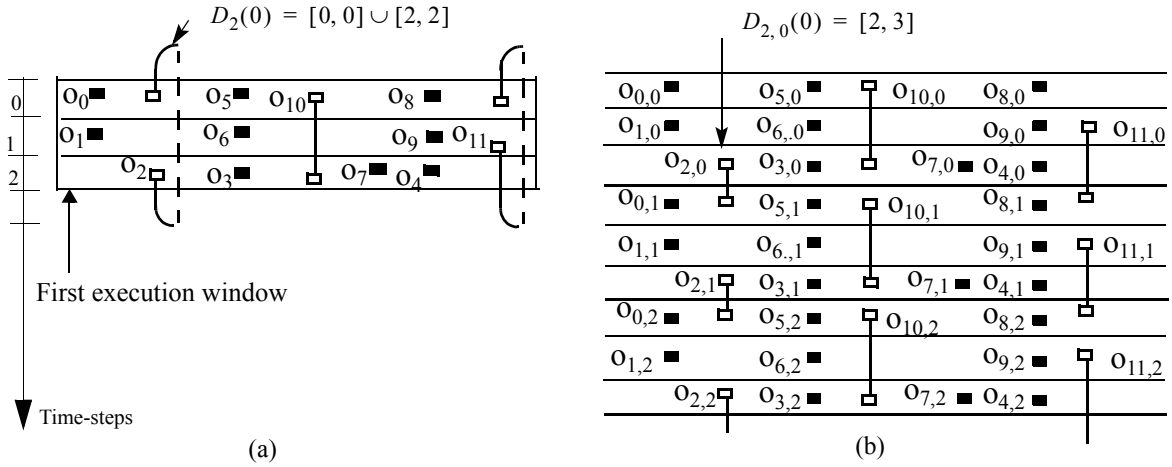


Figure 5. (a) Representation of the initial operation domains $D_i(0)$.
(b) Transformation of the scheduling problem represented in (a).

Let $O_S = \{o_{i,j} \quad / o_i \in O, j = 0, 1, \ldots, S-1\}$. The ILP formulation for the parametrized system *PRU(S)* is:

$$PRU(S)\begin{cases} \text{Minimize } z \text{ subject to:} & (21) \\ \sum_{t=0}^{(S+1)II-1} y_{i,j,t} = 1 \qquad \forall o_{i,j} \in O_S & (22) \\ \tau^s_{i,j} \le \tau_{i,j} \le \tau^l_{i,j} + z \qquad \forall o_{i,j} \in O_S & (23) \\ \sum_{o_{i,j} \in O_S / o_i \in T_r} y_{i,j,t} \le M_r \qquad \forall t \in \{0, 1, \ldots, (S+1)II-1\}, \forall r \in M & (24) \\ \text{where,} \\ \tau_{i,j} = \sum_{t=i.II}^{(j+2)II-1} t y_{i,j,t} \end{cases}$$

$y_{i,j,t}$ equals 1 if operation $o_{i,j}$ is scheduled in control-step t, zero otherwise.

For a fixed value of *S*, the system *PRU(S)* formulates the following scheduling problem: given a set of functional units $\{M_r\}$, a set of operations $O_S = \{o_{i,j}\}$ defined by their initial scheduling domains $\{[\tau^s_{i,j}, \tau^l_{i,j}]\}$, what is the minimal number of additional time-steps necessary to schedule all operations without resource conflict? It was proven in [17] that this type of problems can be solved optimally in polynomial time by a greedy algorithm. This algorithm is given in appendix A.

Theorem 2*: For a fixed value of S, the system* $PRU(S)$ *can be solved optimally in polynomial time.*

Next theorem states that the systems $PRU(2)$ and $PRU(S)$ $\forall S \geq 2$ have the same optimum value of their objective function, and thus it is sufficient to solve $PRU(2)$.

Theorem 3*:* $\forall S \geq 2$, $PRU(S) \Leftrightarrow PRU(2)$

Proof: given in appendix B.

The following expression recapitulates the set of ILP transformations done in this section:

$$P \Rightarrow \hat{PR} \Leftrightarrow PR \Leftrightarrow PRU(2)$$

where $P \Rightarrow \hat{PR}$ means that $\hat{PR}$ is a relaxation of *P*, and $\hat{PR} \Leftrightarrow PR$ means that both systems are equivalent.

6. Lower Bound on the iteration time algorithm

In this section we present the basic algorithm to compute lower bounds on the iteration time under resource constraints. This algorithm is based on the results of Theorems 2 and 3. We show then how the same algorithm is used to compute tight operation mobility-intervals.

6.1. Basic algorithm

Given a constraint graph *CG*, a target initiation interval II , and a set of functional units M_r, the algorithm given in Figure 6 computes a lower bound on the iteration time using three steps: steps 1 and 2 construct the system $PRU(2)$, and step 3 solves it optimally.

Example: We illustrate the *IT_LowerBound* algorithm on the graph of Figure 2(b) under the constraint of one multiplier, one subtracter and four adders. This graph has a critical path length *CP* equal to 8. The sets $\{[\tau_i^s, \tau_i^l]\}$, $\{[\tau_{i,0}^s, \tau_{i,0}^l]\}$ and $\{[\tau_{i,1}^s, \tau_{i,1}^l]\}$ are given in Table 2. Sorting the operations by the increasing values of $\tau_{i,j}^l$, we obtain:

$\{o_{0,0}, o_{5,0}, o_{8,0}, o_{1,0}, o_{6,0}, o_{9,0}, o_{3,0}, o_{4,0}, o_{7,0}, o_{10,0}, o_{11,0}, o_{2,0}, o_{0,1}, o_{5,1}, o_{8,1}, o_{1,1},$

$o_{6,1}, o_{9,1}, o_{3,1}, o_{4,1}, o_{7,1}, o_{10,1}, o_{11,1}, o_{2,1}\}$

As there is only one multiplier unit, it is not possible to schedule operations $o_{3,0}$, $o_{2,0}$ and $o_{0,1}$ inside their respective initial domains $[2, 2]$, $[2, 3]$ and $[3, 3]$, unless the critical path length is increased by one time-step. Similar resource conflicts occur between the multiplications $o_{6,0}$,

$o_{9,0}$, $o_{6,1}$ and $o_{9,1}$ which have as initial domains $[1, 1]$, $[1, 1]$, $[4, 4]$ and $[4, 4]$, respectively. Thus, a lower bound on the iteration time will be $CP + 1 = 9$.

Algorithm: *IT_LowerBound*

input: constraint graph *CG*, target initiation interval *II*, set of resources $\{M_r\}$,

output: lower bound of the iteration time (IT_{LB})

begin

Step 1: Break multi-cycle operations into uni-cycle operations;

Compute the mobility $[\tau_i^s, \tau_i^l]$ of each operation o_i in *CG* using Bellman-Ford algorithm;

$CP = Max(\tau_i^s + 1)$;

Step 2: **for** each operation o_i **do**

Create two operations $o_{i,0}$ and $o_{i,1}$

for $j = 0, 1$ **do**

$\tau_{i,j}^s = \tau_i^s \ mod \ II + j \cdot II$;

$$\tau_{i,j}^l = \begin{cases} \tau_i^s \ mod \ II + (\tau_i^l - \tau_i^s) + j \cdot II & \text{if } (\tau_i^l - \tau_i^s) < II - 1 \\ \tau_i^s \ mod \ II + (II - 1) + j \cdot II & \text{otherwise} \end{cases}$$

end for

end for

Step 3: Sort the set $\{o_{i,j}\}$ by the increasing values of $\tau_{i,j}^l$;

Assign each operation $o_{i,j}$ to the first control step $\tau_{i,j}$ after $\tau_{i,j}^s$ containing a free resource;

$z = max(\tau_{i,j} - \tau_{i,j}^l)$;

$IT_{LB} = z + CP$;

end

Figure 6. Algorithm to find a lower bound of the iteration time.

Table 2: Intermediate results of the *IT_LowerBound* algorithm

	o_0	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}	o_{11}
$[\tau_i^s, \tau_i^l]$	[0, 0]	[1, 1]	[2, 3]	[2, 2]	[2, 2]	[3, 3]	[4, 4]	[5, 5]	[6, 6]	[7, 7]	[3, 6]	[4, 7]
$[\tau_{i,0}^s, \tau_{i,0}^l]$	[0, 0]	[1, 1]	[2, 3]	[2, 2]	[2, 2]	[0, 0]	[1, 1]	[2, 2]	[0, 0]	[1, 1]	[0, 2]	[0, 2]
$[\tau_{i,1}^s, \tau_{i,1}^l]$	[3, 3]	[4, 4]	[5, 6]	[5, 5]	[5, 5]	[3, 3]	[4, 4]	[5, 5]	[3, 3]	[4, 4]	[3, 5]	[4, 5]

Algorithm complexity: Operation mobility-intervals are computed using Bellman-Ford algorithm [36] in $O(|O||E|)$, where O and E are the set of nodes and the set of arcs in the constraint graph respectively. As an operation has in average two input arcs, the complexity of Bellman-Ford algorithm is in $O(|O|^2)$. The sorting and the greedy scheduling of $2 \cdot |O|$ operations can be done in $O(|O|^2)$. Thus, the algorithm complexity is $O(|O|^2)$. Notice that this complexity is minimal since computing the earliest starting times in a cyclic graph is in $O(|O|^2)$.

6.2. Operation mobility-intervals under pipelining and resource constraints

An important property of the *IT_LowerBound* algorithm (Figure 6) is that: the more accurate are the operation mobility-intervals $\{[\tau_i^s, \tau_i^l]\}$, the tighter lower bound is returned. This section presents a method to compute more accurate operation mobility-intervals. The method is based on the technique introduced initially in [16] for computing lower-bound performance of non-pipelined schedules in acyclic data flow graphs.

To obtain tight operation mobility-intervals, three types of constraints are considered simultaneously: timing constraints, resource constraints and pipelining. Let $[{}^{rc}\tau_i^{\,s}, {}^{rc}\tau_i^{\,l}]$ denote the mobility-interval of o_i under the three previous constraints. Let CG_{o_i} be a subgraph of the constraint graph CG containing operation o_i and all operations o_j in CG such that there exists at least one path in CG from o_j to o_i having only positive edges (see Figure 7(a)). Tight value of ${}^{rc}\tau_i^{\,s}$ is obtained by performing the *IT_LowerBound* algorithm on CG_{o_i}: as operation o_i is always the latest operation executed among operations in CG_{o_i}, a lower bound on the iteration time of CG_{o_i} constitutes an earliest starting time of o_i. The order of computing $\{{}^{rc}\tau_i^{\,s}\}$ is important, because it influences their final values. An efficient order consists of computing ${}^{rc}\tau_i^{\,s}$ only after computing all ${}^{rc}\tau_j^{\,s}$ such that o_j is an immediate predecessor of o_i and $\hat{w}(o_i \rightarrow o_j) > 0$.

By inverting the direction of edges in *CG,* the same technique allows to compute $\{{}^{rc}\tau_i^{\,l}\}$. Figure 7(b) shows the accuracy improvement of mobility-intervals for the illustrative example. Notice that the lower bound on the iteration of the whole graph is also improved from 9 to 10.

The use of tight operation mobility-intervals reduces automatically the running time of both exact and heuristic schedulers.

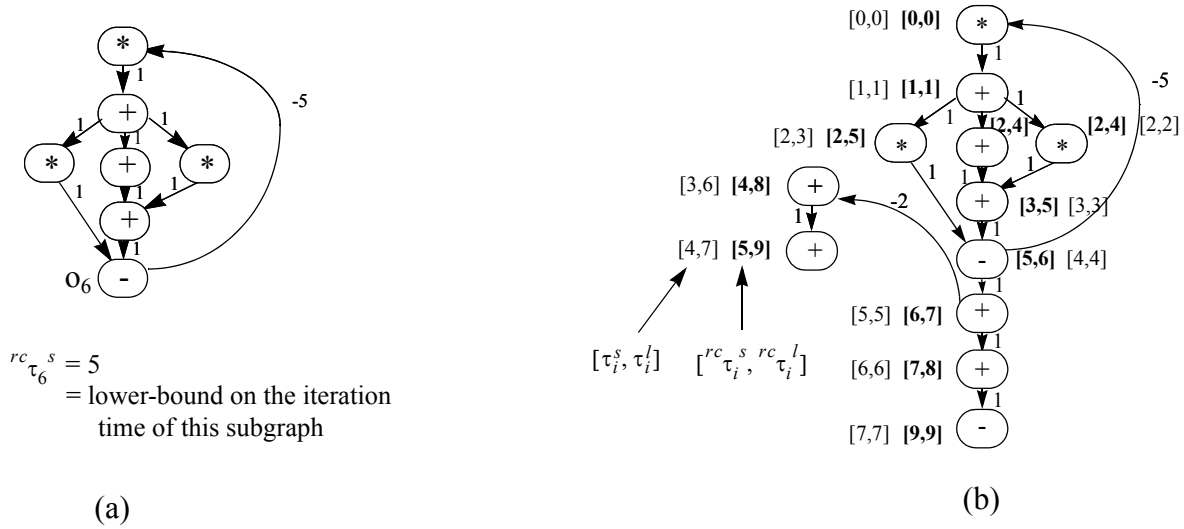


Figure 7. (a) Subgraph CG_{o_6}. (b) Operation mobility-intervals under pipelining and resource constraints (4 ✣, 1 ✻, 1 —).

Algorithm: *II_LowerBound*

input: constraint graph *CG*, set of resources $\{M_r\}$

output: lower bound on the initiation interval (II_{LB})

begin

$II_{LB} = Max\{II_{RC}, II_{LDC}\}$;

$SG = \{CG_{o_i \to o_j} \mid \hat{w}(o_i \to o_j) < 0\}$;

while there are timing-constraint inconsistencies **do**

for each subgraph $CG_{o_i \to o_j}$ of the set SG **do**

$IT = IT\_LowerBound\,(CG_{o_i \to o_j},\ II_{LB},\ \{M_l\})$;

if $IT > \left|\hat{w}(o_i \to o_j)\right|$ **then** /* timing constraint inconsistency*/

$II_{LB} = II_{LB} + 1$;

else

$\hat{w}(o_j \to o_i) = IT$;

end for

end while

end

Figure 8. Algorithm to find a lower bound on the initiation interval

7. Lower bound on the initiation interval algorithm

The description of the algorithm to find a lower bound on the initiation interval is given in Figure 8. Starting from the minimal value given by Theorem 1, the initiation interval will be increased iteratively as long as the current value induces timing-constraint inconsistencies in the constraint graph *CG*. For a target initiation interval, timing-constraint consistencies are checked as follows: for each arc $(o_i \rightarrow o_j)$ in *CG* having a negative weight $(\hat{w}(o_i \rightarrow o_j) < 0)$, we compute a lower bound on the number of control steps separating operation o_i from o_j in any feasible pipelined schedule. This lower bound is obtained by performing the *IT_LowerBound* algorithm on a specific subgraph $CG_{o_i \rightarrow o_j}$ of GC. The subgraph $CG_{o_i \rightarrow o_j}$ contains operations o_i, o_j and all operations o_k such that o_k belongs to at least one path from o_j to o_i in *CG* having only positive weights assigned to edges. If the obtained lower bound is greater than $|\hat{w}(o_i \rightarrow o_j)|$, then the current initiation interval is not feasible and it is increased, otherwise this lower bound will be the new weight of the edge $(o_j \rightarrow o_i)$.

Each pass of the algorithm has a complexity of $O(m \cdot |O|^2)$, where *m* is the number of negative arcs in the constraint graph.

8. Experimental results

We tested the estimation method on different high level synthesis benchmarks. For each benchmark, lower bounds were computed and compared to realizable solutions obtained by the pipelined scheduling systems Theda.Fold [6] and PLS [2]. These realizable solutions are considered as upper bounds. Algorithms are implemented in the C language and run on a SPARC 10 station. Experimental results are shown in Tables 3 to 8, where the field *RC* indicates the number of resources, II_{UB} and IT_{UB} are respectively the upper bounds on the iteration time and the upper bound on the initiation interval published in [2,6], II_{LB} and IT_{LB} are the computed lower bounds. Unless it is specified, we assume that multipliers take two clock-cycles while adders and subtracters take one clock-cycle. ($*^P$) denotes multipliers with two stages pipelining.

The Fifth-order digital filter [37]: this benchmark contains 26 additions and 8 multiplications, it has a large number of intra-iteration and inter-iteration precedence constraints and many loops carried dependencies (LCDs). The minimal initiation interval due to LCDs is equal to 16. Table 3

shows the experimental results for this benchmark.

The second-order IIR filter (Table 4): its algorithmic description and the corresponding cyclic DFG are shown in Figure 9.

The third-order IIR filter [30] (Table 5)*:* it contains 6 additions, 2 multiplications and 3 LCDs. The minimal initiation interval due to LCDs is equal to 3.

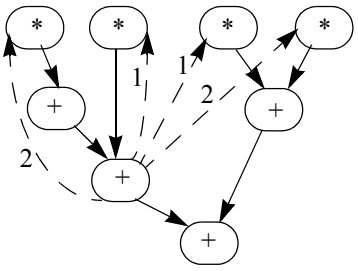
The 16-point FIR filter [1] (Table 6): it contains 15 additions, 8 multiplications and only intra-iteration data dependencies. We suppose that an addition has a duration of 40ns, a multiplication has a duration of 80ns, the clock cycle length is equal to 100ns, and operation chaining are permitted.

The Fast Discrete Cosine Transformation (Table 7): This benchmark is relatively large, it contains 13 additions, 13 subtractions and 16 multiplications. The algorithmic description of this benchmark is a straight-line code, not a loop statement. But we can consider it as a loop statement if we assume that the algorithm is to be performed on many sets of data.

The Fifth-order Digital Wave Filter with no LCDs (Table 8)*:* It is the same benchmark as the first one where all the inter-iteration precedence constraints are removed.

$$
\begin{array}{l}
for\ \ n = 1\ \ \ to\ \ \infty \\
\qquad Y(n) = X(n) + \sum_{k=1}^{2} a_k \cdot X(n-k) + \sum_{k=1}^{2} b_k \cdot Y(n-k) \\
end
\end{array}
$$

(a)



(b)

Figure 9. (a) The behavioral description of the second-order IIR filter.
(b) The corresponding cyclic DFG.

Table 3: Fifth order digital wave filter

RC	II_{UB}	IT_{UB}	II_{LB}	IT_{LB}	$IT_{UB} - IT_{LB}$
3 ✣, 3 ✳	16	18	16	17	1
3 ✣, 2 ✳	16	18	16	18	0
2 ✣, 2 ✳	17	19	**16**	18	1
2 ✣, 1 ✳	19	21	**17**	21	0
3 ✣, 2 $✳^P$	16	18	16	17	1
3 ✣, 1 $✳^P$	16	18	16	18	0
2 ✣, 1 $✳^P$	17	19	17	18	1
1 ✣, 1 $✳^P$	28	28	28	28	0

Table 4: Second-order IIR filter

RC	II_{UB}	IT_{UB}	II_{LB}	IT_{LB}	$IT_{UB} - IT_{LB}$
2 ✣, 3 ✳	3	6	3	5	1
2 ✣, 2 ✳	4	6	4	6	0
1 ✣, 2 ✳	4	7	4	6	1
1 ✣, 1 ✳	8	10	8	10	0
2 ✣, 2 $✳^P$	3	5	3	5	0
1 ✣, 2 $✳^P$	4	6	4	6	0
1 ✣, 1 $✳^P$	4	7	4	6	1

Table 5: Third-order IIR filter

RC	II_{UB}	IT_{UB}	II_{LB}	IT_{LB}	$IT_{UB} - IT_{LB}$
2 ✣, 2 ✳	3	5	3	5	0
2 ✣, 1 ✳	4	6	4	6	0
1 ✣, 1 ✳	6	6	6	6	0
2 ✣, 1 $✳^P$	3	5	3	5	0
1 ✣, 1 $✳^P$	6	6	6	6	0

Table 6: 16-point FIR filter

RC	II_{UB}	IT_{UB}	II_{LB}	IT_{LB}	IT_{UB} - IT_{LB}
15 ✣, 8 ✳	1	6	1	6	0
8 ✣, 4 ✳	2	6	2	6	0
6 ✣, 3 ✳	3	6	3	6	0
5 ✣, 3 ✳	3	6	3	6	0
4 ✣, 2 ✳	4	6	4	6	0
3 ✣, 2 ✳	5	7	5	6	1
2 ✣, 1 ✳	8	10	8	10	0
1 ✣, 1 ✳	15	16	15	15	1

Table 7: Fast discrete cosine transformation

RC	II_{UB}	IT_{UB}	II_{LB}	IT_{LB}	IT_{UB} - IT_{LB}
5✣, 5—, 6✳^{P}	3	9	3	9	0
4✣, 4—, 4✳^{P}	4	10	4	9	1
3✣, 3—, 4✳^{P}	5	10	5	9	1
3✣, 3—, 3✳^{P}	6	11	6	9	2
2✣, 2—, 2✳^{P}	8	12	8	11	1
1✣, 1—, 2✳^{P}	13	16	13	15	1

Table 8: Fifth order digital wave filter with no LCD’s

RC	II_{UB}	IT_{UB}	II_{LB}	IT_{LB}	IT_{UB} - IT_{LB}
26 ✣, 8 ✳^{P}	1	17	1	17	0
13 ✣, 4 ✳^{P}	2	17	2	17	0
9 ✣, 3 ✳^{P}	3	18	3	17	1
7 ✣, 2 ✳^{P}	4	19	4	18	1
6 ✣, 2 ✳^{P}	5	19	5	18	1
5 ✣, 2 ✳^{P}	6	17	6	17	0

Table 8: Fifth order digital wave filter with no LCD's

RC	II_{UB}	IT_{UB}	II_{LB}	IT_{LB}	IT_{UB} - IT_{LB}
4 ✣, 2 ✳^{P}	7	18	7	18	0
4 ✣, 1 ✳^{P}	8	20	8	20	0
3 ✣, 1 ✳^{P}	9	22	9	20	2

The last column in Tables 3 to 8 shows that the lower bounds on the iteration time (IT_{LB}) obtained are close to the upper-bounds (IT_{UB}): the worst results are within two times-steps from the upper bounds. The running time of the algorithm is less than 0.08 seconds for all benchmarks. These results illustrate that the relaxation done on the minimum iteration time problem which is NP-complete is efficient: it does not induce a large accuracy lost, while using a polynomial execution time.

For the initiation intervals, the obtained lower bounds (II_{LB}) are equal to the upper bounds (II_{UB}) except for two experimental cases shown in bold in table 3. We have observed that for these benchmarks the initial lower bound given by Theorem 1 is often feasible, thus this initial bound constitutes a good starting point for the *II_LowerBound* algorithm.

During the scheduling of a data-flow graph, lower bounds constitute good measures to evaluate the performance quality of a solution and to decide when to stop the research of new solutions. If the difference between the performance of the current solution and the lower bound is equal to zero then the solution is optimum. Otherwise, this difference indicates the maximal performance improvement that could be achieved by continuing the research (or the maximal performance loss by choosing such solution). For the above benchmarks, 58% of the scheduling solutions obtained by the heuristics [2,6] are optimum, 37% are at maximum one control step from the optimum, and 5% are at maximum two control steps from the optimum.

9. Conclusions

Fast algorithms for performance estimation allow to make efficient design space exploration and to improve the quality and the performance of pipelined scheduling heuristics. In this papers, we have presented a new method for computing lower bounds on the iteration time and on the

initiation interval of pipelined datapath implementations under functional unit constraints. The method handles behavioral description containing loop statements and timing constraints. And it handles implementations with operation chaining, multicycle operations, and pipelined functional units. Based on an efficient relaxation of the general pipelined scheduling problem, we have developed lower-bound algorithms with complexities of ($O(|O|^2)$ and $O(m \cdot |O|^2)$). The dual problem of computing lower bound on the resource requirement to achieve a target performance can be solved using the same method. Applications of our approach are not limited to the area of high level synthesis, it can be used in different areas of computer design where functional pipelining and loop folding are used.

Appendix A

$$
PRU(S)\begin{cases}
\text{Minimize } z \text{ subject to:} & (25)\\
\sum_{t=0}^{(S+1)II-1} y_{i,j,t} = 1 \qquad \forall o_{i,j} \in O_S & (26)\\
\tau^s_{i,j} \le \tau_{i,j} \le \overline{\tau^l_{i,j}} = \tau^l_{i,j} + z \qquad \forall o_{i,j} \in O_S & (27)\\
\sum_{o_{i,j} \in O_S / o_i \in T_r} y_{i,j,t} \le M_r \qquad \forall t \in \{0, 1, \ldots, (S+1)II-1\}, \forall r \in M & (28)\\
\text{where} & \\
\tau_{i,j} = \sum_{t=j\cdot II}^{(j+2)II-1} t y_{i,j,t} & (29)\\
\tau^s_{i,j} = \tau^s_{i,0} + j\cdot II = \tau^s_i \bmod II + j\cdot II & (30)\\
\tau^l_{i,j} = \tau^l_{i,0} + j\cdot II = \begin{cases} \tau^s_i \bmod II + (\tau^l_i - \tau^s_i) + j\cdot II & \text{if } (\tau^l_i - \tau^s_i) < II-1 \\ \tau^s_i \bmod II + (II-1) + j\cdot II & \text{otherwise} \end{cases} & (31)
\end{cases}
$$

Algorithm: *Greedy-scheduling*

Input: instance of the system $PRU(S)$ with a fixed value of *S,* set of resources

output*:* optimal value of the objective function *z* of $PRU(S)$

begin

Sort operations $o_{i,j}$ by the increasing values of $\tau^l_{i,j}$;

Assign each operation $o_{i,j}$ to the first step $\tau_{i,j}$ after $\tau^s_{i,j}$ which contains a free resource*;*

$z = max(\tau_{i,j} - \tau^l_{i,j})$ *;*

end

Appendix B

The following proposition is used in the proof of Theorem 3.

Proposition 1

(a) For any value of *z* in *PRU(S)*, the maximal number of operations $o_{i,j}$ of a type r such that all the values of $\overline{\tau_{i,j}^{l}}$ belong to a same integer interval of size II, is less or equal to $|T_r|$. Where $|T_r|$ is by definition the number of operations $o_{i,0}$ of type r in *PRU(S)*.

(b) If there exist two operations $o_{i,j}$ and $o_{k,l}$ in the system *PRU(S)* such that the corresponding values $\tau_{i,j}^{s}$ and $\tau_{k,l}^{s}$ belong to a same integer interval of size II, then $|l-j| \leq 1$.

Proof:

(a) By Equations (27) and (31) we have:

$$\tau_{i,j} \leq \overline{\tau_{i,j}^{l}} = \tau_{i,j}^{l} + z \qquad \forall o_{i,j} \in O_S \tag{32}$$

$$\tau_{i,j+1}^{l} = \tau_{i,j}^{l} + II$$

$$\Rightarrow \qquad \tau_{i,j+1} \leq \overline{\tau_{i,j+1}^{l}} = \tau_{i,j}^{l} + II + z \tag{33}$$

From (32) and (33) we deduce:

$$\left|\overline{\tau_{i,j}^{l}} - \overline{\tau_{i,k}^{l}}\right| \geq II \qquad \forall j \neq k$$

Thus, for any integer interval of size II, it is not possible to have two operations $o_{i,j}$ and $o_{i,k}$ such that both values $\overline{\tau_{i,j}^{l}}$ and $\overline{\tau_{i,k}^{l}}$ belong to this interval. As by definition the number of operations $o_{i,j}$ of the same type *r* and for a fixed index *j* is equal to $|T_r|$, then the number of operations $o_{i,j}$ having $\overline{\tau_{i,j}^{l}}$ included in the same interval of size II is less or equal to $|T_r|$.

(b) From Equation (30) we have:

$$\tau_{i,j}^{s} = \tau_{i}^{s} \bmod II + j \cdot II < (j+1) \cdot II \qquad \forall o_{i,j} \in O_S$$

$$\Rightarrow \qquad \tau_{k,j+2}^{s} = \tau_{k}^{s} \bmod II + (j+2) \cdot II \geq (j+2) \cdot II$$

$$\Rightarrow \qquad \tau_{k,j+2}^{s} - \tau_{i,j}^{s} > II \tag{34}$$

From (34) we deduce that if there exist two operations $o_{i,j}$ and $o_{k,l}$ such that $\tau_{i,j}^{s}$ and $\tau_{k,l}^{s}$ belong to a same interval of size *II*, then $|l-j| \leq 1$. ❑

Theorem 3: $\forall S \geq 2$, $PRU(S) \Leftrightarrow PRU(2)$

Proof:

The proof is by contradiction. Suppose that the optimum value of the objective function of $PRU(2)$ is equal to z_0, and that z_0 is also an optimum solution of $PRU(n)$ for $n \geq 2$ but not for $PRU(n+1)$. This implies that if we fix z to z_0 in the constraint (27) of the system $PRU(S = n+1)$, and we use the greedy algorithm (given in appendix A) to schedule operations $o_{i,j}$ under resource constraints, then no feasible solution can be found. We will prove that such case cannot happen.

Let be $o_{g,h}$ the first operation which can not be scheduled before its latest starting time $\overline{\tau^l_{g,h}}$ due to the lack of resources, and let *r* the type of $o_{g,h}$. Depending on the number of resources already used between the control steps ($\overline{\tau^l_{g,h}} - II + 1$) and $\overline{\tau^l_{g,h}}$, we distinguish two cases.

Case 1 In every control step of the interval $[\overline{\tau^l_{g,h}} - II + 1, \overline{\tau^l_{g,h}}]$ all the M_r resources are used. Since by Theorem 1, the value of *II* must satisfy $M_r \cdot II \geq |T_r|$, we deduce that there are at least $|T_r|$ operations already scheduled in this interval. Thus, there are at least $|T_r| + 1$ operations $o_{i,j}$, including the operation $o_{g,h}$, such that their values $\overline{\tau^l_{i,j}}$ belong to the interval $[\overline{\tau^l_{g,h}} - II + 1, \overline{\tau^l_{g,h}}]$. This is in contradiction with Proposition 1.(a)

Case 2 There is at least one control step p in $[\overline{\tau^l_{g,h}} - II + 1, \overline{\tau^l_{g,h}}]$ which contains free resources. As we use the greedy scheduling technique, all operations scheduled after the control step p have the values of $\tau^s_{i,j}$ strictly greater than p. Let A be the set of operations $o_{i,j}$ already scheduled inside the interval $[p+1, \overline{\tau^l_{g,h}}]$. Since it was not possible to schedule operation $o_{g,h}$ before the control step $\overline{\tau^l_{g,h}}$, it implies that the following system *SS* does not admit a solution:

$$SS: \begin{cases} \sum_{t=p+1}^{\overline{\tau^l_{g,h}}} y_{i,j,t} = 1 & \forall o_{i,j} \in A \cup \{o_{g,h}\} \\ \sum_{o_{i,j} \in A \cup \{o_{g,h}\} / o_i \in T_r} y_{i,j,t} \leq M_r & \forall t \in \{p+1, \ldots, \overline{\tau^l_{g,h}}\}, r \in M \\ p+1 \leq \tau^s_{i,j} \leq \tau_{i,j} \leq \tau^l_{i,j} + z_0 & \forall o_{i,j} \in A \cup \{o_{g,h}\} \end{cases}$$

where $\tau_{i,j}$, $\tau^s_{i,j}$ and the set of constraints have the same meaning as in the system *PRU(S)*.

As the size of the interval $[p+1, \overline{\tau^l_{g,h}}]$ is less than II, from Proposition 1.(b) we deduce that for any pair of operations $(o_{i,j}, o_{k,l})$ which belong to the set $A \cup \{o_{g,h}\}$ we have $|l-j| \leq 1$. By substituting in the system SS the smaller index j by 0 and the index $j+1$ by 1, we obtain an equivalent system which is completely included in the system $PRU(2)$. This is in contradiction with the hypothesis that $PRU(2)$ has a solution. ❑

References

[1] N. Park, A. C. Parker, "Sehwa: a software package for synthesis of pipelines from behavioral specifications", IEEE trans. on CAD, vol. 7, pp. 356-370, March 1988.

[2] C. T. Hwang, Y. C. Hsu, Y. L. Lin, "PLS: A Scheduler for pipeline Synthesis" IEEE Trans. on CAD, vol. 12, pp. 1279-1286, September 1993.

[3] R.R. Potasman, J. Lis, A. Nicolau, D. Gajski, "Percolation based synthesis" in proceedings of the 27th Design Automation Conference, pp. 444-449, 1990.

[4] G. Goossens, J. Rabaey, J. Vandewalle, H. D. Man, "An efficient microcode compiler for application specific DSP processors" IEEE Trans. on CAD, vol. 9, September 1990.

[5] C.-Y. Wang, K. K. Parhi, "Loop list scheduler for DSP algorithms under resource constraints" in proceedings of ISCAS, pp. 1662-1665, 1993.

[6] T. F. Lee, A. C. H. Wu, D. D. Gajski, "A transformation-based method for loop folding" IEEE Trans. on CAD, vol. 13, no. 4, pp. 439-450, April 1994.

[7] L.F. Chao, A. LaPaugh, "Rotation scheduling: a loop pipelining algorithm" in proceedings of the 30th Design Automation Conference, pp. 566-572, 1993.

[8] M. Sonia, H. Gerez, E. H. Otto, "Range-chart-guided iterative data-flow graph Scheduling" IEEE Trans. on Circuits and Systems, vol. 39, pp. 351-364, May 1992.

[9] P. G. Paulin, J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's" IEEE trans. on CAD, vol. 8, June 1989.

[10] I.-C. Park, C.-M. Kyung, "FAMOS: An efficient scheduling algorithm for high-level synthesis" IEEE trans. on CAD, vol. 12, pp. 1437-1448, October. 1993.

[11] A. Aiken, A. Nicolau, "A realistic Resource-Constrained Software Pipelining Algorithm", Advances in Languages and Compilers for Parallel Processing, MIT Press, 1991.

[12] C. T. Hwang, J. H. Lee, Y. C. Hsu, "A formal approach to the scheduling problem in high level synthesis" IEEE Trans. on CAD, vol. 10, pp. 669-683, April 1991.

[13] C. E. Leiserson, J. B. Saxe, "Retiming synchronous circuitry" Algorithmica, vol. 6, pp. 5-

35, 1991.

[14] A. Sharma, R. Jain, “Estimation architectural resources and performance for high-level synthesis applications”, IEEE trans. on VLSI, vol. 1, pp. 175-190, 1993.

[15] J. Rabaey, M. Potkonjak, “Estimating implementation bounds for real time DSP application specific circuits”, IEEE trans. on CAD, vol. 13, June 1994.

[16] M. Langevin, E. Cerny, “A recursive technique for computing lower-bound performance of schedules” in proceedings of ICCD, pp. 16-20, 1993.

[17] M. Rim, R. Jain, “Lower-bound performance estimation for the high-level synthesis scheduling problem” IEEE trans. on CAD, vol. 13, pp 81-88, 1994.

[18] T. C. Hu, “Parallel sequencing and assembly line problems” Operations Research, vol. °9, pp. 841-848, 1961.

[19] M. A. Al-Mouhamed, “Lower bound on the number of processors and time for scheduling precedence graphs with communication cost”, IEEE tran. on Software Engineering, vol. 16, December 1990.

[20] S. Chaudhuri, R. A. Walker, “Computing lower bounds on functional units before scheduling” in proceedings of the 7th Intern. Symp. on High Level Synthesis, pp. 36-41, 1994.

[21] R. Reiter, “Scheduling parallel computation” Journal of the ACM, vol. 15, pp. 590-599, 1968.

[22] R. Jain, A. C. Parker, N. Parker, “Predicting system-level area and delay for pipelined and non-pipelined designs” IEEE trans. on CAD, vol. 11, August 1992.

[23] Y. Hu, A. Ghouse, B. S. Carlson, “Lower bounds on the iteration time and the number of resources for functional pipelined data flow graphs” in proceedings of ICCD, pp. 21-24, 1993.

[24] S. H. Gerez, S. M. H de Groot, O. E. Herrmann, “A polynomial-time algorithm for the computation of the iterative-period bound in recursive data-flow graphs” IEEE Trans. on Circuits and Systems, vol. 39, pp. 49-51, January 1992.

[25] A. Aiken, A. Nicolau, “Optimal loop parallelization” SIGPLAN, vol. 23, No. 7, pp. 308-317, 1988.

[26] K. Iwano, S. Yeh, “An efficient algorithm for optimal loop parallelization” Algorithms, Lect. Notes in Computer Science, vol. 450, pp. 202-210, 1990.

[27] A. Zaky, P. Sadayappan, “Optimal static scheduling of sequential loops on multiprocessors” in proceedings of the Intern. Confer. on Parallel Processing, vol. III, pp. 130-137, 1989.

[28] M. Potkonjak, J. Rabaey, “Optimizing throughput and resource utilization using pipelining: transformation based approach” Journal of VLSI Signal Processing, vol. 8, pp. 117-130,

1994.

[29] K. K. Parhi, D. G. Messerschmitt, “Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding” IEEE trans. on Computers, vol. 40, pp. 178-194, February 1991.

[30] L.-G. Jeng, L.-G. Chen, “Rate-optimal DSP synthesis by pipeline and minimum unfolding” IEEE trans. on VLSI, vol. 2, pp. 81-88, March 1994.

[31] L. E. Lucke, K. K. Parhi, “Data-flow transformations for critical path time reduction in high level DSP synthesis” IEEE trans. on CAD, vol. 12, pp. 1063-1068, July. 1993.

[32] G. De Micheli, “Synthesis and optimization of digital circuits”, McGraw-Hill, 1994.

[33] M. Potkonjak, J. Rabaey, “Optimizing resource utilization using transformations” IEEE trans. on CAD, vol. 13, pp. 277-292, Marsh 1994.

[34] M. B. Srivatava, M. Potkonjak “Transforming linear systems for joint latency and throughput optimisztion”, in proceedings of EDAC, pp. 267-271, 1994.

[35] C.Y. Chen, M. Morcz, “A delay distribution methodology for the optimal systolic synthesis of linear recurrence algorithms” IEEE trans. on CAD, vol. 10, pp. 685-697, June 1991.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, “Introduction to algorithms” MIT Press and Mc Graw Hill, 1990.

[37] S. Y. Kung, H.J. Witehouse, T. Kailath, “VLSI and modern signal processing”, Prentice Hall, pp. 258-264, 1985.

[38] R. Roundy, “Cyclic schedules for job shops with identical jobs”, Mathematics of operations research, Vol. 17, No. 4, pp 842-865, November 1992.