Multidimensional Periodic Scheduling: A Solution Approach

Wim F.J. Verhaegh[1] Paul E.R. Lippens[1] Emile H.L. Aarts[1,2] Jef L. van Meerbergen[1]

[1] Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
[2] Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

We present a solution approach to the multidimensional periodic scheduling problem. We introduce the concept of multidimensional periodic operations in order to cope with problems originating from loop hierarchies and explicit timing requirements. We present an iterative algorithm for the scheduling problem, based on an ILP approach for checking the constraints, and we show some experimental results. Finally, we extend the solution approach to handle parametric descriptions.

1 Introduction

The design of a digital signal processing (DSP) system concerns the mapping of a DSP algorithm onto (dedicated or programmable) hardware that implements it. Many high-throughput DSP algorithms consist of repetitive executions of operations, resulting in repetitive production and consumption of data. This can be described by nested loops and multidimensional arrays. Such a DSP algorithm can be viewed as a repetition of executions of operations in several dimensions, each of which corresponds to one loop. A specific execution of an operation can be identified by the corresponding values of the *loop iterators*. In addition to repetitive executions, video signal processing algorithms contain strict timing requirements that constrain the rates at which input data arrive, and the rates at which output data must be produced.

To handle these characteristics, we introduce an explicit timing model containing operations that are executed periodically. The periods of each operation are given by a *period vector*, whose components denote the time between two consecutive iterations in each dimension of repetition. In the multidimensional periodic scheduling problem, we now have to determine for each operation a period vector, a start time, and a processing unit (PU) on which it is executed.

There are three sets of scheduling constraints. First, we have *timing constraints*, which bound the period vectors and start times of the operations. Secondly, we have *processing unit constraints*, which specify that at most one execution of an operation can occupy a PU at a time. Thirdly, we have *precedence constraints*, which specify that data must be produced before it is consumed.

The scheduling objective we consider is to minimize the area occupied by the hardware. In video applications, area is not only determined by processing units, but also by the size of the memories that are used and the number of them. So, a trade-off has to be made between processing units and the total memory size and bandwidth.

The multidimensional periodic scheduling problem is one of the sub-problems in the design methodology Phideo [8], which is aimed at the automated design of dedicated digital video signal processors.

1.1 Related Work

In the area of one-dimensional periodic scheduling, related work is done on mapping video signal processing algorithms onto programmable video signal processors [6] and on synchronous data flow for DSP [7]. If, in addition, all operations have the same period, related work can be found in the area of pipelined scheduling [10, 15].

The literature also presents several approaches to the problem of handling multidimensional executions with multidimensional productions and consumptions of data, however without strict periodicity and strict timing requirements. In the area of high-throughput DSP, work is done on loop transformations [4, 11], in which descriptions with loops are modified in order to obtain, for instance, more parallelism and a higher throughput. In that approach the throughput is considered an objective rather than a constraint. In [13] loop transformations are handled by a method based on placement of polytopes. Further related work, but without strict timing requirements, is done in the area of systolic array design [1] and in the area of data flow analysis for parallel program construction [2, 12].

In the multidimensional periodic scheduling problem, we consider operations to be executed repeatedly, with both multidimensional repetitions and strict periodicity [14]. The executions of the operations are considered as multidimensional repetitions, since considering all executions separately is impracticable. The explicit timing in the model, incorporated by the periodicity, facilitates constraint handling and allows an adequate cost model. Furthermore, it enables us to obtain the required parallelism rather easily.

ED&TC '97 on CD-ROM

1.2 Paper Outline

The objective of this paper is to present the multidimensional periodic scheduling problem, and a solution approach to solve it. Due to space limitations, we restrict ourselves in this paper to the case that the period vectors and the set of processing units are given. Furthermore, we restrict ourselves for the moment to finding a feasible solution instead of finding one with minimal cost. For the general problem without these restrictions, we refer to [14].

The organization is as follows. In Section 2 we formulate the multidimensional periodic scheduling problem. In Section 3 we show how the processing unit constraints and precedence constraints can be checked by means of an all-integer ILP algorithm. Based on this, Section 4 presents an iterative algorithm to find a feasible start time and PU assignment. Section 5 shows some experimental results of this algorithm on practical problem instances. Finally, in Section 6, we extend the solution approach to handle parametric descriptions.

2 Problem Formulation

In this section we discuss the multidimensional periodic scheduling problem. We do this by means of a fictive example of a video algorithm, which is given in Figure 1. For

```
    for f = 0 to ∞ period 30
    begin
        for j1 = 0 to 3 period 7
            for j2 = 0 to 5 period 1
{in}            x[f][j1][j2] = input( )
        for k1 = 0 to 3 period 7
            for k2 = 0 to 2 period 2
{mu}            y[f][k1][k2] = x[f][k1][2k2] * x[f][k1][5 − 2k2]
        for l1 = 0 to 2 period 1
{nl}        z[f][l1][−1] = 0
        for m1 = 0 to 2 period 5
            for m2 = 0 to 3 period 1
{ad}            z[f][m1][m2] = z[f][m1][m2 − 1] + y[f][m2][m1]
        for n1 = 0 to 2 period 1
{out}       = output(z[f][n1][3])
    end
```

Figure 1. Example of a video algorithm, describing repetitive operations and data dependencies. Between curly braces the names of the operations are given.

the moment, we assume that the loop bounds, the periods, and the iterators' coefficients in the array indices are constants; we return to this assumption in Section 6.

A video algorithm is represented by a signal flow graph, of which the nodes denote operations and the edges denote data dependencies. Figure 2 shows the signal flow graph corresponding to the video algorithm of Figure 1.

The operations we consider may have several input and output ports. For reasons of simplicity, we assume in this paper that the consumption of input data and the production

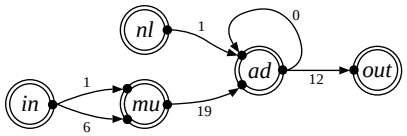

Figure 2. An abstract picture of a signal flow graph. Each operation is repeatedly executed; details of the iterations and of the data dependencies are not shown. The black dots denote the operations' input and output ports. The numbers along the edges denote weights; see Section 3.2.

of output data all take place in the same clock cycle. Input and output operations of a signal flow graph are modeled as operations without input ports and output ports, respectively. Next, we assume that the operations must be executed on dedicated processing units, i.e., we assume a one-to-one relation between operation types and PU types. In typical video applications this is a realistic assumption, since processing units perform rather complex functions, such as filtering. For simplicity, we further assume for the moment that the execution of an operation occupies a processing unit for one clock cycle.

Next, we give a formal definition of a signal flow graph, after which we exemplify its attributes by means of the video algorithm of Figure 1.

Definition 1 (signal flow graph). A signal flow graph G is given by a 6-tuple $(V, t, \boldsymbol{I}, E, \boldsymbol{A}, \boldsymbol{b})$, where

- V is a set of multidimensional periodic operations,
- $t(v)$ denotes the operation type of each operation $v \in V$,
- $\boldsymbol{I}(v) \in \mathbb{IN}_\infty^{\delta(v)}$ denotes the *iterator bound vector*, for each $v \in V$, where $\mathbb{IN}_\infty = \mathbb{IN} \cup \{\infty\}$,
- $E \subset O \times I$ is a set of directed edges representing data dependencies, where O is the set of all operations' output ports and I is the set of all operations' input ports,
- $\boldsymbol{A}(p) \in \mathbb{Z}^{\alpha(p) \times \delta(v)}$ denotes the *index matrix*, for each input or output port p of each operation v,
- $\boldsymbol{b}(p) \in \mathbb{Z}^{\alpha(p)}$ denotes the *index offset vector*, for each port $p \in O \cup I$. □

In the example of Figure 1, the set of operations is given by $V = \{in, mu, nl, ad, out\}$, each of which has its own type. Each operation v has a number $\delta(v)$ of iterators in the enclosing loops, which are combined in an *iterator vector* $\boldsymbol{i}$. For example, the iterator vector of the multiplication is given by

$$\boldsymbol{i} = \begin{bmatrix} f \\ k_1 \\ k_2 \end{bmatrix}.$$

For each operation $v \in V$, the iterator vector $\boldsymbol{i}$ is bounded between the zero vector and the iterator bound vector, i.e., $\boldsymbol{0} \leq \boldsymbol{i} \leq \boldsymbol{I}(v)$. In Figure 1, the multiplication has an iterator

bound vector

$$\boldsymbol{I}(mu) = \begin{bmatrix} \infty \\ 3 \\ 2 \end{bmatrix}.$$

We assume that only dimension 0 of an operation may have an unbounded number of repetitions; the others are finite.

The data dependencies are modeled in a signal flow graph by means of the edge set E and by describing at each output and input port the relation between the indices of the array that is used there, and the iterators of the corresponding operation. In Figure 1 we have e.g. a 3-dimensional array x, and a certain element in that array is indexed by an index vector

$$\boldsymbol{n} = \begin{bmatrix} n_0 \\ n_1 \\ n_2 \end{bmatrix}.$$

Generally, at an output or input port p of an operation v, the relation between index vector $\boldsymbol{n}$ and the iterator vector $\boldsymbol{i}$ of the operation is given by

$$\boldsymbol{n}(p,\boldsymbol{i}) = \boldsymbol{A}(p)\,\boldsymbol{i} + \boldsymbol{b}(p).$$

For example, at the second input of the multiplication operation of Figure 1, we have

$$\begin{bmatrix} n_0 \\ n_1 \\ n_2 \end{bmatrix} = \begin{bmatrix} f \\ k_1 \\ 5-2k_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} f \\ k_1 \\ k_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}.$$

Note that we assume that the index vector can be written as a linear expression in the iterator vector, which is a realistic assumption in video signal processing. Furthermore, for the productions we assume single assignments, i.e., each element of an array can be produced at most once.

A signal flow describes what operations have to be executed and what data is used. Next, a schedule defines when they are executed and on what processing unit.

Definition 2 (schedule). A schedule σ is given by a 4-tuple $(\boldsymbol{p}, s, W, h)$, where

- $\boldsymbol{p}(v) \in \mathbb{Z}^{\delta(v)}$ is a period vector, for each operation $v \in V$,
- $\boldsymbol{s}(v) \in \mathbb{Z}$ is a start time, for each $v \in V$,
- W is a set of processing units, and
- $h : V \to W$ is a function that assigns each operation to a PU of the same type. □

Given a schedule, the time $c(v,\boldsymbol{i})$ at which an execution $\boldsymbol{i}$ of operation v takes place is given by

$$c(v,\boldsymbol{i}) = \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{i} + s(v).$$

For the video algorithm of Figure 1, the multiplication has a period vector

$$\boldsymbol{p}(mu) = \begin{bmatrix} 30 \\ 7 \\ 2 \end{bmatrix},$$

so if the start time of this operation is chosen $s(mu) = 6$, then execution $\boldsymbol{i} = [f \;\; k_1 \;\; k_2]^{\mathrm{T}}$ takes place at time

$$c(mu,\boldsymbol{i}) = 30f + 7k_1 + 2k_2 + 6.$$

For the scope of this paper, we assume that the period vectors $\boldsymbol{p}$ and the set W of processing units are given. For the general problem, we refer to [14].

Next, we define the three kinds of constraints that a schedule has to satisfy. First, we have lower bounds $\underline{s}(v)$ and upper bounds $\overline{s}(v)$ on the start times of the operations $v \in V$, resulting in *timing constraints* that specify that

$$\underline{s}(v) \le s(v) \le \overline{s}(v),$$

for all $v \in V$. If the start time of an operation v is not bounded from below or from above, this is denoted by $\underline{s}(v) = -\infty$ or $\overline{s}(v) = \infty$, respectively.

Secondly, we have *processing unit constraints*, which specify that at most one execution of an operation can occupy a processing unit at a time. So, if we have an execution $\boldsymbol{i}$ of an operation $u \in V$ and an execution $\boldsymbol{j}$ of an operation $v \in V$, with $h(u) = h(v)$ and $(u,\boldsymbol{i}) \neq (v,\boldsymbol{j})$, then we must have

$$c(u,\boldsymbol{i}) \neq c(v,\boldsymbol{j}).$$

Note that this applies to two executions of two different operations, as well as to two different executions of one and the same operation.

Thirdly, we have *precedence constraints*, which specify that data must be produced before it is consumed. So, if we have an edge $(p,q) \in E$ from an output port p of an operation u to an input port q of an operation v, and if we have an execution $\boldsymbol{i}$ of u and an execution $\boldsymbol{j}$ of v for which $\boldsymbol{n}(p,\boldsymbol{i}) = \boldsymbol{n}(q,\boldsymbol{j})$, then we must have

$$c(u,\boldsymbol{i}) < c(v,\boldsymbol{j}).$$

Now we can formulate the restricted version of the multidimensional periodic scheduling problem that we consider in this paper.

Definition 3 (multidimensional periodic scheduling). Given are a signal flow graph G, a set W of processing units, and for each operation a period vector and a lower and upper bound on its start time. Find a schedule σ that satisfies the timing constraints, the processing unit constraints, and the precedence constraints. □

Multidimensional periodic scheduling is NP-hard [16].

3 Constraint Calculations

In this section we briefly discuss how we check the PU and precedence constraints by formulating them as ILP problems and solving them by means of an all-integer ILP algorithm.

3.1 Processing Unit Constraints

The first sub-problem during scheduling is to check the processing unit constraints. To this end, we have to check (i) whether no two executions of two different operations overlap in time if they are assigned to the same PU, and (ii) whether no two different executions of one and the same operation overlap.

(i) To check whether two different operations $u, v \in V$ with $h(u) = h(v)$ have overlapping executions, we have to solve a so-called *PU conflict problem*, in which the question is whether there is a solution $\boldsymbol{i}, \boldsymbol{j}$ to the following set of constraints:

$$\begin{gathered} \boldsymbol{p}^{\mathrm{T}}(u)\,\boldsymbol{i} + s(u) = \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{j} + s(v) \\ \mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{I}(u) \\ \mathbf{0} \leq \boldsymbol{j} \leq \boldsymbol{I}(v) \\ \boldsymbol{i}, \boldsymbol{j} \text{ integer.} \end{gathered}$$

(ii) To check whether an operation v has two overlapping executions, we have to solve a so-called *PU self-conflict problem*, in which the question is whether there is a solution $\boldsymbol{i}, \boldsymbol{j}$ to the following set of constraints:

$$\begin{gathered} \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{i} + s(v) = \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{j} + s(v) \\ \boldsymbol{i} \neq \boldsymbol{j} \\ \mathbf{0} \leq \boldsymbol{i}, \boldsymbol{j} \leq \boldsymbol{I}(v) \\ \boldsymbol{i}, \boldsymbol{j} \text{ integer.} \end{gathered}$$

Although the constraint $\boldsymbol{i} \neq \boldsymbol{j}$ is not linear, it can easily be replaced by a constraint $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{i} > \boldsymbol{c}^{\mathrm{T}}\boldsymbol{j}$ by choosing proper coefficients c_k. Note that the PU self-conflict problems can be solved before the start time and PU assignment, since they do not depend on these assignments.

3.2 Precedence Constraints

The second sub-problem during scheduling is to check the precedence constraints. To this end, we have to check that data is produced before it is consumed. So, for each edge $e = (p, q) \in E$ from an output port p of an operation u to an input port q of an operation v, we must have

$$\boldsymbol{p}^{\mathrm{T}}(u)\,\boldsymbol{i} + s(u) < \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{j} + s(v),$$

for all executions $\boldsymbol{i}$ of u and $\boldsymbol{j}$ of v with $\boldsymbol{n}(p, \boldsymbol{i}) = \boldsymbol{n}(q, \boldsymbol{j})$. So, if we determine a weight $w(e)$ for edge e, by solving the following *precedence problem*:

$$\begin{array}{rll} w(e) = & \text{maximum} & \boldsymbol{p}^{\mathrm{T}}(u)\,\boldsymbol{i} - \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{j} + 1 \\ & \text{subject to} & \boldsymbol{A}(p)\,\boldsymbol{i} + \boldsymbol{b}(p) = \boldsymbol{A}(q)\,\boldsymbol{j} + \boldsymbol{b}(q) \\ & & \mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{I}(u) \\ & & \mathbf{0} \leq \boldsymbol{j} \leq \boldsymbol{I}(v) \\ & & \boldsymbol{i}, \boldsymbol{j} \text{ integer,} \end{array}$$

then we just have to check whether $s(v) - s(u) \geq w(e)$. Note that if the above system does not have a solution, then $w(e) = -\infty$, in which case $s(v) - s(u) \geq w(e)$ always holds. For the video algorithm of Figure 1, the resulting edge weights are denoted in Figure 2.

3.3 All-Integer ILP

Although the checks of the above constraints are NP-complete problems [16], the corresponding ILP instances are small in practice; typically they contain less than ten variables and the number of constraints is also in this order. The reason for this is that the ILP instances have to be solved each time for a pair of operations, and that the number of variables in such an ILP instance is determined by the number of dimensions of the two operations involved. So, if the number of operations increases, then only the number of ILP instances increases, but not their sizes.

Since the ILP instances encountered in practice are small, and since we want to check the constraints exactly, we opt to solve the above ILP problems to optimality. We do this by means of an all-integer ILP algorithm [3], which has the main advantage that no rounding errors occur, and thus prevents wrong conclusions. Furthermore, the algorithm runs extremely fast for practical instances. Due to space limitations, the all-integer ILP algorithm and its application to the mentioned ILP problems is not discussed here; see [14].

4 Start Time and PU Assignment

In this section we present an algorithm for the multidimensional periodic scheduling problem with given periods and a given set of processing units. Since the problem is NP-hard and since practical instances may contain a large number of operations, we opt for a heuristic approach.

The first step of the algorithm is to determine the edge weights of all data dependencies in E, by solving the corresponding precedence problems discussed in Section 3.2, in order to determine the minimal differences between the operations' start times.

The second step of the algorithm is to determine the initial freedom of the operations' start times. To this end, we determine an 'as soon as possible' (ASAP) schedule and an 'as late as possible' (ALAP) schedule, based on the determined edge weights and the lower and upper bounds on the operations' start times. An algorithm for determining the ASAP and ALAP schedules can be found in e.g. [14, 15].

In the remainder of the algorithm we determine a start time and PU assignment by means of an iterative approach, which shows some resemblance with list scheduling [5]. In this approach, a partial solution is iteratively augmented, by selecting in each iteration an unscheduled operation and assigning it a start time and a PU, or reducing its freedom.

The partial solutions consist of a *start time domain* $\{s_{\mathrm{lo}}(v), \ldots, s_{\mathrm{hi}}(v)\}$ for each operation $v \in V$, from which $s(v)$ has to be chosen, and a set $F \subseteq V$ of operations that are already scheduled, with fixed start time $s(v) = s_{\mathrm{lo}}(v) = s_{\mathrm{hi}}(v)$ and processing unit $h(v) \in W$ of the corresponding type. Throughout the algorithm, the start time domains are kept *feasible*, which means that both the start time assignments $s = s_{\mathrm{lo}}$ and $s = s_{\mathrm{hi}}$ obey the precedence and timing

constraints. This implies that whenever the start time domain of an operation changes, then the start time domains of the other operations have to be updated, using the edge weights $w(e)$.

A basic algorithm for the scheduling problem can now be written as shown in Figure 3.

---

1. Calculate the edge weights $w(e)$ for all $e \in E$, by solving the corresponding precedence problems (by means of ILP).
2. Determine the ASAP and ALAP schedules, and initialize the start time domains. Furthermore, set $F = \emptyset$.
3. If $F = V$, then stop. Otherwise, take an operation $v \in V \setminus F$.
4. Try to schedule operation v at start time $s_{\text{lo}}(v)$, i.e., try to assign to v a start time $s(v) = s_{\text{lo}}(v)$ and a processing unit $h(v) \in W$ of the same type, such that v has no conflict with operations $u \in F$ that have already been assigned to the same PU $h(u) = h(v)$. Here, conflicts are detected by solving the corresponding PU conflict problems (by means of ILP).
5. If v can be scheduled at start time $s_{\text{lo}}(v)$, then assign $s(v) = s_{\text{lo}}(v)$ and $s_{\text{hi}}(v) = s_{\text{lo}}(v)$, fix $h(v)$, and add v to F. Otherwise, increase $s_{\text{lo}}(v)$ by 1.
6. Adjust the start time domains $\{s_{\text{lo}}(u),\ldots,s_{\text{hi}}(u)\}$ of the unscheduled operations $u \in V \setminus F$, using the edge weights $w(e)$.
7. If now an operation $u \in V \setminus F$ has an empty start time domain, i.e., $s_{\text{lo}}(u) > s_{\text{hi}}(u)$, then stop; no schedule is found. Otherwise, return to Step 3.

Figure 3. The start time and PU assignment algorithm.

In the remainder of this section, we propose some refinements to the basic algorithm. The first one is to use the following priority function to select the operation v in Step 3. We start by only considering operations $v \in V \setminus F$ for which all preceding operations are already scheduled. This is done since scheduling an operation v at time $s_{\text{lo}}(v)$ severely reduces the freedom of its predecessors. In case the graph contains cycles, we break them by only considering edges with positive weights. If more than one operation is eligible, we select the operation v for which the freedom $s_{\text{hi}}(v) - s_{\text{lo}}(v)$ is minimal.

Next, we slightly alter Step 4 of the algorithm. In this step, the PU assignment $h(u)$ of previously scheduled operations u remains unchanged. However, by reassigning these operations, together with the new operation v, it is more probable that operation v can be scheduled at start time $s_{\text{lo}}(v)$.

Thirdly, we alter the increasing of $s_{\text{lo}}(v)$ in Step 5. If namely an operation v cannot be scheduled at time $s_{\text{lo}}(v)$ due to conflicts with other, already scheduled operations $u \in F$, then based on the solutions of the corresponding ILP instances we can derive a larger amount for shifting v. In this way the efficiency of the algorithm is improved.

Finally, we alter the algorithm in order to find a solution with less memory requirements. To this end, operations v that produce more data than they consume are tried to be scheduled at time $s_{\text{hi}}(v)$ instead of $s_{\text{lo}}(v)$ in Step 4.

5 Results

The presented algorithm has been implemented in C++, and is incorporated in the design methodology Phideo [8]. In this section, we show some experimental results. The presented run times are achieved on an HP9000/735.

The first example is that of Figure 1. Assuming a lower bound 0 and an upper bound 50 for all the start times, the obtained schedule is shown in Figure 4. The parallelism

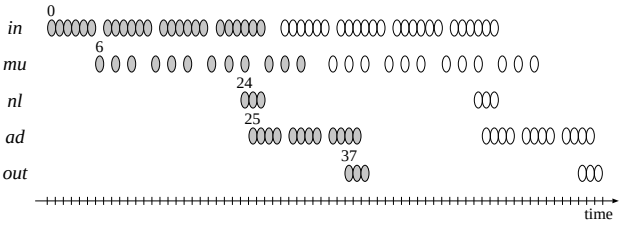

Figure 4. The obtained schedule for the example of Figure 1, showing the executions for iterator values $f = 0$ (grey) and $f = 1$ (white). The numbers denote the operations' start times.

in this schedule, including overlapping frames (iterator f), is obtained straightforwardly. After determining the edge weights and the ASAP and ALAP schedules, the operations *in*, *mu*, *ad*, and *out* are scheduled at their ASAP times, after which operation *nl* is scheduled just before operation *ad*. The run time for this example is 0.2 seconds.

The second example is a discrete cosine transformation [9], with 35 operations. The run time for this example is 6.9 seconds. The period of the (infinite) frame loop is 32 clock cycles. Of the first frame, the first execution takes place in clock cycle 1 and the last one in cycle 183. This means that executions of six different frames overlap in time, making it impracticable to solve the scheduling problem by hand or by loop transformations. Figure 5 shows the executions of the operations in a time window of 32 clock cycles.

6 Parameters

For some applications, the numbers of repetitions of operations are not constant, but they depend on parameters. For instance, the number of pixels on a video line may be parametric, in order to handle multiple video standards. Figure 6 shows a parametric version of the video algorithm of Figure 1. In this example, we have a parameter m that can assume any integer value between 2 and 10, and a parameter n that can assume any integer value between 2 and 5. Now we have to determine a *parametric schedule*, and it has to be feasible for any of these values. The resulting design can then be used for any of the parameter settings.

The use of parameters for the numbers of repetitions introduces parametric periods, as shown in Figure 6. Furthermore, the index expressions of multidimensional arrays generally also become parametric. Therefore, in the model we

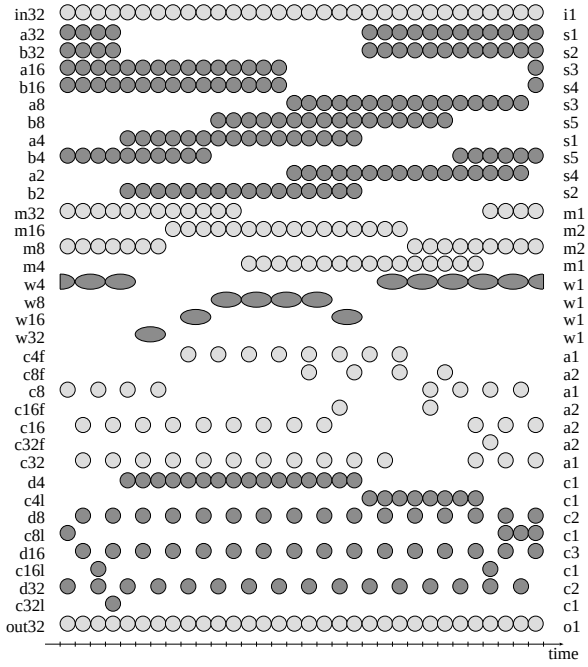

Figure 5. The results for the DCT example, shown in a time window of 32 clock cycles. At the left hand side the operation names are shown. At the right hand side their PU instances are shown. All operations execute in one clock cycle, except operations w4, w8, w16, and w32, which take two clock cycles.

```
    parameter m in {2,...,10}
    parameter n in {2,...,5}
    for f = 0 to ∞ period m(2n+1)+2
    begin
        for j1 = 0 to m-1 period 2n+1
            for j2 = 0 to 2n-1 period 1
{in}            x[f][j1][j2] = input( )
        for k1 = 0 to m-1 period 2n+1
            for k2 = 0 to n-1 period 2
{mu}            y[f][k1][k2] = x[f][k1][2k2] * x[f][k1][2n-1-2k2]
        for l1 = 0 to n-1 period 1
{nl}        z[f][l1][-1] = 0
        for m1 = 0 to n-1 period m+1
            for m2 = 0 to m-1 period 1
{ad}            z[f][m1][m2] = z[f][m1][m2-1] + y[f][m2][m1]
        for n1 = 0 to n-1 period 1
{out}       = output(z[f][n1][m-1])
    end
```

Figure 6. Example of a parametric video algorithm.

no longer have constant entries in the iterator bound vectors $\boldsymbol{I}(v)$, the period vectors $\boldsymbol{p}(v)$, the index matrices $\boldsymbol{A}(p)$, and the index offset vectors $\boldsymbol{b}(p)$, but these entries are substituted by polynomials in the parameters, with integer coefficients. Furthermore, each parameter has a specified range in the model, from which it can assume any integer value. Now the question is to find a parametric start time assignment and a PU assignment, that is feasible for any of the possible parameter settings.

In order to handle parametric video algorithms, we have extended the all-integer ILP algorithm, which we used to solve the sub-problems. Basically, what we did is substituting the constant entries in the ILP tableaus by polynomials in the parameters, and adapting the ILP algorithm to handle these tableaus. In this way, we can solve the parametric ILP problems for checking the PU constraints and the precedence constraints, resulting a.o. in parametric edge weights, after which we based the scheduling algorithm on the new sub-routines. For more details about the parametric algorithms, we refer to [14].

The parametric algorithms have been implemented in C++. Figure 7 shows the parametric start time assignment obtained by these algorithms for the example of Figure 6. The start time assignment is given by $s(in) = 0$, $s(mu) = 2n$, $s(nl) = 2mn$, $s(ad) = 2mn+1$, and $s(out) = 3mn+1$, which is feasible for any value setting of m and n within their ranges. The run time for this example is 0.4 seconds.

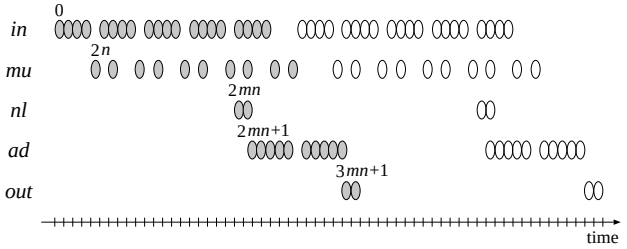

Figure 7. A feasible parametric schedule for the example of Figure 6, here shown for parameter values $m = 5$ and $n = 2$. Again, the executions are shown only for iterator values $f = 0$ (grey) and $f = 1$ (white).

7 Conclusion

We have presented an iterative algorithm to solve the multidimensional periodic scheduling problem with given periods and processing units, both for non-parametric and parametric descriptions. The algorithm is based on a fast (parametric) all-integer ILP routine to solve the sub-problems concerning the checks of the processing unit constraints and the precedence constraints. Furthermore, we showed some experimental results, in which the required parallelism was easily obtained by means of the presented algorithms. The algorithms have been incorporated in the design methodology Phideo [8], which is currently used by several design groups within Philips.

References

[1] J. Bu, E. Deprettere, and L. Thiele. Systolic array implementation of nested loop programs. *Proc. Int. Conf. on Application*

Specific Array Processing, 31–42, 1990.

[2] P. Feautrier. Dataflow analysis of array and scalar references. *Int. Journal of Parallel Programming* **20** (1991) 23–53.

[3] F. Glover. A new foundation for a simplified primal integer programming algorithm. *Operations Research* **16** (1968) 727–740.

[4] G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man. An efficient microcode-compiler for custom DSP-processors. *Proc. of the ICCAD*, 24–27, 1987.

[5] R. Haupt. A survey of priority-rule based scheduling. *OR Spektrum* **11** (1989) 3–16.

[6] J.H.M. Korst. *Periodic Multiprocessor Scheduling.* PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, December 1992.

[7] E.A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. on Comp.* **C-36** (1987) 24–35.

[8] P.E.R. Lippens, J.L. van Meerbergen, A. van der Werf, W.F.J. Verhaegh, B.T. McSweeney, J.A. Huisken, and O.P. McArdle. PHIDEO: A silicon compiler for high speed algorithms. *Proc. of the EDAC*, 436–441, 1991.

[9] P.E.R. Lippens, J.L. van Meerbergen, W.F.J. Verhaegh, D.M. Grant, and A. van der Werf. Design of a 30 MHz, 32/16/8-points DCT processor with Phideo. In J. Rabaey, P. Chau, and J. Eldon, eds., *VLSI Signal Processing VII*, 24–32. IEEE Press, New York, 1994.

[10] N. Park and A.C. Parker. Sehwa: A software package for synthesis of pipelines from behavioral specifications. *IEEE Trans. on CAD* **7** (1988) 356–370.

[11] M. Potkonjak and J. Rabaey. A scheduling and resource allocation algorithm for hierarchical signal flow graphs. *Proc. of the DAC*, 7–12, 1994.

[12] W. Pugh. The omega test: A fast and practical integer programming algorithm for dependence analysis. *Proc. Supercomputing*, 18–22, 1991.

[13] M.F.X.B. van Swaaij, F.H.M. Franssen, F.V.M. Catthoor, and H.J. De Man. Modeling data flow and control flow for high level memory management. *Proc. of the EDAC*, 8–13, 1992.

[14] W.F.J. Verhaegh. *Multidimensional Periodic Scheduling.* PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 1995.

[15] W.F.J. Verhaegh, P.E.R. Lippens, E.H.L. Aarts, J.H.M. Korst, J.L. van Meerbergen, and A. van der Werf. Improved force-directed scheduling in high-throughput digital signal processing. *IEEE Trans. on CAD* **14** (1995) 945–960.

[16] W.F.J. Verhaegh, P.E.R. Lippens, E.H.L. Aarts, J.L. van Meerbergen, and A. van der Werf. Multidimensional periodic scheduling: Model and complexity. *Proc. of the Euro-Par*, II, 226–235, 1996.