Storage Management in Process Networks using the Lexicographically Maximal Preimage

Alexandru Turjan Bart Kienhuis
Leiden Institute of Advanced Computer Science (LIACS),
Leiden, The Netherlands
e-mail: {aturjan,kienhuis}@liacs.nl

Abstract

At the Leiden Embedded Research Center, we are developing a compiler called Compaan that automatically translates signal processing applications written in Matlab into Kahn Process Networks (KPNs). In general, these signal processing applications are data-flow intensive, requiring large storage capacities, usually represented by matrices. An important issue in Compaan is the derivation of a memory management mechanism that allows for efficient inter-process communication. This mechanism has previously been published and is called the Extended Linearization Model (ELM). The controller needed in the ELM is derived using the Ehrhart theory, leading to a computational intensive procedure. In this paper, we present a new approach to derive the ELM controller, based on the notion of Lexicographically Maximal Preimage. Using polytope manipulations and parametric integer linear programming techniques, we get less computational intensive and easier to be derived controller implementation for the ELM.

1 Introduction

The *Compaan* tool-chain [9] automatically transforms digital signal processing applications, written in a subset of Matlab, into Kahn Process Networks [8]. These KPNs represent the input applications in a parallel distributed way, making them suitable for mapping onto parallel architectures. These networks can be converted to VHDL and quickly synthesized to FPGAs [7] or mapped onto some parallel signal processing architecture [11] at a high level of abstraction to obtain first-order performance numbers.

The process of converting a signal processing application to a Kahn Process Network is done in three steps. The first tool, i.e., *MatParser*, performs an array data-flow analysis that transforms Matlab code into single assignment code (SAC). The second tool, i.e., *DgParser*, converts the SAC into a mathematical model based on polytopes called polyhedral reduced dependence graphs (PRDG). Using polytope manipulations, the third tool, i.e., *Panda*, converts the PRDG into a process network (PN).

Given the three steps, Compaan abstracts a Matlab program into concurrent processes that communicate with each other via n-dimensional arrays. Each occurrence of this communication can be abstracted to an instance of the classical Producer/Consumer pair. If the Producer process writes data (into the array) and the Consumer process reads data (from the array) in the same order, then a FIFO between Producer and Consumer is enough in order to ensure a correct translation form Matlab to a Kahn Process Network. There can, however, be a problem when a token is needed more than once by the Consumer. As soon as a token is read from a FIFO by the Consumer process, it cannot be read by another iteration of the Consumer. Therefore, if another iteration needs the same information, it has to be send either another time by the Producer, or it has to be stored temporarily by the Consumer until all iterations that need to read the same token have taken place. This means that the token can be released from the local memory such that the corresponding location can be reused. If such situation occurs, we say that the Producer iteration has *multiplicity*; the token produced by that iteration is consumed by more than one Consumer iteration.

1.1 Multiplicity

To illustrate the concept of multiplicity, consider the simple Matlab program from the left part of Figure 1. In this program, the value `a(x)` is initialized with a particular value as given by the function `Fp()`. Next, the function `Fc()` reads the value of `a(j)`. In the first iteration, it reads `a(1)`, a single time. In the second iteration, it reads `a(2)` two times, in the third iteration it reads `a(3)` three times, and finally, it reads `a(4)` four times.

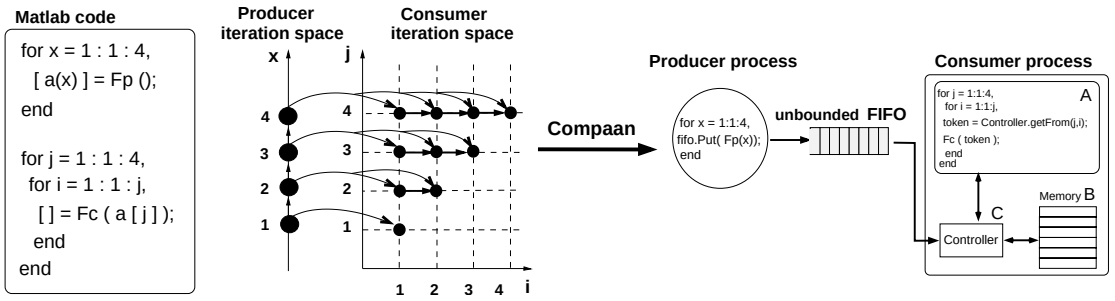

Figure 1. Matlab program with multiplicity large than one of variable `a`

This program can be abstracted to a simple Producer/Consumer pair that communicate variable `a`. The producer consists of the first for-loop with iterator `x`. The Consumer consists of the two for-loops with iterators `i` and `j`. If function `Fc` would read value `a(1)`, `a(2)`, `a(3)`, and `a(4)` only once, the array `a` could have been replaced by a FIFO. But, because variable `a` has a multiplicity larger than one, a FIFO no longer is enough. A read from the FIFO is destructive and therefore, as soon as it has been read, another iteration cannot read it again. This problem is specific to a Process Network; it does not exist in the original Matlab program in which function `Fc` can read the value `a(i)` from main memory as many times as needed as the read does not destruct the value in memory.

To handle the communication involving multiplicity, we have proposed in [15] the *Extended Linearization Model* (ELM) which adds to the Consumer process a piece of memory and a controller as visualized in the right part of Figure 1. Depending on the type of memory and on the functionality of the controller, four different realizations of the ELM have been proposed [17]. As soon as a token is read from the FIFO, it is stored in the local memory. All the iterations inside the consumer that require this token, can read this token as many times as needed from local memory. When no other iteration requires the token anymore, the memory location can be relinquished and reused by another token.

In this paper, we present an approach based on the *Lexicographically Maximal Preimage* to determine when to release the reordering memory or to read from the FIFO in a Compaan derived KPN. This approach allows the use of a simpler realization of the linearization model (LM or ELM) when handling multiplicity that is easier to be derived compared to the approach described in [15]. The usage of the adequate linearization model when linearizing a given P/C pair can be seen as a starting point in deriving the memory lower bound involved in communicating data over a KPN. The releasement of the memory was also addressed by the parallelizing compilers community when doing life time analysis for memory compression. There are several papers that are dealing with compile-time analysis of memory reuse in static nested loop programs. In [18], an approach is presented for a fixed linearization of the memory array. In [20, 13], a constructive approach is given in context of the single assignment language ALPHA, for finding the maximum lifetime of an array, based on which can be derived an optimal memory projection. In [10], in the process of parallelizing a static algorithm by removing output dependences, the authors propose a method of partial array expansion. In [3], an approach is presented to compute the bounding box for the elements that are simultaneously in use. The size of the original array is reduced to the bounding box and accessed by using modulo operations, improving in this way the memory usage. Although their techniques have similarities with the work presented in this paper, i.e., relay on polytope manipulations and integer linear programming, a new approach specific to the KPN involving distributed memory management had to be derived. The formulation of our problem consist of solving two parametric integer linear programming

(PIP) problems, formulated after solving the classical PIP problem used by Feautrier for doing array data flow analysis [6].

2 Kind of Communications

Consider an arbitrary P/C pair as derived by Compaan. According to the data dependencies that exists in the single assignment code, an affine transformation M exists that maps the polyhedral Consumer domain into the Producer domain. If restricted to the Consumer domain, the mapping M is not injective, then there are at least two Consumer iterations that consume the same token. Therefore, we say that the token was produced at an iteration that has the multiplicity bigger than one. Here we give the formal definition of the multiplicity:

Definition 1 *Consider a parameterized polyhedral domain $\mathcal{C}(p) \subset \mathrm{R}^n$ mapped through an affine transformation M into a parameterized polyhedral domain $\mathcal{P}(p) = M(\mathcal{C}(p))$, where $\mathcal{P}(p) \subset \mathrm{R}^m$. The multiplicity is a function $E : (\mathcal{P}(p) \cap \mathrm{Z}^m) \rightarrow \mathrm{Z}^*$ that associates to each integral point $x \in \mathcal{P}(p)$ the number of the integral points in $\mathcal{C}(p)$ that are mapped through M into the same point x.*

According to the previous definition, for an integral point $x \in (\mathcal{P}(p) \cap \mathrm{Z}^m)$, its multiplicity depends on the mapping function M and on the $\mathcal{C}(p)$ domain.

We define the multiplicity of a produced token t as the multiplicity of the iteration point (IP) which produces t. Let $x \in (\mathcal{P}(p) \cap \mathrm{Z}^m)$ be a producer IP that produces the token t. Assume the Consumer domain being represented by the parameterized polytope $\mathcal{C}(p) \in \mathrm{R}^n$, then the set of the consumer's IPs that consume t is represented by the integer points inside the next parameterized polytope:

$$\mathcal{K}(x,p) = \{y \in \mathrm{R}^n \mid y \in \mathcal{C}(p) \wedge x \in M(\mathcal{C}(p)) \wedge x = M(y)\}, \tag{1}$$

such that the multiplicity of x is given by:

$$E(x,p) = Card\{\mathcal{K}(x,p) \cap \mathrm{Z}^n\}, \tag{2}$$

which is a pseudo-polynomial expression, as given by the Ehrhart theory [4, 1].

Depending on M, and on the schedule of producing and consuming data, four different kinds of P/C pairs (dependence-graphs) can be distinguished:

- ***in-order:*** A P/C pair is in-order if and only if every two consecutive Consumer iteration points $x_1 \prec_{lex} x_2$ are mapped onto two consecutive Producer iteration points $(y_1 = Mx_1)$ and $(y_2 = Mx_2)$ such that $y_1 \prec_{lex} y_2$.
- ***out-of-order:*** A P/C pair is out-of-order if and only if restricted to the Consumer domain the mapping M is injective and there are at least two consecutive Consumer iteration points $x_1 \prec_{lex} x_2$ that are mapped onto two Producer iteration points $(y_1 = Mx_1)$ and $(y_2 = Mx_2)$ such that $y_2 \prec_{lex} y_1$.
- ***in-order with multiplicity:*** A P/C pair is in-order with multiplicity if and only if restricted to the Consumer domain the mapping M is not injective and every two consecutive Consumer iteration points $x_1 \prec_{lex} x_2$ are mapped onto the consecutive Producer iteration points $(y_1 = Mx_1)$ and $(y_2 = Mx_2)$ such that $y_1 4_{lex} y_2$.
- ***out-of-order with multiplicity:*** A P/C pair is out-of-order with multiplicity if and only if restricted to the Consumer domain the mapping M is not injective and it exist two consecutive Consumer iteration points $x_1 \prec_{lex} x_2$ that are mapped onto the Producer iteration points $(y_1 = Mx_1)$ and $(y_2 = Mx_2)$ such that $y_2 \prec_{lex} y_1$.

In Figure 2, the four different kinds of P/C pairs are given by showing the Producer and Consumer iteration spaces and the order of their iterations. From this picture, it is clear that the program given in Figure 1 corresponds to an instance of the *in-order with multiplicity* P/C pair. If we would have exchanged the indices of the j and i, it would become an instance of the *out-of-order with multiplicity*. Each P/C pair belongs to one of the four categories; each category replaces the n-dimensional array in the original Matlab program with an optimal linearization model:

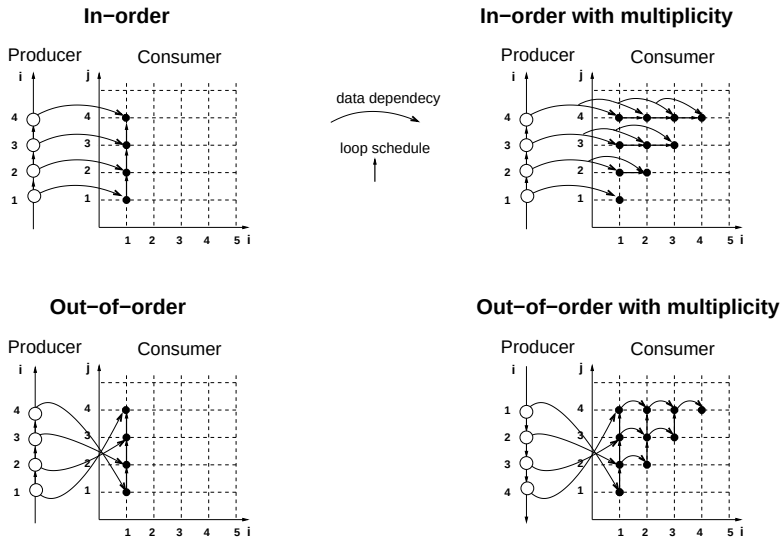

Figure 2. The four kinds of Producer/Consumer pairs

- **Model 1.** An in-order P/C pair is linearized using only a FIFO buffer.
- **Model 2.** An in-order with multiplicity P/C pair is linearized using a FIFO buffer and additional control statements. No Controller and reordering memory is needed.
- **Model 3.** An out-of-order P/C pair is linearized using a FIFO buffer, reordering memory, and a controller. In this model, each time when the controller addresses the reordering memory for reading data, the corresponding memory location can be released.
- **Model 4.** An out-of-order with multiplicity P/C pair is linearized using a FIFO buffer, a reordering memory, and a reordering controller.

To give a feeling how often a particular kind of communication occurs, we looked at the set of benchmark programs used in the development of Compaan. Looking at 26 algorithms containing in total 1435 P/C pairs, we found that 80% of the pairs represent in-order communication that can be handled by a FIFO (Model 1). In 17% of the cases, we found that the communication represent out-of-order communication (Model 3). In 3% of the cases, the communication involves a multiplicity larger then one. In 2.7% of the cases, the communication is in-order with multiplicity (Model 2) and in .6% of the cases. the communication is out-of-order with multiplicity (Model 4).

We remark that the implementations of the linearization models described above increase in their complexity, from **Model 1** to **Model 4**. Model 1 and Model 2 are closely related, except that some control logic is needed to know when to release data in Model 2. Model 3 and 4 requires additional reordering memory. Of the four models identified, Model 4 is the most expensive linearization to be realized, being able to linearize all four possible P/C pairs. As we see from the data, this model is used quite rarely.

To reduce implementation cost it is very important to select adequate linearization model for each P/C pair. Such a selection procedure has been presented in [16] and is realized in Compaan. The general design of ELMs has been presented in [17], but in this paper we indicate that such an ELM can be optimized according to the four linearization models given. In the remainder of this paper, we focus on a technique to determine the control needed to handle multiplicity.

3 Problem Definition

Consider again a P/C pair, but now with multiplicity. This implies that the mapping is not injective, i.e., there exists a token t produced by an iteration point p of the Producer process which is consumed by more than one iteration point of the Consumer process. Therefore, the problem we address in this article, can be

stated as follows:

- *Find a procedure that determines for each Consumer's IP whether the token that it uses will be needed by any of the IPs executed afterwards. If such moment can be found, we know when to release the memory location taken by the token.*

This problem can be solved using the Ehrhart theory by computing the multiplicity of a token (See Equation 2). For example, the multiplicity for the P/C pair given in Figure 1, is a simple polynomial $E(x) = x$. In general, however, the multiplicity function can be rather complex and hard to compute as the following example taken from [14] shows:

Example of deriving the multiplicity: Assume a parameterized Producer iteration space given by the integer polyhedral image $\mathcal{P} = M(\mathcal{C}(N) \cap \mathrm{Z}^2)$, where the parameterized Consumer iteration space is given by $\mathcal{C}(N) = \{(k_1, k_2) \in \mathrm{Z}^2 \mid 0 \leq k_1 \leq N \ \wedge \ k_1 \leq k_2 \leq N\}$ and $M(k) = 2k_1 + k_2 + 3$. Figure 3 shows the Producer and the Consumer iteration spaces. To find the multiplicity of the Producer IPs we have to count the number of integer points inside the parameterized polyhedron $\mathcal{K}(N, j) = \{(k_1, k_2) \in \mathrm{R}^2 \mid 0 \leq k_1 \leq N \ \wedge \ k_1 \leq k_2 \leq N \wedge j = 2k_1 + k_2 + 3\}$. Using the Ehrhart theory, the multiplicity is given by the next parameterized pseudo-polynomial expression:

$$E(N,j) = \begin{cases} \frac{1}{3}j + [0, -\frac{1}{3}, -\frac{2}{3}]_j & \text{if } 3 \leq j \leq N+3, \\ \frac{1}{2}N + [-\frac{1}{6}j + [1, \frac{7}{6}, \frac{1}{3}, \frac{3}{2}, \frac{2}{3}, \frac{5}{6}]_j, -\frac{1}{6}j + [\frac{3}{2}, \frac{2}{3}, \frac{5}{6}, 1, \frac{7}{6}, \frac{1}{3}]_j]_N & \text{if } N+3 \leq j \leq 3N+3, \\ 0 & otherwise. \end{cases} \tag{3}$$

In this case, when a token is read from a FIFO, a counter in the Controller is set to the value of the multiplicity of the token. This value is obtained by evaluating the pseudo-polynomial that gives the multiplicity. Now, each time this token is read by a Consumer's IP, the counter is decremented by one. When the value of the counter reaches zero, the memory entry can be released by the Controller. □

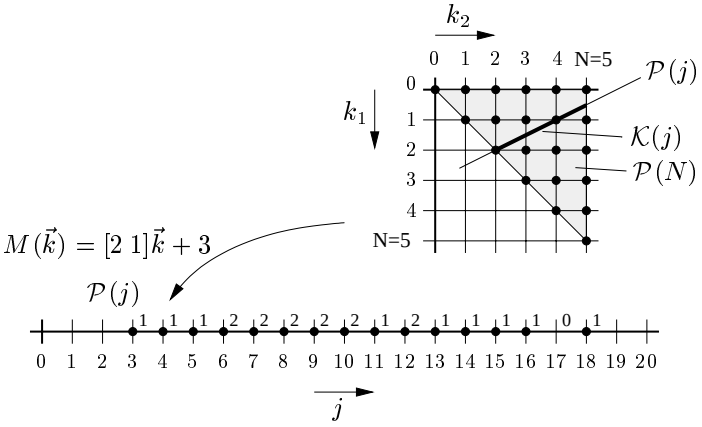

Figure 3. Producer/Consumer pair with non-dense Producer domain

We relay on the PolyLib library [2, 19] for computing the multiplicity using Ehrhart. This library is still under development and as a consequence, there are still cases for which PolyLib cannot derive an enclosed expression for the multiplicity function. Therefore, we present a new approach that provides a more efficient solution to the problem stated above and a solution that can be used in many more occasions.

4 Solution

The problem we face, is to find when a token can be released from the local memory or FIFO, so that it can be reused to store the next token. We now make the following observation. In Figure 1, we notice that all iterations where a token is used last are represented by the next set: $C' = \{(i,j) \in \mathrm{Z}^2 \mid 0 \leq i \leq 4 \wedge j = i\}$. So, given we can find C', we can test whether an iteration is in C' or not. If an iteration is in C', it means the memory location associated ith that iteration can be released, otherwise other iterations still require the data at the memory location. As we will show, by computing the *lexicographically maximal preimage* (LMP), we are able to find C'. We will explain now the notion of the LMP in a more systematic way.

Consider a parameterized polytope $\mathcal{C}(p) \in \mathrm{R}^n$ and an integral affine transformation $M()$ represented by a $m \times n$ matrix M and an offset vector s. The integral image of $\mathcal{C}$ through the function M represents a *linearly bounded lattice (LBL)*:

$$P = M(\mathcal{C} \cap \mathrm{Z}^n) = \{i \mid i = Mk + s \wedge \ k \in (\mathcal{C} \cap \mathrm{Z}^n)\} \tag{4}$$

For each $y \in M(\mathcal{C} \cap \mathrm{Z}^n)$ consider the set $M^{-1}(y) = \{x \in \mathcal{C} \cap \mathrm{Z}^n | M(x) = y\}$. Let $f(y)$ denote the lexicographically maximal element in the set $M^{-1}(y)$. This element will be referred to as the *lexicographically maximal preimage* of y. Consider the sub-domain $C' \subseteq \mathcal{C}$ which consists of all maximal preimages: $C' = \{f(y) | y \in M(C \cap \mathrm{Z}^n)\}$. We say that C' is the *Lexicographically Maximal Preimage* of polytope C through affine mapping M, which is a LBL.

If $\mathcal{C}$ represents the consumer domain and M is the mapping of the P/C pair, the set C' has the following meaning. Let t be a token produce by the producer IP $p \in M(C \cap \mathrm{Z}^n)$. Then C' contains $f(p)$, the lexicographically maximal consumer IP that consumes t. This means that $f(p)$ is the last IP consuming t. C' contains all such last consuming IPs, and only them. Hence, if a consumer IP belongs to C', then the memory location holding the token it uses can be released after the token is read by this IP.

If the current IP is in C', the memory can be released, otherwise not. The problem of finding the C' domain is a parametric integer linear programming (PILP) problem that can be solved using tools like PipLib [5] or Omega [12]. We define the polytope $\mathcal{P}$ as the image of $\mathcal{C}$ through the mapping M:

$$\mathcal{P} = M(\mathcal{C}) = \{i \mid i = Mk + s \wedge \ k \in \mathcal{C}\}. \tag{5}$$

Let x be an arbitrarily integer point: $x \in \mathcal{C} \wedge Z^n$. Consider $y = M\ x$ as the image of x through the mapping matrix M into the $\mathcal{P}$ domain. Therefore, $y \in \mathcal{P}$. Now we formulate the next PILP problem. Consider x variable and y parameter determine, x_m that:

$$\begin{aligned} \textbf{subject to:} \quad & \mathrm{x} \in \mathcal{C}, \\ & y \in \mathcal{P} = M(\mathcal{C}), \\ & y = M\ \mathrm{x}, \\ \textbf{objective:} \quad & \mathrm{x}_m = max_{\mathbf{lex}}\{\mathrm{x}(y)\}. \end{aligned} \tag{6}$$

The solution of this PILP problem is parameterized in y and is represented by a multistage conditional expression:

$$\begin{aligned} &\text{if (y} \in \ \mathcal{P}_1), \\ &\qquad \text{then} \ \ \mathrm{x} = \ F_1(y), \\ &\text{if (y} \in \ \mathcal{P}_2), \\ &\qquad \text{then} \ \ \mathrm{x} = \ F_2(y), \\ &\qquad \vdots \\ &\text{if (y} \in \ \mathcal{P}_n), \\ &\qquad \text{then} \ \ \mathrm{x} = \ F_n(y). \end{aligned} \tag{7}$$

where $\mathcal{P}_1, ..., \mathcal{P}_n$ are parameterized polytopes and $F_1, ..., F_n$ are affine transformations. Some function can be nil (or empty), represented as $F_i = \perp$, which corresponds to the case when integer points of P have

no corespondent points in $\mathcal{C}$, i.e., M is not surjective. Such case is given for example in Figure 3, by the Producer IP equal to $j = 17$. Consider now $\bar{P}$ as the union of polytopes $\mathcal{P}_i$ for which functions Fi are defined:

$$\bar{P} = \bigcup_{i=1}^{n} \mathcal{P}_i, \text{ where } F_i \neq \perp .$$

By mapping P_i through the correspondent mapping F_i, we get the next LBL domain restricted to the $\mathcal{C}$ polytope:

$$C_i = \mathcal{C} \cap F_i(\mathcal{P}_i \cap \mathrm{Z}^m) = \{v \in \mathcal{C} \mid v = F_i(t) \wedge \ t \in (\mathcal{P}_i \cap \mathrm{Z}^m)\}. \tag{8}$$

Then the union of those LBLs represents the lexicographically maximum preimage of $\mathcal{C}$ through the function M:

$$C' = \bigcup_{i=1}^{n} C_i. \tag{9}$$

Now the problem is how one can represent in a systematic way an arbitrary LBL. Basically, one has to formulate a PILP problem similar to the previous one (see relation 6) which has a solution represented by a multistage conditional from which only the non-empty branches play role:

$$\begin{aligned} \textbf{subject to:} \quad & \mathrm{t} \in \mathcal{P}_i, \\ & v \in \mathcal{C} \cap F_i(\mathcal{P}_i), \\ & v = F_i(\,\mathrm{t}), \\ \textbf{objective:} \quad & \mathrm{t}_m = max_{\mathbf{lex}}\{\mathrm{t}(v)\}. \end{aligned} \tag{10}$$

Therefore, if a Consumer iteration point satisfies one of the non-empty branches corresponding to one of the PILP problems derived above, then the correspondent memory location from where the data is read can be released.

5 Realizing communication with multiplicity

In this section we show how the presented technique can be used to simplify the controller for two ELM models: Model 2 and Model 4.

5.1 Out-of-order with multiplicity - linearization Model 4

Consider now a general out-of-order with multiplicity example with the iteration spaces of the producer and consumer as given in Figure 3:

```
for j = 0 : 1 : 3*N+3,
    a[j] = Fp();
end
for k1 = 0 : 1 : N,
  for k2 = k1 : 1 : N,
    Fc( a[2*k1+k2+3] );
  end
end
```

In this case, we have the Consumer domain that corresponds to the function Fc described by the integer points inside the next parameterized polytope:

$$\mathcal{C}(N) = \{(k_1, k_2) \in \mathrm{R}^2 \mid 0 \leq k_2 \leq N \wedge k_1 \leq k_2 \leq N\}, \tag{11}$$

such that by mapping through the affine transformation $f(k_1, k_2) = 2k_1 + k_2 + 3$, the next domain[1] results:

$$\mathcal{P} = \{x \in \mathrm{R} \mid 3 \leq x \leq 3N + 3\} \tag{12}$$

[1]Note that $\mathcal{P}$ does not represent the Producer domain which is in this case a LBL bounded by $\mathcal{P}$

Therefore we have to solve the next PILP problem:

$$\begin{aligned} \textbf{subject to:}\quad & 0 \le k_1 \le N, \\ & k_1 \le k_2 \le N, \\ & 3 \le \mathrm{j} \le 3N+3, \\ & \mathrm{j} = 2k_1 + k_2 + 3, \\ \textbf{objective:}\quad & max_{\mathbf{lex}}\{(k_1, k_2)\}. \end{aligned} \tag{13}$$

The solution of this PILP problem is given by the next solution tree that has only one non-empty branch:

```
1. if (0 <= j-3),
2.     if(0 <= -j+3*N+3),
3.       _d18 = div(2*j,6),
4.           if(0 <= -j+N+2*_d18+1),
5.               (k1,k2)=(_d18-1,j-2*_d18-1);
6.           else
7.               Nil;
8.           end
9.     else
10.       Nil;
11.    end
12. else
13.     Nil
14. end.
```

This solution tree is the one-dimensional LBL that represents the Producer domain as a two dimensional polytope with coordinates j and d_{18}. As you can observe in Figure 4, for $N = 5$ the Producer iteration $j = 17$ is filtered out such that the data is not sent to the FIFO channel:

$$\mathcal{P}(j, d_{18}) = \{(j, d_{18}) \in Z^2 \mid 3 \le j \le 3N+3 \ \wedge \ 0 \le 2j - 6d_{18} \le 5 \ \wedge \ 0 \le -j + 2d_{18} + N + 1\}. \tag{14}$$

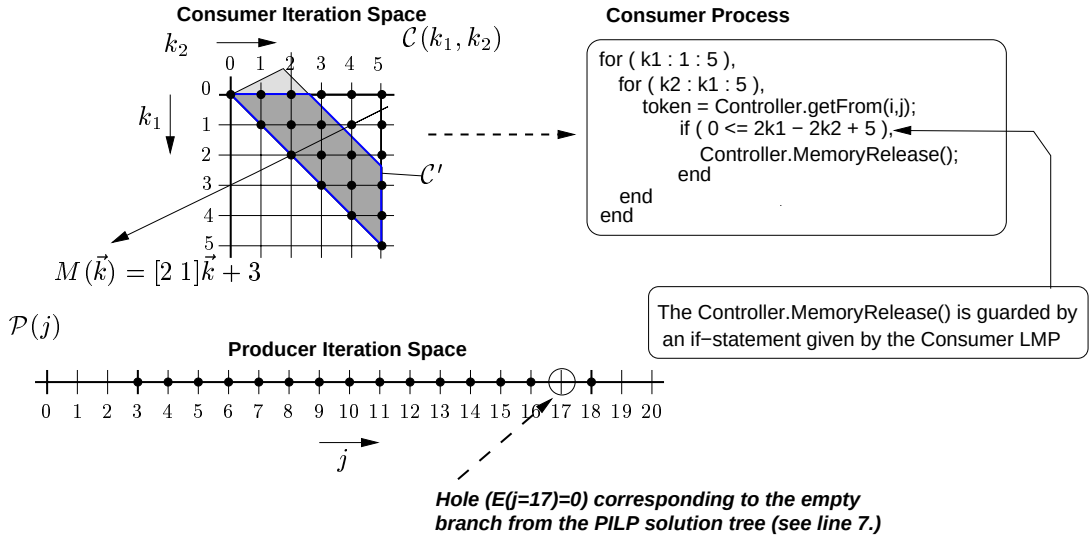

Figure 4. Linearization based on the *Lexicographically Maximal Preimage* for an out-of-order with multiplicity Producer/Consumer pair

As can be seen in the pseudo-code, line 5 indicates the mapping function $f(j, d_{18}) = (d_{18} - 1, j - 2d_{18} - 1)$. By mapping $\mathcal{P}$ through f we get the lexicographically maximum preimage domain which is

represented by the next polytope parameterized in N as can be seen in Figure 4:

$$\mathcal{C}'(N) = \{(k_1, k_2) \in Z^2 \mid 0 \leq -k_1 + k_2 \ \wedge \ k_2 \leq N \ \wedge \ 0 \leq 2k_1 - 2k_2 + 5 \ \wedge \ 0 \leq 2k_1 + k_2\} \quad (15)$$

In general the C' domains is not included in C ,i.e. $C' * C$, but as you can see in Figure 4 the domain difference between C' and C does not contain any integer points. Hence, although C' is defined by four inequalities, three out of four are redundant with the boundaries of the Consumer for-loops and only one inequalities results to determine the release moment. It is interesting to compare the difference between the solution using the Ehrhart theory and the solution found using the LMP. In the latter, a simple if-statement with a linear-expression results.

5.2 In-order with multiplicity - linearization Model 2

Consider the in-order P/C pair with the iteration spaces given in Figure 1. The Consumer domain is represented by the integer points inside the polytope:

$$\mathcal{C} = \{(i, j) \in \mathrm{R}^2 \mid 1 \leq i \leq 4 \wedge 1 \leq j \leq 4 \wedge i \leq j\}. \quad (16)$$

The affine transformation $f(x, y) = x$ maps it onto the following polytope:

$$\mathcal{P} = \{x \in \mathrm{R} \mid 1 \leq x \leq 4\}. \quad (17)$$

Suppose that the P/C is linearized according to Model 2. Consider a producer's iteration point x. If the

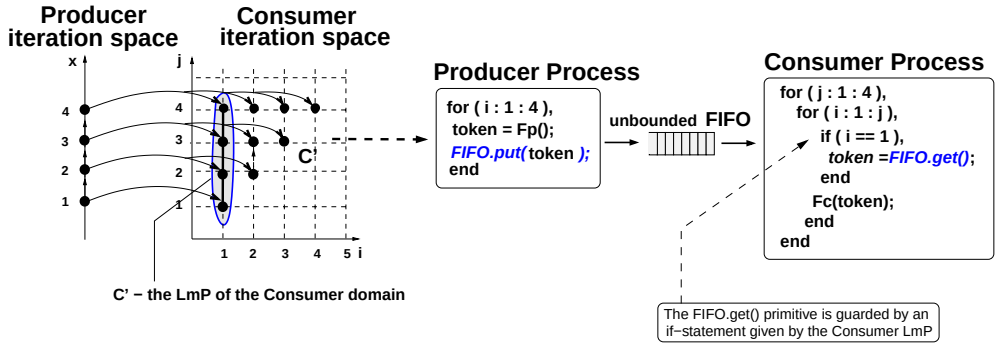

Figure 5. Linearization based on the *Lexicographically Minimal Preimage* for an in-order with multiplicity Producer/Consumer pair

proposed method is applied directly, we obtain the consumer IP which is the lexicographically maximal preimage (LMP) of x: $(i, j) = LMP(x)$, i.e., the last consumer IP (i, j) which consumes x. However, the P/C pair is linearized according to Model 2 and, hence, no reordering memory locations are used and have to be released. Instead, the tokens are consumed only from FIFO at certain moments. Instead of calculating the LMP, we calculate the *Lexicographically Minimal Preimage* (LmP). We follow the approach presented in Section 4, but we changed the objective function in Equation 6 into $\mathrm{x}_m = min_{\mathbf{lex}}\{\mathrm{x}(y)\}$. The meaning of Lexicographically Minimal Preimage is that each time a Consumer iteration arrives inside the Lexicographically Minimal Preimage, data is read from the FIFO for the first time. For the considered P/C pair, computing $LmP(x)$ is equivalent to solving the following PILP problem:

$$\begin{aligned} \textbf{subject to:} \quad & 0 \leq \mathrm{i} \leq 4, \\ & 0 \leq \mathrm{j} \leq 4, \\ & \mathrm{i} \leq \mathrm{j}, \\ & x = \mathrm{j}, \\ \textbf{objective:} \quad & min_{\mathbf{lex}}\{(i, j)\}. \end{aligned} \quad (18)$$

The solution of this PILP problem is given by the next solution tree:

```
1.  if(0 <= -x+4),
2.     if( 0 <= x-1),
3.         (i,j)=(1,x);
4.     end
5.  end
```

This solution tree is represented by a single context polytope $\mathcal{P}_1 = \{x \in \mathrm{Z} \mid 1 \leq x \leq 4\}$ and the correspondent affine function $F_1(x) = (1, x)$ (see line 3 from the previous pseudo-code). By mapping $\mathcal{P}_1$ through the function F_1 we get the LmP domain given by the next polytope domain:

$$\mathcal{C}' = \{(i, j) \in \mathrm{R}^2 | 1 \leq j \leq 4 \wedge i = 1\}, \tag{19}$$

such that the second PILP problem (see relation 10) does not have to be solved. As can be seen in Figure 5, the moment when the data has to be read from the FIFO can be check with a single if-statement that represents $\mathcal{C}'$ restricted to the Consumer domain.

6 Conclusions and Future Directions

When deriving a KPN from a given Matlab program using Compaan, in about 10% of all the communications between a Producer and a Consumer process, the communication involves multiplicity. That is communication in which the token produced by an iteration is consumed by more than one Consumer iterations. In these cases, an ELM is needed instead of a FIFO buffer to realize the correct communication. The Controller of the ELM needs additional code to determine when a particular memory location can be released, to use the local memory as efficient as possible.

To derive the additional control, we presented in this paper a novel technique that calculates a domain that indicates that a token is used for the last time and therefore a memory location can be release. The ELM Controller only needs to check if an iteration is in or out of the domain to make the correct decision. The domain is represented by a linearly bounded lattice that is calculated by solving a parametric integer linear program problem that gives the Lexicographically Maximal Preimage. This procedure is compared to the approach based on computing the multiplicity using Ehrhart's theory. We showed two cases in which the Lexicographically Maximal Preimage was calculated; an in-order with multiplicity and an out-of-order case with multiplicity. This shows the applicability of the new procedure.

The presented technique is used to solve the releasing of a token from memory. The technique can also be used to do domain reconstruction of the output port in the single assignment code generated by MatParser. This way, we know for sure that data send over a FIFO will be consumed, leading to efficient communication. Furthermore, the approach to derive the Lexicographically Maximal Preimage can easily be reformulated to get the Lexicographically Minimal Preimage. Using these two notion of the maximum and minimum, one could find the first and last use of a token, thereby giving the life time of a token. This fact could, for example, be exploited to minimize local memory usage at the consumer process. Finally, we remark that the complete procedure is not yet fully implemented in Compaan, but, all infrastructure to implement the presented procedure is available within Compaan as should be realized in the near future.

References

[1] Philippe Clauss and Vincent Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing*, 19:179–194, July 1998.

[2] Philippe Clauss and Vincent Loechner. Polylib. http://icps.u-strabg.fr/Polylib, February 2002.

[3] E De Greef, F Catthoor, and H De Man. Memory size reduction through storage order optimization for embedded parallel multimedia applications. In *Parallel Processing and Multimedia*, Geneva, Switzerland, July 1977.

[4] Eugène Ehrhart. *Polynômes arithmétiques et Méthode des Polyédres en Combinatoire*. Birkhäuser Verlag, Basel, international series of numerical mathematics vol. 35 edition, 1977.

[5] P Feautrier. Parametric integer programming. In *RAIRO Recherche Oprationnelle, 22(3):243-268*, 1988.

[6] Paul Feautrier. Dataflow Analysis of Scalar and Array References. *Int. Journal of Parallel Programming*, 20(1):23–53, 1991.

[7] Tim Harriss, Richard Walke, Bart Kienhuis, and Ed Deprettere. Compilation from matlab to process networks realized in fpga. *Design automation of Embedded Systems*, 7(4), November 2002.

[8] Gilles Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*. North-Holland Publishing Co., August 5-10 1974.

[9] Bart Kienhuis, Edwin Rijpkema, and Ed F. Deprettere. Compaan: Deriving process networks from matlab for embedded signal processing architectures. In *8th International Workshop on Hardware/Software Codesign (CODES'2000)*, San Diego, USA, May 2000.

[10] Vincent Lefebvre and Paul Feautrier. Automatic storage management in paralel programs. volume 24, pages 649 – 671. Parallel Computing, nov 1998.

[11] Paul Lieverse, Todor Stefanov, Pieter van der Wolf, and Ed Deprettere. System level design with spade: an m-jpeg case study. In *proc. Int. Conference on Computer Aided Design (ICCAD'01)*, pages 31 – 38, San Jose CA, USA, November 4 – 8 2001.

[12] W. Pugh. The omega test: A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.

[13] Fabien Quillere and Sanjay Rajopadhye. Optimizing memory usage in the polyhedral model. In *ACM Transactions on Programming Languages and Systems*, volume 22, pages 773–815, September 2000.

[14] Edwin Rijpkema. Modeling task level parallelism in piece-wise regular programs, September 2002. PhD thesis, Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands.

[15] Alexandru Turjan, Bart Kienhuis, and Ed Deprettere. A compile time based approach for solving out-of-order communication in Kahn Process Networks. In *proc. of 13th Int. Conference on Aplication-specific Systems, Architectures and Processors (ASAP'2002)*, San Jose, CA, USA, July 17-19 2002.

[16] Alexandru Turjan, Bart Kienhuis, and Ed Deprettere. A technique to determine inter-process communication in the polyhedral model. In *proc. Int. Workshop on Compilers for Parallel Computers*, January 2002.

[17] Alexandru Turjan, Bart Kienhuis, and Ed Deprettere. *Realizations of the Extended Linearization Model*. Marcel Dekker, Inc., Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation edition, 2003.

[18] V Vanhoof, I Bolsens, and Hugo De Man. Compiling multi-dimensional data streams into distributed dsp asic memory. In *In Proc. IEEE Int. Conf. Comp. Aided Design*, Santa Clara, CA, August 1989.

[19] Doran Wilde. A library for doing polyhedral operations. In *Technical Report PI 785, IRISA, Rennes, France*, 1993.

[20] Doran Wilde and Sanjay Rajopadhye. Memory reuse in the polyhedral model. In *In Proc. Euro-Par96*, Lyon, France, August 2002.