Range-Chart-Guided Iterative Data-Flow Graph Scheduling

Sonia M. Heemstra de Groot, *Member, IEEE,* Sabih H. Gerez, and Otto E. Herrmann, *Member, IEEE*

Abstract—**This paper describes an alternative method for the scheduling of iterative data-flow graphs. The method is based on the scheduling-range chart, which contains the information on the range within which each operation in the graph can be scheduled. The scheduling range is determined by considering the intra- and inter-iteration precedence relations. The goal is to find an optimal position within the scheduling range of each operation in such a way that some quality criteria (number of hardware resources, iteration period, latency, register life-time) are optimized. A formal proof of the NP-completeness of the problem is given and two polynomial-time heuristics are introduced: fixed-rate (rate-optimal as a special case) scheduling where the number of hardware resources is optimized at the same time that a specific iteration period is guaranteed, and maximum-throughput scheduling with limited resources where the iteration period is optimized for a fixed number of processors. The algorithms are able to find optimal solutions for well-known benchmark examples.**

I. Introduction

In order to achieve fast implementations of computation-intensive digital signal processing (DSP) algorithms, parallel processing is necessary. As a consequence, tools to detect parallelism and appropriate scheduling techniques have to be developed.

DSP algorithms are characterized by their infinite repetition, which allows the exploitation of parallelism not only between operations of the same iteration (*intra-iteration parallelism*) but also the parallelism between operations of consecutive iterations (*inter-iteration parallelism*) [1]. Besides, for a large number of algorithms without data-dependent conditionals, it is possible to apply deterministic scheduling techniques at compile or design time [2].

The use of the *data-flow graph* model to describe DSP algorithms is very extended [2]–[5]. Some scheduling techniques for *iterative data-flow graphs* (IDFG's) are based on applying traditional single-iteration scheduling methods [6]–[8] for optimizing the schedule length of one iteration of the algorithm (or a finite number of algorithm iterations in the case of direct blocking [9], [10]). These techniques totally or partially ignore the repetitive nature of the algorithms, overlooking in this way part of the parallelism [11].

On the other hand, a number of nonpreemptive deterministic scheduling techniques that are able to exploit inter- and intra-iteration parallelism have been reported in the literature (see for example [1], [2], [5], [12], and [13]).

In *acyclic IDFG's,* by using pipelining, the iteration period, T_0, can be reduced until the limit imposed by the computational time of the tasks in the graph and the available computing power. In contrast, the minimum iteration period for *cyclic IDFG's* depends not only on those factors, but also on the topology of the graph. The minimum iteration period of a cyclic IDFG, for an unconstrained number of hardware resources, is called the *iteration period bound* $T_{0\min}$, and is determined by the *critical loop* [12] of the graph. Schedules that can be executed at this iteration period are called *rate-optimal schedules.*

Basically, three scheduling techniques that provide such schedules have been reported: maximum-spanning tree [12], search of cyclo-static schedules [5], and optimum unfolding [1].

The first has the disadvantage that it does not try to optimize the number of processors, leading in general to solutions with suboptimal processor utilization [11]. The second consists of a depth-first search of cyclo-static solutions, by fixing both the iteration period and the number of processors. This method cannot guarantee a schedule since a solution may not exist when both parameters are fixed (see Section V). The third method consists of reducing the graph to an equivalent perfect-rate data-flow program that can always be scheduled rate-optimally. This is performed by unfolding with an optimal unfolding factor and scheduling the resulting graph. The main disadvantage is that the memory space necessary to store the program increases proportionally with the unfolding factor.

This paper is focused on an alternative approach to the scheduling of IDFG's based on the scheduling-range chart. The goal is to find an optimal position within the scheduling window of each operation in such a way that some quality criteria (number of hardware resources, iteration period, latency, register life time) are optimized. Based on this approach, two types of algorithms are introduced: *fixed-rate* (rate-optimal as a special case) *scheduling,* where the number of hardware resources is optimized at the same time a specific iteration period is guaranteed, and *maximum-throughput scheduling with limited resources,* where for a given number of processors the iteration period is minimized.

The structure of this paper is as follows. Section II introduces the IDFG model and some concepts of graph theory that will be used in this text. Section III describes the multiprocessor scheduling problem for IDFG's. The notions

Manuscript received July 12, 1990; revised August 7, 1991. This paper was recommended by Associate Editor K. Thulasiraman.

The authors are with the Faculty of Electrical Engineering, University of Twente, 7500 AE Enschede, The Netherlands.

IEEE Log Number 9108174.

of scheduling range and scheduling-range charts for noniterative and IDFG's are given in Section IV. Section V presents some bounds related to the scheduling problem and discusses why fixing both the number of processors and the iteration period might not lead to a solution. Two scheduling algorithms based on the scheduling-range concept, one for optimization of hardware resources and another for optimization of the iteration period, are presented in Section VI. Section VII discusses an efficient algorithm for the computation of the scheduling-range chart. The overall time complexity of the proposed algorithms is analyzed in Section VIII. Section IX compares the proposed algorithms with other scheduling algorithms for IDFG's, while experimental results are presented in Section X. The effects of the introduction of non-negligible communication delays to the scheduling problem are considered in Section XI. The main conclusions are summarized in Section XII. Finally, a formal proof of the NP-completeness of the scheduling of IDFG's is given in Appendix A.

II. The IDFG Model

In order to approach the scheduling problem, it is necessary to define a clear model to represent the computations and data dependencies in a DSP algorithm. This section discusses the IDFG, a special case of a data-flow graph.

The family of DSP algorithms that can be modeled by a synchronous data-flow graph [3] is of special interest since the scheduling can be performed statically, at compile or design time. This covers a large group of algorithms; the only restriction is that they may not contain data-dependent conditionals. Multiple sampling rates and hierarchy are easily expressed within this model.

An IDFG is a synchronous homogeneous data-flow graph [3], IDFG(V, E). $V = C \cup D \cup I \cup O$ constitutes the set of vertices (C, D, I, and O are pairwise disjoint). C is the set of vertices associated with computational nodes (or operations), D is the set of delay elements, and I and O are the set of inputs and set of outputs, respectively. An edge $e \in E$ can connect any pair of vertices in V with the constraint that no edge is incident *to* an input and no edge is incident *from* an output.

In IDFG's there is a token stream flowing from the input nodes that make the nodes to be fired repetitively, producing token streams flowing to the output nodes. The set of computations fired as a consequence of one set of tokens at the input nodes will be referred to as the computation of one iteration. Each token can be indexed by the iteration number in which it was produced. The time between two consecutive firings of a node will be called the *iteration period,* T_0.

Data dependencies between nodes fired by tokens of different iterations are indicated in IDFG's by using the *delay* node. A delay node delays a token it receives for a number of iteration periods. For $v \in D$, this number is denoted by $v.multiplicity$.[1] For $v \in C \cup I \cup O$, $v.multiplicity = 0$.

[1] This notation should be understood as the selection of a field in a record in a Pascal-like language.

The duration of the computation performed by a vertex $v \in C$ is given by $v.duration$. For $v \in D \cup O \cup I$, $v.duration = 0$ by definition. The computational nodes in this paper are assumed to be *atomic* and *nonpreemptive.*

In contrast to IDFG's the nodes of a *noniterative* DFG are fired only once. In a noniterative DFG, delay nodes do not have any meaning.

Here we introduce some well-known concepts of graph theory [14] that are used further in the text. A *directed path* in the graph starts at a certain vertex, goes through zero or more vertices, and finishes in another vertex following the edges in the graph according to their orientations. When the first and the last vertex of a directed path coincide, the path is called a *directed loop.* A directed path or loop is *simple* when the path or loop passes at most once through each vertex in the graph. In the rest of the text a directed path will be called a *path* and a directed loop will be called a *loop.* In an IDFG, all loops contain at least one delay element. Noniterative DFG's are always acyclic.

Two nodes $v_i, v_j \in V$ are called *strongly connected* if there is a path from v_i to v_j and a path from v_j to v_i. A subgraph of the IDFG such that all possible pairs of vertices in the graph are strongly connected is called a *strongly connected component.* Note that a partitioning of the graph into strongly connected components can contain subgraphs composed by a single vertex.

In this paper it is assumed that the DSP algorithm is modeled as an IDFG. As an example, Fig. 1 shows the IDFG of a second-order digital filter section. The IDFG is by definition homogeneous, and therefore it cannot directly model multiple sampling rates. However, a general synchronous DFG can always be expressed as a simpler homogeneous graph; a systematic method for performing this is given in [15].

III. The Scheduling Problem

The IDFG and the *set of hardware resources* provide the model for the scheduling problem. For reasons of simplicity the set of hardware resources is assumed to be a set of identical processors.

The (processor) scheduling of an IDFG consists in finding a suitable *periodic ordering* of the computational nodes $c \in C$, and their assignment to the set of processors $\boldsymbol{P}$, such that the precedence relationships are not violated, and a certain objective function is optimized. The objective function can take into account one or more quality factors, as the number of hardware resources, iteration period, register life time, latency, etc. The set $\boldsymbol{P}$ will be assumed to be a set of identical processors. A formal proof of the NP-completeness of IDFG's is given in appendix A.

Two optimization methods are considered here.

1) Given a specific iteration period, minimize the number of required processors. This problem will be referred to as *fixed-rate scheduling.*
2) Given a finite set of processors, minimize the iteration period. This problem will be referred to as *maximum-throughput scheduling with limited resources.*

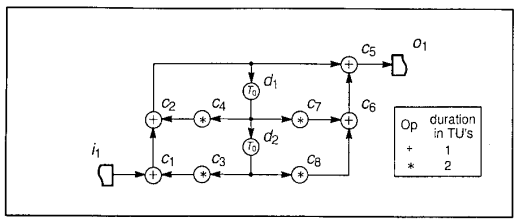


Fig. 1. IDFG of a second-order digital filter section. The duration is given in an abstract time unit, TU.

The assignment of every computational node $c \in C$, to one or more consecutive (repetitive) time slots without considering the processor assignment will be called a *time schedule.*

Even when the iteration period is T_0, usually more than one period is covered by the time schedule. In order to take into account the effect of the overlapping in time of the execution of different iterations due to the periodic-in-T_0 schedule, the execution of one iteration (time schedule) is folded modulo T_0 into the range $[0, T_0 - 1]$. In this way, the operations are grouped in T_0 *equivalence classes* [5], [11]. The operations in each equivalence class are executed in parallel.

When both the iteration period and the number of resources are fixed, as is done in [9], a solution for the scheduling problem may not exist (see Section V). In order to guarantee a solution, either of the two parameters should be left free to be optimized by the algorithm.

Interprocessor communication delays are first assumed to be negligible with respect to the computation time. The implications of communication delays are briefly discussed in Section XI.

IV. The Scheduling-Range Chart

The time interval where an operation can start its execution in a valid time schedule will be called its *scheduling range.* The length of this interval is called the *mobility* of the operation. The *scheduling-range chart* displays this information for every operation in the iterative or noniterative DFG. The concept of the scheduling range will first be explained for noniterative DFG's and will later be extended to IDFG's.

4.1. Noniterative DFG's

In noniterative DFG's it is often the case that the scheduling goal consists in the minimization of the schedule length (makespan). The minimum schedule length is equal to the critical path of the graph. The scheduling range of each operation is *finite* and determined by its positions in the early and late schedule [16], [17]. For an operation in the critical path, its early and late-schedule positions are the same, and therefore its mobility is zero.

These concepts can be visualized with the aid of an example. Fig. 2 shows a noniterative DFG, its critical path, and the Gantt charts of an early and a late schedule. In order to ease the readability of the schedules, the subindex i is used to refer to the computational node $c_i \in C$.

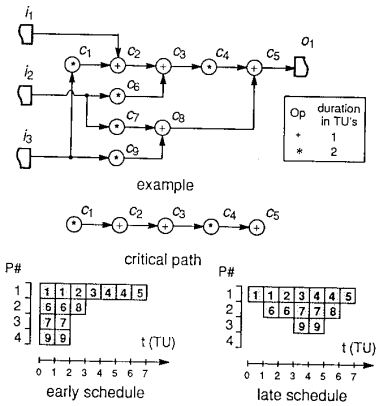


Fig. 2. Some concepts in the scheduling of a noniterative DFG.

It is also possible to represent the time schedule of a noniterative DFG by a time-schedule chart, as shown in Fig. 3 for the early and late schedules. The scheduling-range chart for noniterative DFG's is determined by merging the information of the early and late time-schedule charts into a single chart.

Various scheduling algorithms for noniterative DFG's based on the idea of determining the scheduling ranges of the operations have been reported. Early examples are PERT [18], the scheduling strategies proposed by Ramamoorthy *et al.* in [16], and the critical path algorithm for microcode compaction[2] proposed by Ramamoorthy and Tsushita [19]. Another example is the recently proposed force-directed scheduling [17] for behavioral synthesis of ASIC's, which has been reported as very successful.

4.2. IDFG's

In noniterative DFG's, the schedule is nonperiodic and the scheduling ranges take the beginning of the schedule as a reference. In contrast, in an IDFG, the schedule is periodic. To determine the scheduling ranges it is necessary to define a periodic reference. We choose the periodic reference to be the (periodic) time assignment of a *reference node* $c_r \in C$.

In contrast to the case of noniterative DFG's where the scheduling ranges are finite, in IDFG's the ranges can be finite or infinite. Each strongly connected component, considered as a whole, has an infinite scheduling range with respect to another one. The nodes of a multinode strongly connected component have finite scheduling ranges with respect to a reference operation inside that component.

When all the predecessors or successors of an operation have been time scheduled, we say that its scheduling range has a *fixed left limit* or a *fixed right limit,* respectively.

The extension of the notation for IDFG's is shown in Fig. 4.

To build the scheduling-range chart for an IDFG, an operation is selected as a reference. The scheduling range of

[2] Do not confuse with the "critical path list scheduling" algorithm.

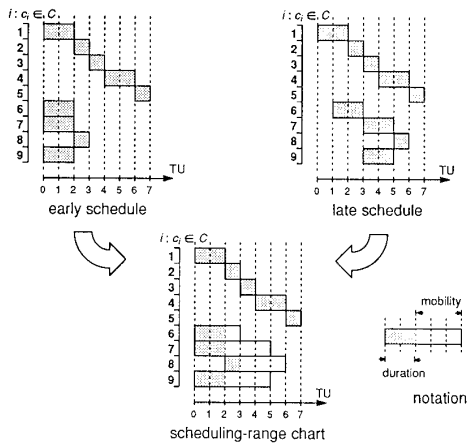


Fig. 3. Time-schedule charts and scheduling-range chart for the noniterative DFG of Fig. 2.

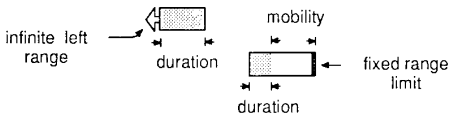


Fig. 4. Notation for the scheduling-range chart of IDFG's.

the rest of the operations in the graph is determined by (the joint effects of) the following constraints.

- Forward precedence relation (see Fig. 5): Any pair of operations $c_i, c_j \in C, (c_i, c_j) \in E$, should be scheduled such that:

$$t_j \geq t_i + c_i.duration \quad (1)$$

where t_k is the time when operation c_k starts its execution.

- Backward precedence relation (see Fig. 6): This constraint expresses precedence relations between operations of different iterations. Two operations $c_i, c_j \in C$, connected by a path R of delay nodes should be scheduled such that the token produced by c_i and delayed by the delay elements in the path is available at c_j for the execution of the corresponding iteration. Expressed in another way:

$$t_j + T_0 \sum_{d \in vset(R)} d.multiplicity \geq t_i + c_i.duration \quad (2)$$

where $vset(R)$ gives the set of vertices contained in path R.

- Slack time (see Fig. 7): The slack time of a loop L is defined by

$$\text{slack time} = T_0 \sum_{d \in vset(L)} d.multiplicity - \sum_{c \in vset(L)} c.duration. \quad (3)$$

The slack time in loops is a consequence of the forward and backward precedence relations between nodes in a (multinode) strongly connected component [10].

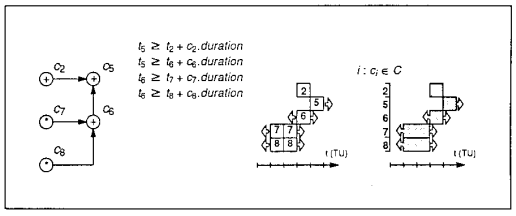


Fig. 5. Forward precedence constraints.

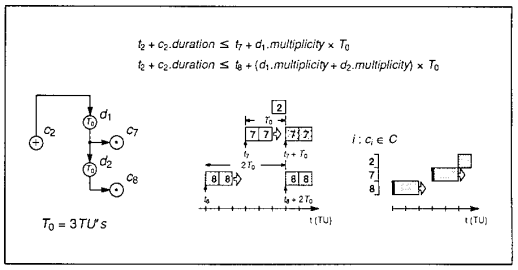


Fig. 6. Backward precedence constraints for $T_0 = T_{0_{min}} = 3$ TU's.

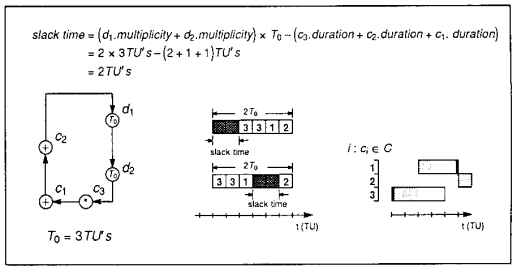


Fig. 7. Scheduling range of operations in strongly connected components, for $T_0 = T_{0_{min}} = 3$ TU's.

Fig. 8 shows the scheduling-range chart for the IDFG of Fig. 1, for $T_0 = T_{0_{min}} = 3$ TU's. Node c_2, member of the strongly connected component $\{c_1, c_2, c_3, c_4, d_1, d_2\}$ has been taken as reference. Computational node c_4, as well as reference operation c_2, are members of the critical loop. Therefore, the slack-time in the directed loop $d_1 - c_4 - c_2 - d_1$ is zero and the mobility of c_4 is zero. Operations c_1 and c_3 are members of the noncritical loop $d_1 - d_2 - c_3 - c_1 - c_2 - d_1$; their mobilities are equal to the slack time of that loop, i.e., 2 TU's. The rest of the operations in the IDFG have infinite scheduling range. Since operations c_2 and c_4 have a fixed position in the chart, i.e., they are time scheduled, the scheduling range has a fixed limit for operations c_3, c_7, and c_8 (whose only predecessor is c_2), and c_1 (whose only successor is c_2).

V. Scheduling-Problem Bounds

This section addresses the processor and iteration-period bounds of the scheduling problem. It will be shown that a solution to the scheduling problem might not exist when both the number of processors and the iteration period are fixed to those values.

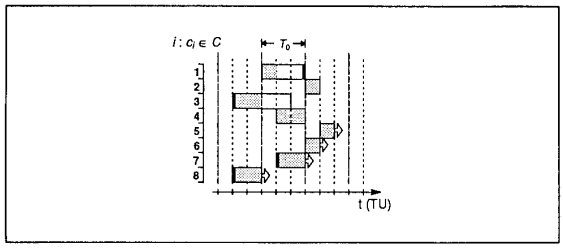


Fig. 8. Scheduling range chart for the IDFG of Fig. 1, $T_0 = T_{0_{\min}} = 3$ TU's.

Processor Bound for a Fixed Sampling Rate

The *computational Processor bound for an iteration period* T_0, $P_{0_{T_0}}$ is defined as [5]:

$$P_{0_{T_0}} = \left\lceil \frac{\sum_{c \in C} c.duration}{T_0} \right\rceil. \quad (4)$$

Iteration Period Bounds

- *Iteration period bound of the IDFG for unlimited resources,* $T_{0_{\min}}$: In *cyclic IDFG's,* the iteration period can be reduced, by supplying enough computing power, until the limit imposed by the *critical loop* [12], L_c. A critical loop is that directed loop for which the summation of the processing time of each operation in the loop divided by the number of delay elements in the loop is maximal. This quotient is $T_{0_{\min}}$. This bound is given by

$$T_{0_{min}} = \frac{\sum_{c \in vset(L_c)} c.duration}{\sum_{d \in vset(L_c)} d.multiplicity}. \quad (5)$$

  In *acyclic IDFG's,* by supplying enough computing power, the iteration period for a fully static unblocked schedule can be reduced until the duration of the longest operation. So, for the case of acyclic IDFG's, we define $T_{0_{\min}} = \max_{c \in C}(\text{c.duration})$. By the use of cyclo-static techniques [5] or direct blocking and with *unlimited resources,* the iteration period can be reduced below that limit [10].
- *Computational iteration period bound for a number P of processors,* T_{0_P}: This bound takes into consideration the number of processors P, and the total computational delay of the IDFG. The topology of the graph is not considered. This bound is defined as

$$T_{0_P} = \left\lceil \frac{\sum_{c \in C} c.\ duration}{P} \right\rceil. \quad (6)$$

- *Iteration period bound of the IDFG for a number P of processors,* $T_{\min_P}$: This bound considers both a fixed number of processors and the topology of the graph. It is given by:

$$T_{\min_P} = \max\left(T_{0_{\min}}, T_{0_P}\right). \quad (7)$$

When using the above mentioned bounds to find a solution for the multiprocessor scheduling problem, the following points have to be considered.

- *Not every IDFG has a solution to the scheduling problem when both the iteration period and the number of resources are fixed.* See the IDFG example of Fig. 9; $T_{0_{\min}} = 4$ TU's and $P_{0_4} = 2$. The initial state of the scheduling-range chart shows that the precedence constraints are such that operation c_7 is the only one with mobility. The remaining ones have a fixed position. There are only two possible class assignments for operation c_7 (its duration is 2 TU's, and due to the nonpreemption condition it should be assigned to two consecutive classes): (0–1) and (1–2). However, only classes (2–3) are available. A solution with P_{0_4} in $T_{0_{\min}}$ does not exist; either the number of processors or the iteration period has to be increased.
- *A fully static unblocked processor assignment may not exist when both the iteration period and the number of processors are fixed, even when a valid equivalence-class assignment (periodic time schedule) does exist.* Consider the IDFG of Fig. 10. This graph has $T_{0_{\min}} = 3$ and $P_{0_3} = 2$. The three operations are members of the critical loop. The corresponding scheduling-range chart and equivalence-class assignment, after the equivalence-class assignment phase, are shown in the figure. However, it is not possible to assign the operations to only two processors without the use of direct blocking [9], [11], or cyclo-static techniques [5], [9]. A fully static unblocked solution will require to increase either the number of processors or the iteration period.
- *The use of a heuristic cannot guarantee an optimal solution.* The iterative scheduling problem is NP-complete (a formal proof is given in Appendix A). Therefore, the use of heuristics, as those proposed in this paper, in finding a schedule in polynomial time, cannot guarantee an optimal solution.

VI. The Scheduling Algorithms

The scheduling-range chart concept provides an alternative approach for the scheduling of IDFG's. Range-chart-guided techniques are based on finding an optimal position within the scheduling range of each operation in such a way that some quality criteria (number of hardware resources, iteration period, latency, register life time) are optimized.

Here, two straightforward and greedy algorithms are presented:

- *fixed-rate scheduling,* where the iteration period is fixed and the algorithm optimizes the number of hardware resources; and
- *maximum-throughput scheduling,* where the number of resources is fixed and the algorithm minimizes the iteration period.

6.1. Fixed-Rate Scheduling for Minimization of Hardware Resources

The algorithm described in this section can be applied to cyclic as well as acyclic IDFG's. For rate-optimal schedules

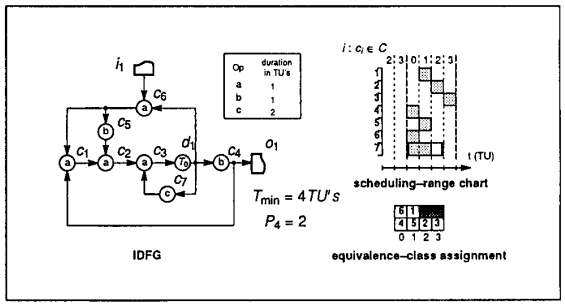


Fig. 9. IDFG with no solution to the scheduling problem for $T_0 = T_{0_{min}}$, and $P = P_{T_{0min}}$.

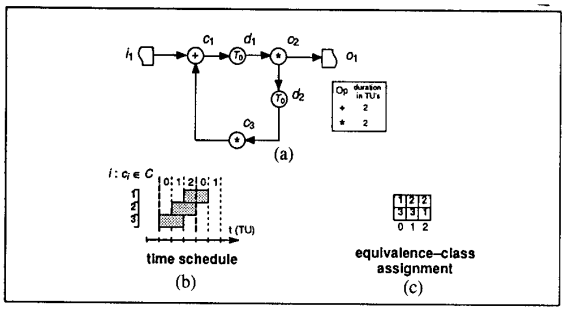


Fig. 10. IDFG with no fully static unblocked solution to the scheduling problem for $T_0 = T_{0_{min}}$ and $P = P_{T_{0min}}$.

of cyclic IDFG's, the iteration period is equal to $T_{0_{min}}$, the iteration period bound.

The T_0 equivalence classes are divided into levels; one processor per level is necessary. The algorithm assigns operations to equivalence classes and time slots according to a priority determined by their mobilities, and in such a way that *the number of levels is optimized.* Once this has been performed for all the operations, the algorithm enters a new phase where the operations are assigned to processors.

When the duration of all the computational nodes $c_i \in C$ is the same, the processor assignment phase does not need to be delayed. A different processor can be associated to each level, and the processor assignment is performed simultaneously to the equivalence-class assignment. When the duration of the computational nodes is not the same, the processor assignment problem on its own is NP-hard. After the equivalence-class assignment phase, each operation c_i is assigned to one or more consecutive classes. The contiguous equivalence classes to which each operation has been assigned to constitute a circular interval on a circle of circumference equal to T_0. The problem of optimal processor assignment consists in determining a partition of C into m pairwise disjoint subsets such that there is no overlapping between the circular intervals corresponding to nodes of the same subset, and m is minimized. The circular intervals can be used as the vertex set of a *circular-arc graph* $G(C, E)$, where there is an edge $(c_i, c_j) \in E$ if and only if the corresponding intervals have a non-empty intersection. The optimal processor assignment problem then becomes equivalent to finding a minimum coloring of the corresponding circular-arc graph. In [20] it was shown that this problem is NP-hard. The determination of processor assignment in the range-chart guided algorithms proposed here is performed by a heuristic.

A description of the fixed-rate scheduling algorithm is given below.

1) For an acyclic IDFG provide T_0. For a cyclic IDFG determine the critical loop and $T_{0_{min}}$. Provide an iteration period such that $T_0 \geq T_{0_{min}}$ (make $T_0 = T_{0_{min}}$ for rate-optimal schedules).
2) Select a reference operation and determine the scheduling-range chart for T_0.
3) Partition the scheduling-range chart into sectors that correspond to each equivalence class.
4) Maintain for each equivalence class a pointer to the *first available level*; initially this pointer has a value 1 for all equivalence classes.
5) Select the unscheduled operation with the shortest scheduling range. In case of equal length, give priority to an operation with a fixed-limit range. Select the operation c_i with the lowest index i if multiple operations have a range with fixed limit.
6) Fix the position of the selected operation within its scheduling range, such that it can be assigned to those equivalence classes with the lowest level. When multiple positions are possible, give a preference to the position closest to the fixed limit. Since the level of a class is not associated to any processor, operations that cover more than one class can be assigned to different levels. Update the first available level of the affected equivalence classes.
7) Update the scheduling-range chart to incorporate the fixed position of the operation just scheduled.
8) Repeat Steps 5, 6, and 7 until all operations have been scheduled.
9) Assign operations to processors.
   (a) Sort operations according to their computational delay; the longest first; in case of equal duration sort by operation index i.
   (b) Maintain for each equivalence class a pointer to its first available level which now is associated to a processor. This pointer should be initialized to 1 for all equivalence classes.
   (c) Remove the first operation from the list and assign it to the first processor level that has empty all the equivalence classes to which the operation has been assigned to. Update the first-available-level pointers.
   (d) Repeat the last two steps until all the operations have been assigned to processors.

As an example, consider the rate-optimal scheduling of the IDFG of Fig. 1. First, the critical loop is determined, and $T_{0_{min}}$ is computed (see Fig. 11(a)). Next, operation c_2 is selected as a reference and the scheduling-range chart is determined for $T_0 = T_{0_{min}} = 3$ TU's; the chart is divided into sectors corresponding to the three equivalence classes, 0, 1, and 2 (see Fig. 11(b)). Then, the algorithm enters the

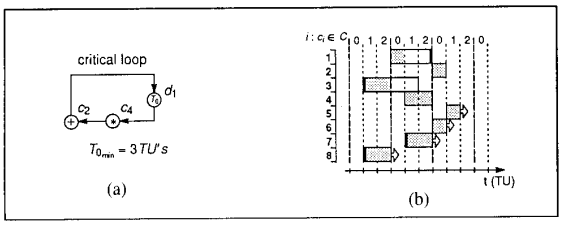


Fig. 11. Fixed-rate scheduling algorithm. (a) Critical loop and scheduling-range chart for $T_0 = T_{0_{min}}$ with (b) equivalence-class sectors.

equivalence class assignment phase, shown in Fig. 12. Operations c_2 and c_4, which do not have mobility, are automatically assigned to classes 0, and 1 and 2, respectively, in level 1 (Fig. 12(a)). The operation selected next is c_1. There are no more empty levels, so a new level is created. The operation is scheduled giving preference to the position where the fixed limit is, filling level 2 of class 2; the chart is updated (Fig. 12(b)). The algorithm continues with operation c_3. Of the three possible positions, (1–2), (2–0), and (0–1), the algorithm chooses the first in order not to create a new level (Fig. 12(c)). The algorithm continues in this way until all the operations have been time scheduled and assigned to equivalence classes (Fig. 12(g)).

The processor assignment phase is shown in Fig. 13(a); the processor utilization is 100%. Fig. 13(b) visualizes the overlapping in the execution of four consecutive iterations.

6.2. Maximum-Throughput Scheduling with Limited Resources

This algorithm works essentially in the same way as the fixed-rate scheduling algorithm introduced in the previous subsection; the modifications are given below (the steps that are not mentioned are not modified).

1) Provide P, the number of processors; set $T_0 = T_{\min_P}$.
6) As Step 6 in fixed-rate scheduling. However, if the first available level for any equivalence class exceeds $P + 1$, set $T_0 := T_0 + 1$ and restart from Step 2.
9(c) As Step 9(c) in fixed-rate scheduling. However, if the number of processors after assigning the operation exceeds P, set $T_0 := T_0 + 1$ and restart from Step 2.

An alternative approach that avoids restarting the *iteration period every time* is adjusted is discussed in [21].

VII. Constructing the Scheduling-Range Chart

Given the set of all inequalities derived from the forward and backward precedence relations, the goal is to find the scheduling range of each operation. The following remarks should be made at this stage.

- A unique solution exists only when the starting time of one operation is fixed. This operation is called the reference operation, c_r. By definition $t_r = 0$.
- Given the reference operation, none of the operations has an empty scheduling range. This is a consequence of the choice $T_0 \geq T_{0_{min}}$.

To our knowledge, the best method for computing the scheduling ranges for each operation is to represent the set of inequalities in a so-called *inequality graph* and to compute *longest paths* in these graphs. The inequality graph is a weighted directed graph $G(C, F)$ that represents the forward and backward precedence relations between the computational nodes in an IDFG. C is the set of computational nodes and F is the set of edges. Each edge $f_k \in F$ has a weight $w(f_k)$.

An edge $f_k = (c_i, c_j) \in F$ expresses the following relation:

$$t_j - t_i \geq w(f_k). \tag{8}$$

We can rewrite (1) and (2) as

$$t_j - t_i \geq c_i.duration \tag{9}$$

$$t_j - t_i \geq -T_0 \sum_{d \in u set(R)} d.multiplicity + c_i.duration. \tag{10}$$

Equations (9) and (10) have now the same form as (8) and can be used for constructing $G(C, F)$. Fig. 14 shows the inequality graph corresponding to the IDFG of the second-order filter section of Fig. 1.

In the inequality graph defined in this way, the lower bound of the scheduling range of a computational node c_i is given by the longest path from the reference node c_r to c_i. Because $t_r = 0$, the lower bound of t_i is exactly the length of longest path. If there is no path, there is no lower bound; so $t_i \geq -\infty$. Similarly, the upper bound is found by computing the longest path from c_i to c_r *multiplied by* -1.

The longest paths from the reference node to all other nodes have to be computed for the lower bounds. This is the *single-source longest path problem.* It can be solved by the *Bellman–Ford algorithm* (see, e.g., [20]). Although the original version of this algorithm is for shortest paths, it can easily be adapted for longest-path computations. The graph on which the algorithm operates can be cyclic, but should not have any directed loops where the total edge weight is positive (so-called positive loops). The inequality graph satisfies this restriction, since $T_0 \geq T_{0_{min}}$.

A concise description of the Bellman–Ford algorithm applied to our problem is given below (it has been adapted from [23]):

1) Associate a variable l_i with each node c_i (this variable will store the length of the longest path from c_r to c_i, when the algorithm terminates); initialized $l_r = 0$ and $l_i = -\infty$ for $i \neq r$.
2) Use two variables S_1 and S_2, that will store sets of nodes; initialized $S_1 = \{c_r\}$ and $S_2 = \emptyset$.
3) Consider all edges $f_k = (c_i, c_j)$ such that $c_i \in S_1$; if $l_j < l_i + w(f_k)$ then set $l_j = l_i + w(f_k)$ and add c_j to S_2.
4) If $S_2 \neq \emptyset$, set $S_1 = S_2$, $S_2 = \emptyset$ and continue with Step 3.

For the upper bounds of the scheduling ranges, the longest path from all nodes c_i to c_r has to be computed. This can be done by applying the Bellman–Ford algorithm in a graph $G'(C, F')$, which is obtained from $G(C, F)$ by reversing the

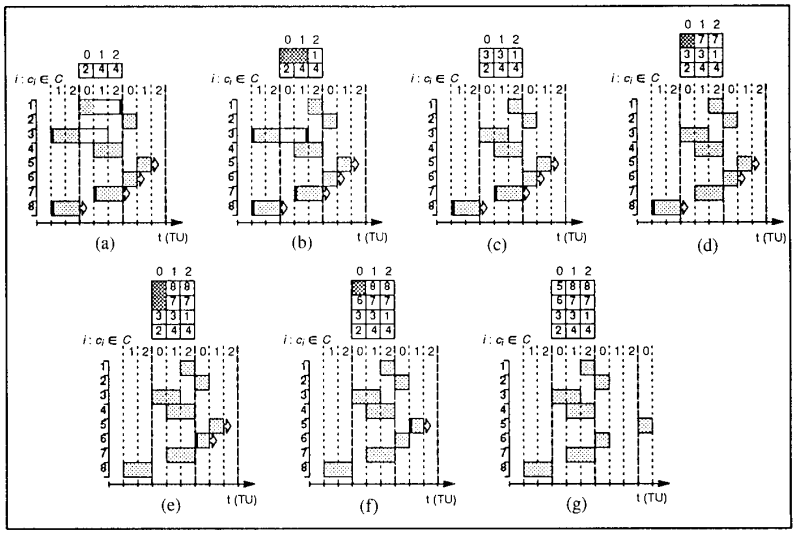


Fig. 12. Fixed-rate scheduling algorithm. Sequential steps of the time-scheduling and equivalence-class assignment, (a) to (g).

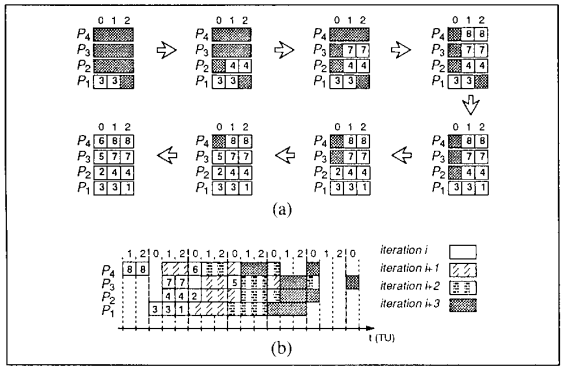


Fig. 13. (a) Processor-assignment phase. (b) Processor execution of four consecutive iterations.

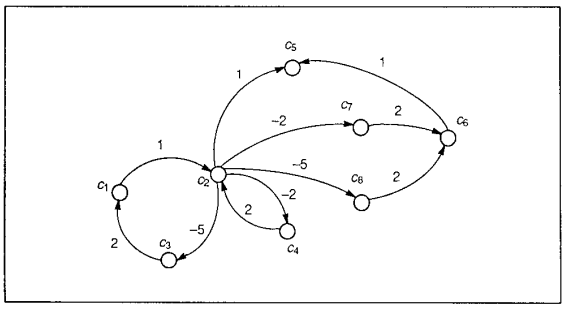


Fig. 14. The inequality graph for the IDFG of the second-order filter example.

orientations of all edges and multiplying the lengths found by -1.

The time complexity of the Bellman–Ford algorithm is $\boldsymbol{O}(cf)$, where c and f are the cardinalities of the sets C and F, respectively. One by one, the scheduling algorithms fix the operations c_i at a time t_i somewhere within their scheduling range. This can be represented in the graph $G(C, F)$ by adding an edge with weight t_i from c_r to c_i and an edge with weight $-t_i$ from c_i to c_r. Fixing the operation affects the ranges of the unscheduled operations; they can be updated by the Bellman–Ford algorithm. For updating, the old values of the lower (upper) bounds can be taken as initial values, and Step 1 in the algorithm above can be skipped. This leads to an improvement in the average execution time, but the worst-case time complexity remains the same. As the range chart has to be updated c times until all operations have been scheduled, the contribution of the range-chart calculations to the overall time complexity of the scheduling algorithms is $\boldsymbol{O}(c^2 f)$.

VIII. Time-Complexity Issues

In order to determine the overall time complexity of the algorithms, the following main steps have to be considered.

1) *The computation of* $T_{0_{\min}}$. This value is necessary for any scheduling algorithm for cyclic IDFG's, not only the one presented in this paper. In [10], an efficient algorithm is presented that does not enumerate all loops (see also [24]). It is based on the *minimum cost-to-time ratio cycle* algorithm in [25]. It has a time complexity of $\boldsymbol{O}(de + d^3 \log d)$, where d is the number of delay elements and e is the number of edges in the IDFG.
2) *The generation of the inequality graph G(C, F) from the IDFG(V, E).* Edges in F originating from forward precedences are found by inspecting every edge E once. Edges originating from backward precedences can be found by means of a depth-first search starting from the computational nodes incident to delay elements; this can also be done in $\boldsymbol{O}(e)$ time. Note that f, the cardinality of F, is $\boldsymbol{O}(e)$.
3) *The repeated computation of the scheduling-range*

chart. As explained in Section VII, this can be done in $O(c^2 f)$ where c is the cardinality of C.

4) *The search of the next operation to be scheduled and its scheduling.* The best candidate to schedule during each iteration can be found by a linear search through all unscheduled operations and therefore requires $O(c)$ time. For equivalence-class assignment, at most $T_{0_{\min}}$ locations have to be considered for each operation. For processor assignment the search space is two dimensional and at most $T_{0_{\min}} \times P$ positions have to be considered. So, the time complexity of the search and scheduling is $O(c^2 + cPT_0)$.

The IDFG's considered here have computational nodes with a number of inputs bounded by a constant (e.g., two-input additions). Besides, it is assumed that the number of retries in maximum-throughput scheduling does not depend on the problem size. This means that f and e are $O(c)$. Combining all these considerations we get the following overall time complexity for the algorithms:

$$O\left(cd + d^3 \log d + c^3 + cPT_0\right).$$

IX. Comparison with Other Scheduling Algorithms

This section compares the introduced algorithms with other scheduling algorithms for IDFG's and analyzes the differences. Other scheduling methods reported in the literature can be grouped in the following categories.

- *Single iteration methods.* These methods optimize schedules for a single iteration of the algorithm. The IDFG is converted into a noniterative DFG that corresponds to the precedence graph of one iteration by replacing delay nodes by input and output nodes. Deterministic scheduling of noniterative DFG's is a well-studied problem [7]: the resulting acyclic graph can be scheduled by any algorithm for minimization of the schedule length of a noniterative DFG, as, for example, critical-path list scheduling [8].
- *Direct blocking methods* [9], [11]. These methods are basically the same as above, but here the schedule is optimized for the precedence graph over a number of iterations equal to the blocking factor $n (n \geq 1)$, capturing more parallelism in general.
- *Fixed-rate methods.* These techniques are able to generate schedules for a given iteration period $T_0 \geq T_{0_{\min}}$. In the special case that $T_0 = T_{0_{\min}}$, the schedules are called rate-optimal.

  Maximum-spanning tree. This scheduling technique has been proposed by Renfors and Neuvo in [12] and [26]. It is based on the scheduling of an *essentially equivalent* network N'[27], obtained from the original network N (where the operations have delay zero) by inserting delays d_i equal to the processing time t_i of the operations, removing the original ideal delay elements, and inserting *shimming delays* when necessary.

  Optimum unfolding. This method, proposed by Parhi [1], consists of reducing the graph to an equivalent perfect-rate data-flow program that can always be scheduled rate optimally. This is performed by unfolding with an optimal unfolding factor, and scheduling the resulting graph.

  Search of cyclo-static schedules. The theory of cyclo-static processor schedules was developed by Schwartz ([5], [9]). A *cyclo-static* system is a synchronous multiprocessor system whose schedule is characterized by its periodicity, in the discrete time-processor space, $P \times T$. The approach proposed by Schwartz to find cyclo-static schedules is by means of a depth-first search in the lattice $P \times T$ with iteration period equal to $T_{0_{\min}}$ (rate-optimal solution) and a number of processors equal the *processor bound* (Equation (4) for $T_0 = T_{0_{\min}}$).

Table I summarizes some of the major characteristics of the methods mentioned above and the proposed range-chart algorithms. The schedules obtained by means of single-iteration methods give, in general, poor results in number of processors and iteration period due to the fact that all possible inter-iteration parallelism is disregarded. The efficiency increases when the graph is blocked since more parallelism can be captured. However, as the number of computational nodes in the direct blocked graph increases proportionally with the blocking factor n, so does the memory space necessary to store the schedule. The iteration period of the fixed-rate methods, especially rate-optimal methods, is by definition optimal. The maximum-spanning tree method does not explicitly try to optimize the number of resources, leading in general to schedules that need more processors than optimal. With optimal unfolding, the number of processors can be optimized by using a suitable heuristic, but the memory space necessary to store the program increases proportionally with the unfolding factor. A fixed-rate cyclo-static schedule with a number of processors equal to the processor bound for the selected iteration period does not always exist; therefore, the method based on a search in that time-processor space cannot guarantee a solution. The two proposed algorithms based on the range-chart guarantee a solution with minimal memory space; in fixed-rate scheduling the iteration period is fixed and the algorithm optimizes the number of processors, while in maximum-throughput scheduling, the number of resources are fixed and the algorithm minimizes the iteration period.

X. Experimental Results

The two algorithms described in Section VI have been implemented in Common Lisp. The implementation actually treats fixed-rate scheduling as a special case of maximal-throughput scheduling, where the limit on the number of processors is set to infinity.

In this section, some experimental results obtained by this implementation are described. We have run the algorithm for several benchmark graphs: the second-order filter section of

TABLE I
COMPARISON OF SEVERAL SCHEDULING ALGORITHMS FOR IDFG'S

		Number of processors	Iteration period	Memory space	Guarantee of solution
Single-iteration methods		poor	poor	minimal	yes
Direct Blocking		efficiency increases with the blocking factor		increases with the blocking factor	yes
Maximum-spanning tree		not optimized	optimal	minimal	yes
Optimum Unfolding		optimized	optimal	increases with the unifolding factor	yes
Search of cyclo-static schedules		optimal	optimal	minimal	no
Scheduling-range chart	rate optimal	optimized	optimal	minimal	yes
	maximum throughput	fixed	optimized		

Fig. 1, a fourth-order Jaumann wave digital filter (mentioned in [12]; see Fig. 15), a fourth-order all-pole lattice filter (mentioned in [26]; see Fig. 16), a 16-point FIR filter (introduced in [28] and often used as a benchmark in high-level synthesis; see Fig. 17), and a fifth-order wave digital elliptic filter (introduced in [29] and also often used as a benchmark in high-level synthesis; see Fig. 18).

The results are shown in Table II. For each benchmark the minimal number of processors for different iteration periods is displayed. Besides, the table shows the computational processor bound, the processor utilization, the duration of addition and multiplication for the benchmark (original data are used as much as possible), and the CPU time for the execution of the algorithm on an Apollo DN 10000 machine. From the table it can be concluded that the algorithm is able to find solutions with a number of processors equal to the computational processor bound, except for the fifth-order elliptic filter when $T_0 = 16$ and $T_0 = 21$. An analysis of these cases based on scheduling ranges shows that no solutions exist for $P = 3$, respectively, $P = 2$. This can be seen by considering a suitable interval in the initial range chart (folded by modulo T_0) and showing that even when as many operations as possible are scheduled outside this interval, at least one equivalence class should have 4, respectively, 3, levels because the total computation time of the operations remaining in the interval exceeds 3, respectively, 2, times the length of the interval.

It should be noted that the results are not independent of the reference operation and that, therefore, the reference operation should be carefully chosen.

Our implementation includes a program that automatically generates schedules in PostScript format. Two examples are presented in Figs. 19 and 20. In these figures, a crossed rectangle indicates an idle processor and a shaded rectangle refers to the execution of an operation belonging to iteration other than the depicted one.

As can be seen from the table, the program executes very fast. The CPU-times given refer to the computation of $T_{0_{\min}}$ and the schedule. A good user interface will need an order of magnitude more of CPU time than the actual scheduling; e.g., the generation of the PostScript description for the fifth-order elliptic filter already requires 2.0 s.

XI. COMMUNICATION DELAYS

The scheduling model used in the previous sections considers communication delays negligible with respect to the processing times. The inclusion of interprocessor communication delays increases the difficulty of the scheduling problem. In this section the main consequences will be briefly discussed. A detailed discussion can be found in [10].

When interprocessor communication delays cannot be neglected, they have to be considered in the formulation of the forward and backward precedence relations that determine the scheduling ranges.

In the general case, when extending our model in a similar way to the EDAG proposed by Hwang et al. in [30] for noniterative DFG's, these relations can be formulated as follows.

- *Forward precedence relation:*

$$t_j \geq t_i + c_i.duration + t_{\text{comm}}(P_{c_i}, P_{c_j}) \times \eta(c_i, c_j) \tag{11}$$

where $(c_i, c_j) \in E$, P_{c_i} indicates the processor to which a node $c_i \in C$ has been assigned, $t_{\text{comm}}(P_m, P_n)$ is the time necessary to transfer a message unit from processor P_m to processor P_n (usually $t_{\text{comm}}(P_m, P_n) = 0$ when $m = n$), and $\eta(c_k, c_l)$ indicates the number of messages sent from an *immediate predecessor* (through zero or more delay elements) c_k, to a successor c_l.

- *Backward precedence relation:*

$$t_j + T_0 \sum_{d \in vset(R)} d.multiplicity \geq t_i + c_i.duration + t_{comm}(P_{c_i}, P_{c_j}) \times \eta(c_i, c_j). \tag{12}$$

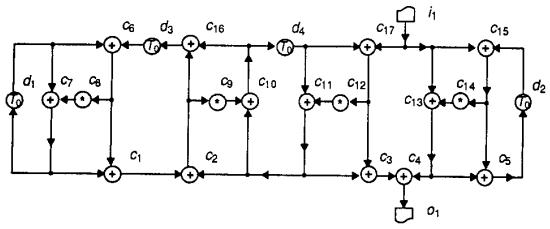


Fig. 15. The four-order Jaumann wave digital filter benchmark.

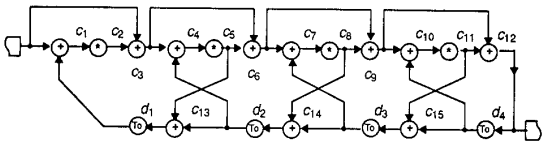


Fig. 16. The four-order all-pole lattice filter benchmark.

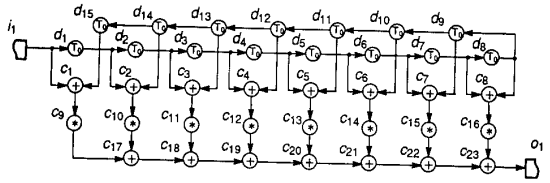


Fig. 17. The 16-point FIR filter benchmark.

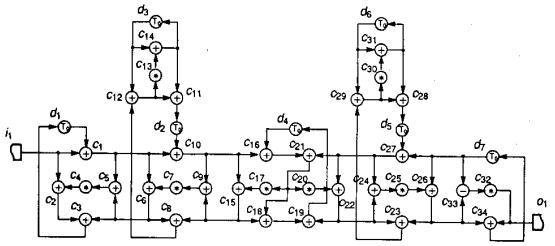


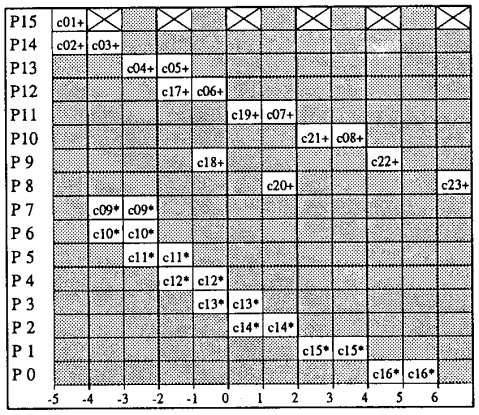
Fig. 18. The fifth-order wave digital elliptic filter benchmark.

- *Slack time in a loop L:*

$$\text{slack time} = T_0 \sum_{d \in vset(L)} d.multiplicity - \sum_{c \in vset(L)} c.duration + - \sum_{c_k, c_l \in vset(L)} t_{comm}(P_{c_k}, P_{c_l}) \times \eta(c_k, c_l) \quad (13)$$

where $\eta(c_k, c_l) = 0$ whenever c_k is not an immediate predecessor of c_l.

In order to discuss the effects of the interprocessor communication delays on the schedule, the effects on each strongly connected component (considered as a whole) in the IDFG, and on each operation of a multinode strongly connected component, will be considered separately. A processing model with I/O coprocessors is assumed.



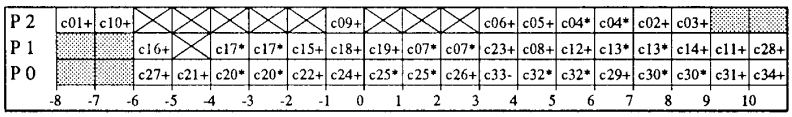
16-Point FIR Filter: T0 = 2; Ref. = c14*; Proc. Util. = 97%.

Fig. 19. The schedule found for the 16-point FIR filter for $T_0 = 2$.

TABLE II
OVERVIEW OF THE EXPERIMENTAL RESULTS (THE MINIMAL VALUES FOR T_0 HAVE BEEN SHOWN IN BOLD)

Example name	T_0	P	$P_{0_{T_0}}$	Proc. util. %	Duration +,*	CPU time seconds
Second-order section	**3**	4	4	100	1,2	0.09
	4	3	3	100		
	6	2	2	100		
	12	1	1	100		
Jaumann filter	**16**	3	3	69	1,5	0.17
	17	2	2	97		
	33	1	1	100		
All-pole Filter	**14**	3	3	74	1,5	0.13
	16	2	2	97		
	31	1	1	100		
16-point FIR filter	**2**	16	16	97	1,2	0.19
	3	11	11	94		
	4	8	8	97		
	5	7	7	89		
	6	6	6	86		
	7	5	5	89		
	8	4	4	97		
	11	3	3	94		
	16	2	2	97		
	31	1	1	100		
Fifth-order elliptic filter	**16**	4	3	66	1,2	0.33
	17	3	3	82		
	21	3	2	67		
	22	2	2	95		
	42	1	1	100		

Given an arbitrary IDFG that might contain directed loops, it can be transformed into a *reduced IDFG*. The reduced graph is obtained from the original IDFG by replacing each strongly connected component by a single node [16]. A reduced graph is always acyclic. Therefore, (13) is not applicable. With the exception of the reference node, each node has an infinite scheduling range. The inclusion of communication delays due to interprocessor communication between the nodes of the reduced IDFG only affects the location of the finite limit of the ranges as can be seen by comparing (11) and (12) to (1) and (2), their counterparts for negligible communication delays.



Fifth-Order Wave Digital Elliptic Filter: T0 = 17; Ref. = c19+; Proc. Util. = 82%.

Fig. 20. The schedule found for the fifth-order elliptic filter for $T_0 = 17$.

That means that if there are enough communication links, and the total computational time of each multinode strongly connected component does not exceed the selected iteration period, the latter is not affected by interprocessor communication, only the latency. When scheduling, communication links have to be included as another resource that is periodically used. Each operation $c_i \in C$ is assigned to $c_i.duration$ equivalence classes of a processor, while each interprocessor message between nodes c_k, c_l is assigned to $t_{\text{comm}}(P_{c_k}, P_{c_l})$ equivalence classes of a communication link between P_{ck} and P_{cl}. As communication time increases, more equivalence classes of a link are necessary to transmit a message. Therefore, for an iteration period T_0, a partition/assignment (grouping of the computational nodes of the IDFG into clusters, each of them assigned to a different processor) should be found such that the following relation is satisfied:

$$\max_{lk \in LK} \left(\eta_{lk_{ij}} \times t_{\text{comm}}(P_i, P_j) \right) \leq T_0 \quad \forall P_i, P_j \in \boldsymbol{P} \quad (14)$$

where $\boldsymbol{P}$ is the set of processors, LK is the set of communication links of the system, and $\eta_{lk_{ij}}$ is the total number of messages per iteration period in lk_{ij}, the communication link between processor P_i and P_j.

In the paragraph above, interprocessor communication was restricted to take place only between strongly connected components, but not within the strongly connected component itself. In case this restriction cannot be satisfied and a multinode strongly connected component is assigned to more than one processor, the slack time of the loop is affected by the inclusion of communication delays in accordance to (13). This case can be approached by performing a partitioning/assignment for each component where interprocessor communication takes place such that the scheduling ranges are nonempty (i.e., a valid ordering of the operations is found). Next, a final partitioning/assignment that includes the remaining operations has to be found such that (14) is satisfied.

XII. Conclusions

The notion of a scheduling range for operations of IDFG's has been introduced. The scheduling-range concept is an important aid for the understanding of the scheduling problem. The knowledge of the scheduling range of the operation gives deep insight into the problem, which is of fundamental value during the scheduling process. In terms of scheduling ranges, the scheduling problem consists of determining an optimal position within the scheduling range of each operation in a such a way that some quality criteria (number of hardware resources, iteration period, latency, register life time) are optimized.

By means of the scheduling-range concept, it has been shown that when both the sampling period and the number of processors is fixed, a solution to the scheduling problem might not exist. To guarantee a solution either of the two parameters should be left free to be optimized by the algorithm.

Two scheduling algorithms based on the scheduling-range concept were proposed: one for fixed-rate scheduling where the number of processors is optimized at the same time that a specific sampling rate (for example, optimal rate) is guaranteed, and maximum-throughput scheduling with limited resources where the sampling period is optimized for a fixed number of processors. The decisions in the scheduling algorithms are guided by the information on the scheduling ranges of the operations, which can be visualized in the scheduling-range chart.

The introduced algorithms for fully static unblocked schedules, although simple, are powerful enough to solve many problems optimally. They always guarantee a solution in polynomial time by fixing one of both parameters, iteration period or number of processors, while optimizing the other. The memory space required is optimal since the schedule corresponds to only one execution of the IDFG.

Our current research in this area concentrates on improvements of the proposed algorithms and the application of range-chart methods to high-level synthesis.

Appendix

A. The NP-Completeness of IDFG Scheduling

In this section a proof of the NP-completeness [31] of the IDFG Scheduling problem will be presented. First, the problem has to be formulated as a decision problem. Then this formulation is used in the two parts required for the proof.

1) Showing that IDFG Scheduling is in the class of problems called NP.
2) Showing that a problem known to be NP-complete can be *transformed* in polynomial time to IDFG Scheduling.

IDFG Scheduling can be formulated as a decision problem as follows.

Given an iterative data-flow graph IDFG, and two integers P and T_0, does there exist a fully static unblocked schedule periodic in T_0 such that the tasks in the IDFG can be scheduled on P processors respecting the forward and backward precedence constraints implied by the IDFG? Yes or no?

For the first part of the proof it should be shown that there exists a search space for each problem instance from which a

nondeterministic choice can be made, and once an element has been chosen, it should be shown that there exists a polynomial time algorithm to decide whether this element is a solution to the problem instance. A schedule is an assignment of a processor p_i and a starting time t_i to each operation $c_i \in C$ in the IDFG. Obviously $1 \leq p_i \leq P$. The fact that t_i is bounded requires some explanation. For purposes of normalization set the starting time of the operation that is executed the earliest to zero. So, $0 \leq t_i$. Divide the time axis in intervals of length T_0 starting from 0. As for any $c_i \in C$, $c_i.duration \leq T_0$ (otherwise scheduling with iteration period T_0 is impossible) the longest schedule without unnecessary latency obeys the property that each time interval of length T_0 either contains the beginning or the end of an operation's execution (if there is an interval that does not contain the execution of any operation, there is unnecessary latency). So, $0 \leq t_i \leq 2cT_0 - 1$.

Given the bounded search space, a nondeterministic machine is able to select any element by assigning a processor and a starting time to each operation in the IDFG. The next thing to show is that the test whether such an element constitutes a solution can be done in polynomial time. Such a test should check that the t_i satisfy the forward and backward precedence constraints and it should also check that at most one task is assigned to any processor at any moment in time taking into account that one should be able to fold the schedule into the interval $[0, T_0 - 1]$ without creating overlaps. Both checks can be done in polynomial time and this means that IDFG Scheduling is in NP.

A problem that is known to be NP-complete [31] and that can be transformed in a straightforward way into IDFG Scheduling is Bin Packing. The Bin Packing problem is as follows.

> Given a finite set U of items $U = \{u_1, u_2, \cdots, u_u\}$, where the size of an element $u_i \in U$ is given by $u_i.size \in Z^+$, and two parameters B, $K \in Z^+$. Does a partition of U into disjoint sets $U_1, U_2, \cdots, U_K$ exist such that for all $j \sum_{u_i \in U_j} u_i.size \leq B$?

Given an instance of Bin Packing, its transformation to an instance of IDFG Scheduling is as follows.

- Create a computational node c_i for every item u_i, such that $c_i.duration = u_i.size$.
- Create the edges (c_i, c_{i+1}), $i = 1, 2, \cdots, u - 1$. (Note: the exact way that the edges are created is irrelevant as long as the IDFG is connected and acyclic.)
- Set $P = K$ and $T_0 = B$.

Note that all steps in the transformation can be done in polynomial time. Because the IDFG is acyclic, all operations except the one chosen as a reference have an infinite scheduling range. An infinite range means that there are no restrictions to the equivalence classes to which an operation can be assigned. So, the obtained instance of the IDFG Scheduling problem has a solution if and only if there is a solution to the instance of the Bin Packing problem.

This completes the proof of NP-completeness of IDFG Scheduling.

Acknowledgment

The authors would like to thank András Oláh for his contributions related to time complexity issues, and Ronald Peer for his assistance in implementing the algorithms.

References

[1] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Computers,* vol. 40, pp. 178–195, Feb. 1991.

[2] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Computers,* vol. C-36, pp. 24–35, Jan. 1987.

[3] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE,* vol. 75, pp. 1235–1245, Sept. 1987.

[4] S. H. Lee and T. P. Barnwell, "A toplogical sorting and loop cleansing algorithm for a constrained MIMD compiler of shift-invariant flow graphs," in *Proc. Int. Conf. Accoustics, Speech, Signal Processing,* pp. 2927–2930, 1986.

[5] D. A. Schwartz and T. P. Barnwell, "Cyclo-static multiprocessor scheduling on the optimal realization on shift-invariant flow graphs," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing,* pp. 1384–1387, 1985.

[6] K. R. Baker, *Introduction to Sequencing and Scheduling.* New York: Wiley, 1974.

[7] J. Blazewicz, "Selected topics in scheduling theory," in *Surveys in Combinatorial Optimization,* P. L. Hammer, Ed. Amsterdam: North-Holland, 1987, pp. 1–59.

[8] W. A. Kohler, "A preliminary evaluation of the critical path method for scheduling tasks on multiprocessor systems," *IEEE Trans. Computers,* vol. C-24, pp. 1235–1238, Dec. 1975.

[9] D. A. Schwartz, "Synchronous multiprocessor realizations of shift invariant flow graphs," Ph.D. dissertation, Georgia Institute of Technology, 1985.

[10] S. M. Heemstra de Groot, "Scheduling techniques for iterative data-flow graphs, an approach based on the range chart," Ph.D. dissertation, Univ. Twente, Faculty of Electrical Engineering, Dec. 1990.

[11] S. M. Heemstra de Groot and O. E. Herrmann, "Evaluation of some multiprocessor scheduling techniques of atomic operations for recursive DSP graphs," in *Proc. European Conf. Circuit Theory and Design,* pp. 400–404, Sept. 1989.

[12] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under speed constraints," *IEEE Trans. Circuits Syst.,* vol. CAS-28, pp. 196–202, Mar. 1981.

[13] S. M. Heemstra de Groot and O. E. Herrmann, "Rate-optimal scheduling of recursive DSP algorithms based on the scheduling range chart," in *Proc. IEEE Int. Symp. Circuits and Systems,* pp. 1805–1808, May 1990.

[14] A. Gibbons, *Algorithmic Graph Theory.* Cambridge, UK: Cambridge University Press, 1985.

[15] E. A. Lee, "A coupled hardware and software architecture for programmable digital signal processors," Ph.D. dissertation, Univ. California, Berkeley, 1986.

[16] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez, "Optimal scheduling strategies in a multiprocessor system," *IEEE Trans. Computers,* vol. C-21, pp. 137–146, Feb. 1972.

[17] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. Computer-Aided Design,* vol. 8, pp. 661–679, June 1989.

[18] R. I. Levin and C. A. Kirkpatrick, *Planning and Control with PERT/CPM.* New York: McGraw-Hill, 1966.

[19] D. Landskov, S. Davidson, and B. Shriver, "Local microcode compaction techniques," *ACM Computing Surveys,* vol. 12, pp. 261–294, Sept. 1980.

[20] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, "The complexity of coloring circular arcs and chords," *SIAM J. Algebraic and Discrete Methods,* vol. 1, pp. 216–227, June 1980.

[21] S. M. Heemstra de Groot and O. E. Herrmann, "Maximum-throughput scheduling with limited resources for iterative data-flow graphs by means of the scheduling-range chart," in *Proc. Euromicro Workshop on Real Time systems,* pp. 8–16, 1990.

[22] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: MIT Press, 1990.

[23] W. L. Schiele, "On a longest path algorithm and its complexity if

applied to the layout compaction problem," in *Proc. European Conf. Circuit Theory and Design,* pp. 263–265, 1983.

[24] S. H. Gerez, S. M. Heemstra de Groot, and O. E. Herrmann, "A polynomial-time algorithm for the computation of the iteration-period bound in recursive data-flow graphs," *Tech. Rep. EL-BSC-90N060.* Univ. Twente, Faculty of Electrical Engineering, Apr. 1990. Also in *IEEE Trans. Circuits Syst. – I,* pp. 49–52, Jan. 1992.

[25] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rinehart and Winston, 1976.

[26] M. Renfors and Y. Neuvo, "Fast multiprocessor realizations of digital filters," in *Proc. Int. Conf. Acoustics Speech, Signal Processing,* pp. 916–919, 1980.

[27] A. Fettweis, "Realizability of digital filter networks," *Archiv für Elektronik und Übertragungstechnik,* vol. 30, pp. 90–96, 1976.

[28] N. Park and A. Parker, "Sehwa: A program for synthesis of pipelines," in *Proc. 23rd Design Automation Conf.,* pp. 454–460, 1986.

[29] P. Dewilde, E. Deprettere, and R. Nouta, "Parallel and pipelined VLSI implementations of signal processing algorithms," In S. Y. Kung, H. J. Whitehouse, and T. Kailath, Eds., *VLSI and Modern Signal Processing.* Englewood Cliffs, Prentice Hall, 1985, pp. 257–276.

[30] J.-J. Hwang, Y.-C. Chow, F. D. Angers, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM J. Computing,* vol. 18, pp. 244–257, Apr. 1989.

[31] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness.* San Francisco, CA: W. H. Freeman, 1979.

Sonia M. Heemstra de Groot (S'85–M'86) graduated from Mar del Plata National University, Argentina, 1983, received the master's degree in electrical engineering from the Philips International Institute of Technology, Eindhoven, The Netherlands, in 1986, and received the Ph.D degree in applied sciences from the University of Twente, The Netherlands, in 1990.

From 1986 to 1990 she worked as an assistant researcher in the group for Network Theory and VLSI Design of the Faculty of Electrical Engineering, concentrating her attention on topics related to real-time processing of DSP algorithms and efficient multiprocessor scheduling. In 1991 she became a lecturer in the group Teleinformatics and Open Systems of the same faculty. Her research includes different aspects of high-level synthesis, with emphasis on VLSI implementations of time-critical protocol functions and other telecommunication applications.

Sabih H. Gerez received the M.Sc. degree in electrical engineering and the Ph.D. degree in applied sciences from the University of Twente in 1984 and 1989, respectively.

He has worked as an assistant researcher in the group for Network Theory and VLSI Design of the Faculty of Electrical Engineering at the University of Twente from 1984 to 1989, concentrating on automatic routing algorithms and hardware description languages. In 1990 he became a lecturer in the same group. His main research interest is design automation for VLSI, with an emphasis in layout design and high-level synthesis.

Otto E. Herrmann (A'72) received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering from the University of Technology Aachen, FRG, in 1959 and 1965, respectively. In 1971 he received the "venia legendi" for telecommunication from the University of Erlangen, FRG.

From 1959 to 1965 he was a research associate and lecturer at the Universities of Aachen and Karlsruhe. From 1966 to 1972 he was senior staff member and from 1972 to 1975 "Abteilungsvorsteher" in the Department of Electrical Engineering of the University of Erlangen. During 1972 he was a visiting scientist on leave at Bell Laboratories, Murray Hill, NJ. Since 1975, he has been a full professor at the Faculty of Electrical Engineering at the University of Twente, Enschede, The Netherlands, heading the professional group for network theory, signal processing, and CACSD.