Multidimensional Periodic Scheduling
Model and Complexity

W.F.J. Verhaegh[1] P.E.R. Lippens[1] E.H.L. Aarts[1,2]
J.L. van Meerbergen[1] A. van der Werf[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 discuss the multidimensional periodic scheduling problem, which originates from the design of high-throughput real-time digital signal processing systems. We introduce the concept of multidimensional periodic operations in order to cope with problems originating from loop hierarchies and explicit timing requirements. We present a model of the multidimensional periodic scheduling problem and show that this problem and two related sub-problems are NP-hard. Furthermore, we identify several special cases induced by practical situations. Some of these special cases are proven to be well-solvable. Finally, we present a sketch of a solution approach.

1 Introduction

The design of a digital signal processing (DSP) system concerns the mapping of a DSP algorithm onto 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 the 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. The problem is now to determine for each operation a period vector and a start time, subject to a set of constraints. In addition, we need to assign the operations to processing units, on which they are executed.

There are three sets of scheduling constraints. First, we have *timing constraints*, which bound the period vectors and start times of the operations. Second, we have *processing unit constraints*, which specify that each processing unit can perform only one execution of an operation at a time. Third, we have *precedence constraints*, which are due to data dependencies.

The scheduling objective we consider is to minimize the area occupied by the hardware. Due to the high throughput, 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 [7], which is aimed at the automated design of high-throughput real-time DSP systems. Furthermore, the model of multidimensional peri-

odic operations also plays an important role in other sub-problems emerging from this design methodology, such as memory synthesis.

The model in this paper considers operations that are executed repeatedly, with both multidimensional repetitions and strict periodicity [12]. 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. Since the parallelism depends directly on the periods and start times of the operations, it is very straightforward to determine them such that the desired throughput is obtained. Furthermore, the multidimensional periodicity matches very well with the application domain of video signal processing.

In the area of one-dimensional periodic scheduling, related work is done on mapping video signal processing algorithms onto programmable video signal processors [5] and on synchronous data flow for DSP [6]. If, in addition, all operations have the same period, related work can be found in the area of pipelined scheduling [8, 13].

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, 9], 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 [11] the loop transformations are handled by a method based on placement of polytopes. Furthermore, related work is done in the area of systolic array design [1] and in the area of data flow analysis for parallel program construction [2, 10].

The objective of this paper is to present a formal model of multidimensional periodic operations and the multidimensional periodic scheduling problem, and to discuss its computational complexity. A detailed solution approach and experimental results are beyond the scope of this paper. The organization is as follows. In Section 2 we present a model of periodic operations and formulate the multidimensional periodic scheduling problem. In Sections 3 and 4 we discuss the computational complexity of checking processing unit constraints and precedence constraints, respectively, together with several special cases. Next, we discuss in Section 5 the complexity of the scheduling problem in its entirety. Due to space limitations, the complexity proofs will only be sketched. For full proofs, see [12]. Finally, we present in Section 6 a sketch of a solution approach.

2 Modeling Multidimensional Periodic Operations

Input for scheduling is a signal flow graph representing a video algorithm, and a set of timing requirements. Figure 1 shows an example of a part of a video algorithm.

```
for i0 = 0 to 1
    for i1 = 0 to 2
        x[3i0 + i1] = v(...)
for j0 = 0 to 5
    ... = u(x[j0])
```

Figure 1. Example of a part of a video algorithm, with a two-dimensional operation v and a one-dimensional operation u, using a one-dimensional array x.

Before describing a signal flow graph, we first describe the operation types and the corresponding processing unit types. We assume that the operations in a signal flow graph

must be executed on dedicated processing units, i.e., we assume a one-to-one relation between operation types and processing unit types. Before we define them, we mention that the time unit we maintain throughout this paper is the clock cycle, and all time points are given by clock cycles $c \in \mathbb{Z}$, where $\mathbb{Z}$ is the set of integer numbers. For an explanation of the symbols in the definition, see the text below.

Definition 1 (operation type set). An operation type set is given by a 6-tuple $(T, \tilde{I}, \tilde{O}, r, o, a)$, where

- T is a set of operation types,
- $\tilde{I}(t)$ denotes the set of *operation input ports*, for each $t \in T$,
- $\tilde{O}(t)$ denotes the set of *operation output ports*, for each $t \in T$,
- $r(t, \tilde{p}) \in \mathbb{Z}$ denotes the *relative transfer time*, for each $t \in T$, $\tilde{p} \in \tilde{P}(t) = \tilde{I}(t) \cup \tilde{O}(t)$,
- $o(t) \in \mathbb{N}$ denotes the *occupation time*, for each $t \in T$, and
- $a(t) \in \mathbb{N}$ denotes the *area cost*, for each $t \in T$. □

Here, $\mathbb{N}$ is the set of non-negative integers. The definition allows an operation type to have several inputs and several outputs. If an execution of an operation of type t is scheduled at time c, then the production or consumption of a variable at operation port $\tilde{p} \in \tilde{P}(t)$ takes place at time $c + r(t, \tilde{p})$, and the processing unit that executes the operation is occupied at times $c, \ldots, c + o(t) - 1$. Input and output nodes of a signal flow graph are modeled as operations without operation input ports and operation output ports, respectively.

Now we can define a signal flow graph as follows. For an explanation of the symbols, see the text below.

Definition 2 (signal flow graph). A signal flow graph G is given by a 6-tuple (V, t, I, E, A, b), where

- V is a set of multidimensional periodic operations,
- $t(v) \in T$ denotes the operation type, for each $v \in V$,
- $I(v) \in \mathbb{N}_\infty^{\delta(v)}$ denotes the *iterator bound vector*, for each $v \in V$,
- $E \subset O \times I$ is a set of directed edges representing data dependencies, where $O = \{(v, \tilde{o}) \mid v \in V \wedge \tilde{o} \in \tilde{O}(t(v))\}$ denotes the set of *output ports*, and $I = \{(v, \tilde{\imath}) \mid v \in V \wedge \tilde{\imath} \in \tilde{I}(t(v))\}$ denotes the set of *input ports*,
- $A(p) \in \mathbb{Z}^{\alpha(p) \times \delta(v)}$ denotes the *index matrix*, for each $p = (v, \tilde{p}) \in P = I \cup O$,
- $b(p) \in \mathbb{Z}^{\alpha(p)}$ denotes the *index offset vector*, for each $p \in P$. □

Here, $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. The length $\delta(v)$ of an iterator bound vector $I(v)$ denotes the number of dimensions in which the operation is repeated, i.e., the number of enclosing loops. Each execution of an operation $v \in V$ can be identified by an *iterator vector* $i \in \mathbb{Z}^{\delta(v)}$, for which $\mathbf{0} \leq i \leq I(v)$. So, the iterator i_k in dimension $k = 0, \ldots, \delta - 1$ runs from 0 to $I_k(v)$. We assume that only dimension 0 may have an unbounded number of repetitions, denoted by $I_0 = \infty$; the others are finite. The set of all possible iterator vectors of operation v is given by $\mathcal{I}(v) = \{i \in \mathbb{Z}^{\delta(v)} \mid \mathbf{0} \leq i \leq I(v)\}$, and is called the *iterator space*. Figure 2 shows the two-dimensional operation v of Figure 1.

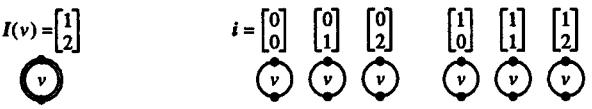

Figure 2. (left) An example of a two-dimensional operation, denoted by a double circle, and (right) its executions, denoted by single circles.

In a video algorithm, data transport between operations is described by means of mul-

tidimensional arrays. In a signal flow graph this is represented by means of output and input ports, and edges between them. At an output port $p = (v, \vec{o})$ of an operation v, data is produced at each execution $\boldsymbol{i}$ of v, which corresponds to an element of a multidimensional array with dimension $\alpha(p)$. Such an element is indexed by an *index vector* $\boldsymbol{n}(p,\boldsymbol{i}) \in \mathbb{Z}^{\alpha(p)}$, which is given by

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

given the index matrix $\boldsymbol{A}(p)$ and index offset vector $\boldsymbol{b}(p)$. If we now have an input port $q = (u, \vec{\imath})$ of an operation u, with $(p,q) \in E$, then the array element is consumed at input port q at those executions $\boldsymbol{j}$ of operation u for which $\boldsymbol{n}(q,\boldsymbol{j}) = \boldsymbol{n}(p,\boldsymbol{i})$, where similarly $\boldsymbol{n}(q,\boldsymbol{j}) = \boldsymbol{A}(q)\,\boldsymbol{j} + \boldsymbol{b}(q)$. For the productions we assume single assignments, i.e., each element of an array can be produced at most once. Figure 3 shows the data transport

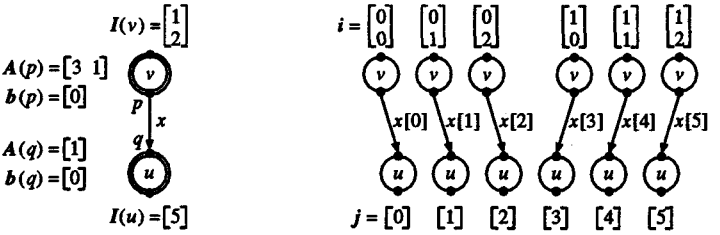

Figure 3. (left) An example of data transport between two multidimensional operations via a one-dimensional array x, and (right) the different executions and array elements.

between operation v and u of Figure 1. Next, we need the definition of a schedule.

Definition 3 (schedule). Given a signal flow graph $G = (V,t,\boldsymbol{I},E,\boldsymbol{A},\boldsymbol{b})$, a schedule σ is given by a time assignment and a processing unit assignment, represented by a 4-tuple $(\boldsymbol{p},s,W,h)$, where

- the *time assignment* is given by a period vector $\boldsymbol{p}(v) \in \mathbb{Z}^{\delta(v)}$ and a start time $s(v) \in \mathbb{Z}$, for each operation $v \in V$, resulting in a time $c(v,\boldsymbol{i})$ at which an execution $\boldsymbol{i}$ of operation v is scheduled that is given by
$$c(v,\boldsymbol{i}) = \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{i} + s(v),$$
- the *processing unit assignment* is given by a set W of processing units and a function $h : V \rightarrow W$ that assigns each operation to a processing unit that executes it, such that $t(v) = t(h(v))$, for all $v \in V$. Note that we assume that t is also defined on W. □

Figure 4 shows a possible time assignment for the operation v of Figure 1. The time at

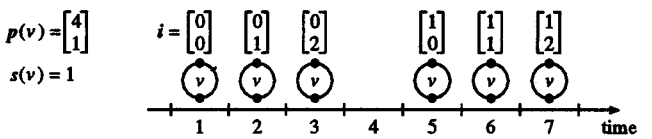

Figure 4. A possible time assignment for a two-dimensional operation.

which data is produced or consumed at port $p = (v, \vec{p})$ at execution $\boldsymbol{i}$ of an operation v is given by the time of that execution plus the relative transfer time, i.e.,

$$c(p,\boldsymbol{i}) \;=\; c(v,\boldsymbol{i}) + r(t(v),\vec{p}) \;=\; \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{i} + s(v) + r(t(v),\vec{p}) \;=\; \boldsymbol{p}^{\mathrm{T}}(p)\,\boldsymbol{i} + s(p),$$

where we define $\boldsymbol{p}(p) = \boldsymbol{p}(v)$ and $s(p) = s(v) + r(t(v),\vec{p})$. For notational ease, we define $\boldsymbol{I}(p) = \boldsymbol{I}(v)$, $\mathcal{I}(p) = \mathcal{I}(v)$, and $\delta(p) = \delta(v)$, and we speak of execution $\boldsymbol{i}$ of port p.

Next, we define the three sets of constraints that a schedule must satisfy.

Definition 4 (timing constraints). Given are a signal flow graph $G = (V,t,I,E,A,b)$, a schedule $\sigma = (p,s,W,h)$, and for each operation $v \in V$ a lower bound $\underline{p}(v) \in \mathbb{Z}_\infty^{\delta(v)}$ and an upper bound $\overline{p}(v) \in \mathbb{Z}_\infty^{\delta(v)}$ on the period vector $p(v)$, and a lower bound $\underline{s}(v) \in \mathbb{Z}_\infty$ and an upper bound $\overline{s}(v) \in \mathbb{Z}_\infty$ on the start time $s(v)$, where $\mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty,+\infty\}$. Then for each $v \in V$, $p(v)$ and $s(v)$ must satisfy

$$\underline{p}(v) \le p(v) \le \overline{p}(v) \ \wedge \ \underline{s}(v) \le s(v) \le \overline{s}(v). \qquad \square$$

Definition 5 (processing unit constraints). Given are a signal flow graph $G = (V,t,I,E,A,b)$ and a schedule $\sigma = (p,s,W,h)$. Then for each execution i of an operation $u \in V$, and for each execution j of an operation $v \in V$, with $h(u) = h(v)$ and $(u,i) \neq (v,j)$, we must have

$$c(u,i) + k \neq c(v,j) + l,$$

for all $k,l \in \{0,\ldots,o-1\}$, where $o = o(t(u)) = o(t(v))$. $\square$

Definition 6 (precedence constraints). Given are a signal flow graph $G = (V,t,I,E,A,b)$ and a schedule $\sigma = (p,s,W,h)$. Then for each execution i of an output port $p \in O$, and for each execution j of an input port $q \in I$, with $(p,q) \in E$, we must have

$$n(p,i) = n(q,j) \ \Rightarrow \ c(p,i) < c(q,j). \qquad \square$$

The cost function we consider in this paper reflects the total area, which is not only determined by the processing units, but also by the required memory size and bandwidth. Therefore, we include in the cost function the maximum number of variables that are simultaneously alive and the maximum number of simultaneous memory accesses. To this end, we define a set of *resource types* $T^* = T \cup \{t_v,t_a\}$, where t_v corresponds to memory cells, with area cost $a(t_v) \in \mathbb{N}$ for each variable, and t_a corresponds to memory terminals, with area cost $a(t_a) \in \mathbb{N}$ for each access. To avoid multiple countings, we determine the lifetimes and accesses of each cluster of ports belonging to one array, rather than for each data precedence in E. To this end, we first define array clusters.

Definition 7 (array clusters). Two ports $p,q \in P$ are said to access the same array, denoted by $p \bowtie q$ and pronounced as p joins q, if and only if they are weakly connected, i.e., $p = q$ or there is a path of ports $(p_0,p_1,\ldots,p_n)$ with $p_0 = p$ and $p_n = q$ and

$$(p_i,p_{i+1}) \in E \ \vee \ (p_{i+1},p_i) \in E$$

for all $i = 0,\ldots,n-1$. Now, an array cluster is denoted by a set A of ports, which is a subset of P, such that $p \bowtie q$ for all $p,q \in A$, and such that A cannot be extended. The set of all array clusters is denoted by $\mathcal{A}$. $\square$

Assuming that a variable is alive from the first time after its production up to and including the time of its last consumption, the set of array elements of an array cluster $A \in \mathcal{A}$ that are alive at time c is given by

$$\mathcal{L}(A,c) = \{n \in \mathbb{Z}^\alpha \mid \ p,q \in A \wedge (p,q) \in E \wedge i \in \mathcal{I}(p) \wedge j \in \mathcal{I}(q) \wedge n = n(p,i) = n(q,j) \wedge c(p,i) < c \le c(q,j)\}.$$

The set of elements of an array cluster $A \in \mathcal{A}$ that are accessed at time c is given by

$$\mathcal{B}(A,c) = \{n \in \mathbb{Z}^\alpha \mid p \in A \wedge i \in \mathcal{I}(p) \wedge n = n(p,i) \wedge c(p,i) = c\}.$$

Definition 8 (total area cost). The total area cost of a schedule σ is the sum of the processing unit cost, the storage cost, and the access cost, and is given by

$$f(\sigma) = \sum_{w \in W} a(t(w)) + a(t_v) \max_{c \in \mathbb{Z}} \sum_{A \in \mathcal{A}} |\mathcal{L}(A,c)| + a(t_a) \max_{c \in \mathbb{Z}} \sum_{A \in \mathcal{A}} |\mathcal{B}(A,c)|. \qquad \square$$

We can now define the scheduling problem we consider in this paper.

Definition 9 (multidimensional periodic scheduling (MPS)). Given are a signal flow graph G, a set of resource types T^*, and lower and upper bounds on the period vectors and start times of the operations. Find a schedule σ that obeys the timing constraints, processing unit constraints, and precedence constraints, for which the total area cost $f(\sigma)$ is minimal. □

3 Complexity of Checking the Processing Unit Constraints

In this section we consider the problem of deciding whether two different operations give a conflict when assigned to the same processing unit.

Definition 10 (processing unit conflict (PUC)). Given are two operations u and v, with their iterator bound vectors $\boldsymbol{I}(u) \in \mathbb{N}_\infty^{\delta(u)}$, $\boldsymbol{I}(v) \in \mathbb{N}_\infty^{\delta(v)}$, their period vectors $\boldsymbol{p}(u) \in \mathbb{Z}^{\delta(u)}$, $\boldsymbol{p}(v) \in \mathbb{Z}^{\delta(v)}$, their start times $s(u), s(v) \in \mathbb{Z}$, and their occupation time $o \in \mathbb{N}_+$. Are there there vectors $\boldsymbol{i} \in \mathbb{Z}^{\delta(u)}$ and $\boldsymbol{j} \in \mathbb{Z}^{\delta(v)}$, $\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{I}(u)$, $\mathbf{0} \leq \boldsymbol{j} \leq \boldsymbol{I}(v)$, and integers x and y, $0 \leq x, y \leq o-1$, for which

$$\boldsymbol{p}^{\mathrm{T}}(u)\,\boldsymbol{i} + s(u) + x = \boldsymbol{p}^{\mathrm{T}}(v)\,\boldsymbol{j} + s(v) + y\,?$$ □

In this definition, $\mathbb{N}_+$ is the set of positive integers. By concatenation and rewriting [12], we can show that this problem is equivalent to the following one.

Definition 11 (PUC reformulated). Given are an iterator bound vector $\boldsymbol{I} \in \mathbb{N}_+^\delta$, a period vector $\boldsymbol{p} \in \mathbb{N}_+^\delta$, and a number $s \in \mathbb{N}_+$. Is there a vector $\boldsymbol{i} \in \mathbb{Z}^\delta$, $\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{I}$, for which $\boldsymbol{p}^{\mathrm{T}}\boldsymbol{i} = s$? □

To discuss the complexity of PUC, we use the subset sum problem [3].

Definition 12 (subset sum (SUB)). Given are a set A, with for each $a \in A$ a size $s(a) \in \mathbb{N}_+$, and a number $B \in \mathbb{N}_+$. Is there a subset $A' \subset A$ for which $\sum_{a \in A'} s(a) = B$? □

SUB is NP-complete, and it can be solved in pseudo-polynomial time [3].

Theorem 1. *PUC is NP-complete, and it can be solved in pseudo-polynomial time.*

Proof (sketch). First, for a given vector $\boldsymbol{i}$ one can verify in polynomial time whether it satisfies the constraints, so PUC $\in$ NP. Next, SUB can be reduced straightforwardly to PUC, by introducing for each $a_k \in A$ an iterator i_k with iterator bound $I_k = 1$ and period $p_k = s(a_k)$, and choosing $s = B$. This proves that PUC is NP-complete.

In a similar way, each instance of PUC can be transformed into an instance of SUB, so a pseudo-polynomial time algorithm for SUB can be used for solving PUC. □

The first special case of PUC that we consider is the case with divisible periods.

Definition 13 (PUCDP). Given are an iterator bound vector $\boldsymbol{I} \in \mathbb{N}_+^\delta$, a period vector $\boldsymbol{p} \in \mathbb{N}_+^\delta$, sorted in non-increasing order, with $p_{k+1} | p_k$ for all $k = 0, \ldots, \delta-2$, and a number $s \in \mathbb{N}_+$. Is there a vector $\boldsymbol{i} \in \mathbb{Z}^\delta$, $\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{I}$, for which $\boldsymbol{p}^{\mathrm{T}}\boldsymbol{i} = s$? □

Theorem 2. *PUCDP can be solved in polynomial time.*

Proof (sketch). The proof is based on the fact that if PUCDP has a solution, then the lexicographically maximal solution $\boldsymbol{i}^*$ satisfies

$$i_k^* = \min\{I_k, \lfloor (s - \sum_{l=0}^{k-1} p_l i_l^*) \,/\, p_k \rfloor\}, \tag{1}$$

for all $k = 0, \ldots, \delta-1$; see [12]. So, if a solution exists, then there is a lexicographically maximal one, and by using (1) it can be computed in polynomial time. □

The second special case of PUC is the case with a *lexicographical execution*, which means that we can identify an innermost loop, with the smallest period, which is com-

pletely executed within the next greater period. This next outer loop is then again completely executed within the next greater period, etc.

Definition 14 (PUCL). Given are an iterator bound vector $I \in \mathbb{N}_+^\delta$, a period vector $p \in \mathbb{N}_+^\delta$, and a number $s \in \mathbb{N}_+$. Furthermore, for all vectors $i,j \in \mathbb{Z}^\delta$ with $0 \leq i,j \leq I$ holds

$$i <_{\text{lex}} j \Leftrightarrow p^T i < p^T j,$$

where $<_{\text{lex}}$ stands for lexicographically smaller than. Is there a vector $i \in \mathbb{Z}^\delta$, $0 \leq i \leq I$, for which $p^T i = s$? □

Theorem 3. *PUCL can be solved in polynomial time.*

Proof (sketch). Again, if PUCL has a solution, then the lexicographically maximal solution i^* satisfies (1), for all $k = 0,\ldots,\delta - 1$. □

In the reformulation of PUC, the periods and iterator bounds of both operations have been combined into one new period vector and iterator bound vector. Although both operations may have a lexicographical execution, this does not guarantee that the new iterator bound vector and period vector also give a lexicographical execution. Therefore, we identify the following special case.

Definition 15 (PUCLL). Given are an iterator bound vector $I \in \mathbb{N}_+^\delta$, a period vector $p \in \mathbb{N}_+^\delta$, and a number $s \in \mathbb{N}_+$. Furthermore, p can be split into two vectors $p' \in \mathbb{N}_+^{\delta'}$ and $p'' \in \mathbb{N}_+^{\delta''}$, with $\delta' + \delta'' = \delta$, and I can be split accordingly into $I' \in \mathbb{N}_+^{\delta'}$ and $I'' \in \mathbb{N}_+^{\delta''}$, such that I' with p' gives a lexicographical execution, as well as I'' with p''. Is there a vector $i \in \mathbb{Z}^\delta$, $0 \leq i \leq I$, for which $p^T i = s$? □

Theorem 4. *PUCLL is NP-complete.*

Proof. Again, the proof is based on a reduction from the subset sum problem [12]. □

The fourth special case of PUC is the situation with two periods not equal to 1 and one period equal to 1. An example of this case follows from two one-dimensional periodic operations, for instance in one-dimensional periodic scheduling.

Definition 16 (PUC2). Given are three iterator bounds $I_0, I_1, I_2 \in \mathbb{N}_+$, two periods $p_0, p_1 \in \mathbb{N}_+$, with $p_0, p_1 \neq 1$, and a number $s \in \mathbb{N}_+$. Are there integer numbers i_0, i_1, i_2, with $0 \leq i_k \leq I_k$, $k = 0,1,2$, for which $p_0 i_0 + p_1 i_1 + i_2 = s$? □

Theorem 5. *PUC2 can be solved in polynomial time.*

Proof. For a proof, we again refer to [12]. □

4 Complexity of Checking the Precedence Constraints

In this section we analyze the computational complexity of checking the precedence constraints. To this end, we use a complementary formulation.

Definition 17 (precedence conflict (PC)). Given are an output port p that is connected to an input port q, and their iterator bound vectors $I(p) \in \mathbb{N}_\infty^{\delta(p)}$, $I(q) \in \mathbb{N}_\infty^{\delta(q)}$, their period vectors $p(p) \in \mathbb{Z}^{\delta(p)}, p(q) \in \mathbb{Z}^{\delta(q)}$, their start times $s(p), s(q) \in \mathbb{Z}$, their index matrices $A(p) \in \mathbb{Z}^{\alpha(p) \times \delta(p)}, A(q) \in \mathbb{Z}^{\alpha(q) \times \delta(q)}$, and their index offset vectors $b(p) \in \mathbb{Z}^{\alpha(p)}, b(q) \in \mathbb{Z}^{\alpha(q)}$. Are there vectors $i \in \mathbb{Z}^{\delta(p)}$ and $j \in \mathbb{Z}^{\delta(q)}$, $0 \leq i \leq I(p)$, $0 \leq j \leq I(q)$, for which

$$A(p)\,i + b(p) = A(q)\,j + b(q) \;\wedge\; p^T(p)\,i + s(p) \geq p^T(q)\,j + s(q)\,?$$ □

By combining all iterators into one vector, we can show that this problem is equivalent to the following problem [12].

Definition 18 (PC reformulated). Given are an iterator bound vector $I \in \mathbb{N}_+^\delta$, a period vector $p \in \mathbb{Z}^\delta$, an integer s, an index matrix $A \in \mathbb{Z}^{\alpha \times \delta}$ with lexicographically positive columns, and an index offset vector $b \in \mathbb{Z}^\alpha$. Is there a vector $i \in \mathbb{Z}^\delta$, $0 \leq i \leq I$, for which $A\,i = b$ and $p^T i \geq s$? □

To discuss the complexity of PC, we use the following problem [3].

Definition 19 (zero-one integer programming (ZOIP)). Given are a matrix $M \in \mathbb{Z}^{m \times n}$, a vector $d \in \mathbb{Z}^m$, a vector $c \in \mathbb{Z}^n$, and an integer B. Is there a vector $x \in \{0,1\}^n$ such that $M\,x = d$ and $c^T x \geq B$? □

Without loss of generality, we may assume that the columns in M are lexicographically positive. ZOIP is NP-complete in the strong sense [3].

Theorem 6. *PC is NP-complete in the strong sense.*

Proof (sketch). First, for a given vector i one can check in polynomial time whether it satisfies the constraints, so PC $\in$ NP. Next, ZOIP can be reduced to PC by choosing $\delta = n$, $I_k = 1$, for all $k = 0,\ldots,\delta-1$, $p = c$, $s = B$, $\alpha = m$, $A = M$, and $b = d$. □

The first special case of PC that we consider is the case with a *lexicographical index ordering*. This property is analogous to the property of a lexicographical execution.

Definition 20 (PCL). Given are an iterator bound vector $I \in \mathbb{N}_+^\delta$, a period vector $p \in \mathbb{Z}^\delta$, an integer s, an index matrix $A \in \mathbb{Z}^{\alpha \times \delta}$ with lexicographically positive columns, and an index offset vector $b \in \mathbb{Z}^\alpha$. Furthermore, for all vectors $i,j \in \mathbb{Z}^\delta$ with $0 \leq i,j \leq I$ holds

$$i <_{\text{lex}} j \;\Leftrightarrow\; A\,i <_{\text{lex}} A\,j.$$

Is there a vector $i \in \mathbb{Z}^\delta$, $0 \leq i \leq I$, for which $A\,i = b$ and $p^T i \geq s$? □

Theorem 7. *PCL can be solved in polynomial time.*

Proof (sketch). In a way similar to (1), one can determine the lexicographically maximal solution $i^* \in \mathbb{Z}^\delta$, if one exists. For details, see [12]. □

Analogously to PUCLL, we define the second special case of PC.

Definition 21 (PCLL). Given are an iterator bound vector $I \in \mathbb{N}_+^\delta$, a period vector $p \in \mathbb{Z}^\delta$, an integer s, an index matrix $A \in \mathbb{Z}^{\alpha \times \delta}$ with lexicographically positive columns, and an index offset vector $b \in \mathbb{Z}^\alpha$. Furthermore, the columns of A can be divided among two matrices $A' \in \mathbb{Z}^{\alpha \times \delta'}$ and $A'' \in \mathbb{Z}^{\alpha \times \delta''}$, with $\delta' + \delta'' = \delta$, and the entries of I can be divided accordingly among $I' \in \mathbb{N}_+^{\delta'}$ and $I'' \in \mathbb{N}_+^{\delta''}$, such that I' with A' gives a lexicographical index ordering, as well as I'' with A''. Is there a vector $i \in \mathbb{Z}^\delta$, $0 \leq i \leq I$, for which $A\,i = b$ and $p^T i \geq s$? □

Theorem 8. *PCLL is NP-complete in the strong sense.*

Proof (sketch). For this proof, we polynomially transform PC into PCLL. Let an instance $\mathcal{I}_{\text{pc}}$ of PC be given, then we define an instance $\mathcal{I}_{\text{pcll}}$ of PCLL as

$$A_{\text{pcll}} = \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ A_{\text{pc}} & \mathbf{O} \end{bmatrix},\; b_{\text{pcll}} = \begin{bmatrix} I_{\text{pc}} \\ b_{\text{pc}} \end{bmatrix},\; I_{\text{pcll}} = \begin{bmatrix} I_{\text{pc}} \\ I_{\text{pc}} \end{bmatrix},\; p_{\text{pcll}} = \begin{bmatrix} p_{\text{pc}} \\ \mathbf{0} \end{bmatrix},\; s_{\text{pcll}} = s_{\text{pc}}.$$

Here, $\mathbf{I}$ is the $\delta \times \delta$ identity matrix, and $\mathbf{O}$ is the $\alpha \times \delta$ zero matrix. Now, a solution i_{pc} of instance $\mathcal{I}_{\text{pc}}$ corresponds to a solution $i_{\text{pcll}} = \begin{bmatrix} i_{\text{pc}} \\ I_{\text{pc}} - i_{\text{pc}} \end{bmatrix}$ of $\mathcal{I}_{\text{pcll}}$. □

The third special case of PC is the case with only one index equation, i.e., $\alpha = 1$.

Definition 22 (PC1). Given are an iterator bound vector $I \in \mathbb{N}_+^\delta$, a period vector $p \in \mathbb{Z}^\delta$, an integer s, a vector $a \in \mathbb{N}_+^\delta$, and an index offset $b \in \mathbb{N}_+$. Is there a vector $i \in \mathbb{Z}^\delta$, $0 \leq i \leq I$, for which $a^T i = b$ and $p^T i \geq s$? □

To discuss the complexity of PC1, we use the knapsack problem [3].

Definition 23 (knapsack (KS)). Given are a set U, with for each $u \in U$ a size $s(u) \in \mathbb{N}_+$ and a value $v(u) \in \mathbb{N}_+$, and positive integers B and K. Is there a subset $U' \subset U$ for which $\sum_{u \in U'} s(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K$? □

KS is NP-complete, and it can be solved in pseudo-polynomial time [3].

Theorem 9. *PC1 is NP-complete, and it can be solved in pseudo-polynomial time.*

Proof (sketch). KS can be reduced straightforwardly to PC1, showing that PC1 is NP-complete. Similarly, each instance of PC1 can be transformed into an instance of KS, so a pseudo-polynomial time algorithm for KS can be used for solving PC1 [12]. □

The last special case of PC is the case with one index equation and divisible coefficients.

Definition 24 (PC1DC). Given are an iterator bound vector $I \in \mathbb{N}_+^\delta$, a period vector $p \in \mathbb{Z}^\delta$, an integer s, a vector $a \in \mathbb{N}_+^\delta$, sorted in non-increasing order, with $a_{k+1} | a_k$ for all $k = 0, \ldots, \delta - 2$, and an index offset $b \in \mathbb{N}_+$. Is there a vector $i \in \mathbb{Z}^\delta$, $0 \leq i \leq I$, for which $a^T i = b$ and $p^T i \geq s$? □

Theorem 10. *PC1DC can be solved in polynomial time.*

Proof. For a proof, we refer to [12]. □

5 Complexity of Multidimensional Periodic Scheduling

In this section we discuss the complexity of the multidimensional periodic scheduling problem in its entirety. To this end, we consider the decision variant MPSD of it, i.e., the question is whether there is a feasible schedule with cost not exceeding a given integer.

Theorem 11. *MPSD is NP-hard in the strong sense.*

Proof. The proof can be based on several reductions. First, one can show that MPSD is NP-hard by means of a reduction from PUCLL. However, this does not show NP-hardness in the strong sense. Second, one can reduce PC to MPSD, showing that MPSD is NP-hard in the strong sense. Third, one can give a reduction from strictly periodic single processor scheduling [5] which is NP-complete in the strong sense. For more details, see [12]. □

6 Towards a Solution Approach

In this section we present a sketch of a solution approach for multidimensional periodic scheduling. To this end, we first decompose the problem into two stages. In the first stage we assign period vectors to all operations and in the second stage we assign start times to the operations and assign the operations to processing units.

The main objective to be minimized in the first stage is the storage cost, subject to the timing and precedence constraints. In order to do so, we also have to determine preliminary start times, which may be altered in the second stage. The processing unit cost is not taken into account, since no processing unit assignment is determined. For similar reasons the access cost is not taken into account. The determination of periods is based on a linear programming approach. To this end, so-called stop operations are added which denote the ends of the variables' lifetimes, and the storage cost is estimated by a function that is linear in the periods and start times. Furthermore, a branch-and-bound technique is applied to find solutions that satisfy the non-linear constraints.

In the second stage, we opt for a resource and time constrained approach. So, we assume that the number of processing units of each type is given. Next, a start time and a processing unit assignment are determined, such that a feasible schedule is obtained. This is done based on integer linear programming (ILP) techniques for detecting processing unit and precedence conflicts, which are tailored towards the well-solvable special cases. The sizes of these ILP sub-problems are small since they only depend on the number of dimensions, which is usually smaller than 10, and not on the number of operations.

First, we determine the minimum difference in start times between each pair of operations, such that the precedence constraints are met. Together with the lower and upper bounds on the start times, we then determine an as soon as possible (ASAP) schedule and an as late as possible (ALAP) schedule.

Finally, we determine the start times and processing unit assignment by means of a list scheduling algorithm. A partial schedule is augmented by considering an unscheduled operation and trying to schedule it at its ASAP start time on a processing unit of the corresponding type. If this is possible, we add the operation to the schedule. If not, then we increase its ASAP value. The list scheduling algorithm terminates when all operations are scheduled, or when an operation has no alternative start times left.

The details of the approach sketched above can be found in [12]. The corresponding algorithms have been implemented in C++, and are incorporated in the design methodology Phideo [7]. In practice they are well applicable for finding good solutions in a reasonable amount of time, using them in an iterative and interactive way.

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. Feautier. Dataflow analysis of array and scalar references. *Int. Journal of Parallel Programming* **20** (1991) 23–53.

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

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

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

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

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

[8] 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.

[9] M. Potkonjak and J. Rabaey. A scheduling and resource allocation algorithm for hierarchical signal flow graphs. *Proc. 31st DAC*, 7–12, 1994.

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

[11] 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. EDAC*, 8–13, 1992.

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

[13] 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.