Parallelizing dynamic sequential programs using polyhedral process networks

Nadezhkin, D.

Chapter 5

Identifying Communication Models in Polyhedral Process Networks derived from Dynamic Programs

In Section 1.1, we have demonstrated that the Linearization is an important step of the parallelization approach depicted in Figure 1.2(b). As a result of the Dependence Analysis step, the initial program is translated into its functionally equivalent Polyhedral Reduced Dependence Graph (PRDG). The storage structure of the initial application is transformed such that each pair of statements communicates data over a dedicated multidimensional memory array as shown in Figure 1.2(c). However, the target model, Polyhedral Process Network (PPN), requires FIFO channel as communication medium. Therefore, the Linearization step converts such memory accesses into managed dataflow over FIFO queues.

Mapping array communication onto FIFO channels requires complex address generators, especially if the arrays have multiple dimensions. Therefore, the Linearization also solves the Communication Model Identification (CMI) problem, which investigates communication characteristics of each Producer/Consumer (P/C) pair.

In Section 1.1.2, we have demonstrated that there are four possible communication models that can describe the dataflow in a P/C pair. They are: *in-order (IO)*, *out-of-order (OO)*, *in-order with multiplicity (IOM)* and *out-of-order with multiplicity (OOM)*. Also, we have shown that the realization of different communication models requires different utilization resources (such as memory) and produces different runtime overhead. The difference in realization puts the communication models into a

hierarchy: from the most general *OOM* which can realize all communication models but requires most of the resources, to the specific, for example, *IO*, which can be realized as a FIFO in as straightforward way. Therefore, the identification procedure solves the optimization problem by finding the most optimal realization for each P/C pair.

In Section 2.6, in Definitions 2.6.1 and 2.6.3 we gave the overview of the Reordering and Multiplicity problems [14] which are used to identify communication models while translating *static* affine nested loop programs into functionally equivalent PPNs. In this chapter we present our novel compile-time procedure for communication model identification while translating the *dynamic* programs defined in Section 2.2 into functionally equivalent PPN. This procedure is used in the two parallelization approaches we have presented in Chapter 3 and Chapter 4.

In the following Section 5.1 we give the rationale of the communication model identification approach which will be presented in Section 5.2. Section 5.3 presents the conclusions.

5.1 Rationale

The overall challenge of Communication Model Identification while deriving a PPN from a dynamic program is how to deal with uncertainties introduced by the relaxations presented in Section 1.2. In Section 1.3 we have demonstrated that the approach used for CMI in static programs is inapplicable for dynamic programs.

In this dissertation, we use the PPN model of computation to specify both static and dynamic programs in parallel form. Therefore, the communication models in PPNs derived from dynamic programs are the same as in PPNs derived from static programs. These models are presented in Section 1.1.2.

Because the communication models in PPNs derived from static and dynamic programs are the same, we could have assumed that in order to identify the communication model in a P/C pair derived from a dynamic program we can use the RP and MP presented in Section 2.6. This problems are formulated according to Definitions 2.6.1 and 2.6.3 which give the formal definitions of ordering and multiplicity in a P/C pair. The key elements of these definitions are the mapping functions which are used to describe the dependency relations in a P/C pair. The definition of a mapping function is given in Definition 2.3.5.

However, there is a big difference in dependency relations between dynamic and static programs. In static programs, different instances of a program correspond to one and the same single dependency pattern which is known at compile-time. Therefore, only one unique set of mapping functions exist for all P/C pairs derived from a static program. We can observe this in Figure 1.3(b). In contrast to static programs, in dynamic programs, data dependency patterns correspond to different *instances* of a dynamic program, and are unknown at compile time. This has been demonstrated in Figure 1.6 explained in Section 1.3. Figure 1.6 depicts data dependency relations

between statements S1, S2 and S3 in three instances of the dynamic program in Figure 1.5(a) for $M = N = 4$. An instance of a dynamic program is an evaluation of the program with a particular input dataset. Figure 1.6 illustrates iteration domains of statements S1, S2 and S3, where the points on the coordinate systems designate the evaluations of statements and the arrows reflect the data dependencies between evaluations. The numbers at the points show the lexicographical order of statement evaluations.

Similar to static programs, from this figure it can be seen that for every instance of a dynamic program there is a unique set of mapping functions that determine the dependency relations between all statements for evaluation of the program with particular input data set. However, for different instances of a dynamic programs, there exist different sets of mapping functions that compensate for unknown, at compile-time, information inherent to dynamic programs.

To capture all unknown information at compile-time our novel approach is to define and use *parameterized* mapping functions that can be used to describe all possible dependency relations that exist in all instances of a dynamic program. By using these parameterized mapping functions, it is possible to analytically identify the most general communication model of a P/C pair in all possible instances of a dynamic program. Based on this information, the communication of a P/C pair can be realized with the most general communication model which implements all possible data dependency patterns occurring in all instances of a dynamic program.

In the following section we will demonstrate how these parameterized mapping functions are derived and how the parameters are determined at run-time. The derivation of parameterized mapping functions is our novel contribution.

5.2 Solution Approach

Because, the communication model identification procedure is a part of the Linearization step of the parallelization approaches presented in Chapters 3 and 4, we, first, briefly recall the main steps of these approaches. To illustrate these steps, we will use the running example shown in Figure 1.5(a).

The first steps of the parallelization approaches presented in Chapters 3 and 4 use the dependence analysis in order to extract dependency relations of the initial program. The Fuzzy Array Dependence Analysis (FADA) [37, 38] is used. The FADA is an enhanced version of Exact Array Dependence Analysis (EADA) [4] and it is used to analyze programs with dynamic behavior.

For example, consider the dynamic program depicted in Figure 1.5(a). There are two statements $S1$ and $S2$ writing to array `y[]` and one statement $S3$ which reads from it. Statement $S2$ is guarded by an `if`-condition whose values are determined at run-time. The FADA analysis of this program is described in Section 2.5. The result of the analysis is given below.

$$\sigma(\langle S3,(i_3,j_3)\rangle,(\alpha_i,\alpha_j)) = \left| \begin{array}{ll} \textbf{if} & i_3 \geq \alpha_i \wedge j_3 = \alpha_j \\ \textbf{then} & \langle S2,(\alpha_i,\alpha_j)\rangle \\ \textbf{else} & \langle S1,(j_3)\rangle \end{array} \right. \tag{5.1}$$

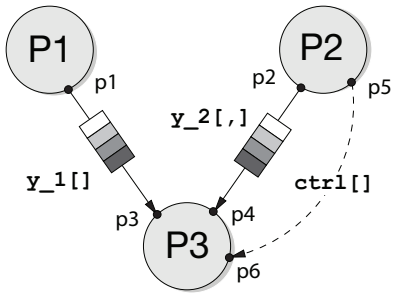
From Equation (5.1) it can bee seen that for any reading operation $\langle S3,(i_3,j_3)\rangle$ the source of the data can be from two different locations. The source is in $S1$ when for given j_3 none of the previous evaluations of the condition at line `C` in Figure 1.5(a) was `true`. Otherwise, the source is in $S2$. Parameters α_i and α_j are introduced by the FADA algorithm. Parameters are used to hide unknown, at compile-time, information and will be determined at run-time. Because the values of parameters are unknown at compile-time, the result of the dependence analysis (i.e., the source operation) shown in Equation (5.1) is unknown at compile-time either, and the result is approximated.

Based on the results of the FADA dependence analysis, the initial sequential program is translated into a *dynamic* Single Assignment Code (dSAC) representation of the initial dynamic program. The dSAC was proposed in [21] as an extension of the Single Assignment Code [4]. A dSAC program is input-output equivalent to the corresponding initial dynamic program and it has the property that every data variable or an array element is written *at most once*. This implies that some variables may not be written at all.

```
1  for j = 1 to 4,
2    ctrl[j] = (5,5)
3  end
4  for k = 1 to 4,
S1:  y_1[k] = F1()
6  end
7  for i = 1 to 4,
8    for j = i to 4,
C:     if y_1[j]<=2,
S2:     y_2[i,j] = F2()
11      ctrl[j] = (i,j)
12     end
13     c1 = ctrl[j].i
14     c2 = ctrl[j].j

15     if i >= c1 && j == c2,
16       in_0 = y_2[c1, c2]
17     else
18       in_0 = y_1[j]
19     end
S3:    [] = F3(in_0)
21   end
22 end
```

(a) The dSAC form



(b) PPN specification

Figure 5.1: An example of a dSAC program and corresponding PPN specification derived from the program shown in Figure 1.5(a).

For example, the dSAC form of the program in Figure 1.5(a) derived from Equation 5.1 is depicted in Figure 5.1(a) where the parameters N and M are set to 4. It is in dSAC form because if we consider line 10, we do not know at compile-time at which iteration the elements of array `y_2[]` will be written. The only thing known is that they will be written at most once. The other array `y_1[]` has the same property as an array in SAC form: every element is written exactly once.

Another property of the dSAC form is the presence of parameters that originate from the FADA analysis. For example, in Figure 5.1(a) the program variables are `c1` and `c2` which correspond to parameters α_i and α_j in Equation 5.1. In order to make dSAC to be functionally equivalent to the initial dynamic program, the values of these parameters have to be changed at run-time.

Parameters are changed with the help of control variables that store the correct value of the parameters for every iteration. For example, dynamic change of the values of `c1` and `c2` is accomplished by lines 13 and 14. The control array `ctrl[]` at line 11 stores the iterations for which the data-dependent condition at line *C* is true. The control variables must be initialized with values that are greater than the maximum value of the corresponding parameters. Therefore, the values of control array `ctrl[]` are initialized with value `(5,5)` at lines 1–3 in Figure 5.1(a). Writing to the control variables is performed just after the functional statement $F2()$, see line 11 in Figure 5.1(a). This guarantees that when the function is executed, the current iteration is stored in the control variables. The values of the control variables are propagated and assigned to the program variables `c1` and `c2` at lines 13 and 14. These variables are used to evaluate the conditions at lines 15 and 16 which determine the source of the data for function $F3()$.

From the dSAC we can build the topology of the PPN depicted in Figure 5.1(b). Every functional statement becomes a process, and every variable or array becomes a channel. For example, lines 4-6 form process P1; lines 1-3 and 7-12 form process P2; and, finally, lines 7-8, 13-21 form process P3. Processes P1 and P3 are connected with a FIFO channel via ports `p1` and `p3`. Processes P2 and P3 are connected with two FIFO channels: the first one (ports `p2` and `p4`) originated from array `y_2[,]` for transferring data, and the second channel which originates from control variable `ctrl[]`, to communicate the outcome of the condition at line C.

5.2.1 Parameterized mapping functions

From the dSAC form and Equation 5.1 we derive parameterized mapping functions which are functions of the Consumer iteration point and vectors of parameters: $f(\vec{y}, \vec{\alpha})$. Vector of parameters $\vec{\alpha}$ is used to uniformly specify a set of unique mapping functions which exist for every instance of a dynamic program, thus capturing the unknown information at compile-time. Values of vector $\vec{\alpha}$ will be determined at run-time during the execution of a PPN.

We define a parameterized mapping function as follows:

Definition 5.2.1 (parameterized mapping function)
A parameterized mapping function is an affine mapping $f_{pq} : I_p \rightarrow O_q : O_q = f(I_p, \vec{\alpha})$, where $I_p \in IPD_p(\vec{\alpha})$ and $O_q \in OPD_q$.

For example, for the P1/P3 pair (ports `p1-p3`) in the PPN shown in Figure 5.1(b), the parameterized mapping function and its domain are:

$$f_{p3p1} : \mathbb{Z}^4 \rightarrow \mathbb{Z} : k = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} (i_3, j_3, \alpha_i, \alpha_j)^{\mathbf{t}}, \tag{5.2}$$
$$\mathbf{D}(f_{p3p1}) = \{(i_3, j_3, \alpha_i, \alpha_j) \in \mathbb{Z}^4 \mid 1 \leq i_3 < \alpha_i \leq 4 \vee i_3 \leq j_3 \neq \alpha_j \leq 4\}.$$

For the P2/P3 pair (ports `p2-p4`), the parameterized mapping function and its domains are:

$$f_{p4p2} : \mathbb{Z}^4 \rightarrow \mathbb{Z}^2 : (i_2, j_2)^{\mathbf{t}} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} (i_3, j_3, \alpha_i, \alpha_j)^{\mathbf{t}}, \tag{5.3}$$
$$\mathbf{D}(f_{p4p2}) = \{(i_3, j_3, \alpha_i, \alpha_j) \in \mathbb{Z}^4 \mid 1 \leq i_3 \geq \alpha_i \leq 4 \wedge i_3 \leq j_3 = \alpha_j \leq 4\}.$$

Finally, we use the parameterized mapping functions to formulate the *Reordering Problem* (RP) and the *Multiplicity Problem* (MP) used to identify the communication models in a PPN derived from a dynamic program. This problems are shown in Figure 5.2 and correspond to Definitions 2.6.1 and 2.6.3, respectively.

The meanings of all constraints in Figure 5.2 are the same as in Figure 2.3. A major difference is that *parameterized* mapping functions are used. The novelty of our communication model identification procedure is to model unknown information at compile-time by the parameterized mapping functions.

$$\begin{cases} y^1, y^2 \in \mathrm{LmP}(\mathbf{D}(f)), \\ y^1 \ll y^2, \\ f(y^1, \alpha^1) \gg f(y^2, \alpha^2). \end{cases}$$

(a) Reordering Problem (RP)

$$\begin{cases} y^1, y^2 \in \mathbf{D}(f) & (c1) \\ y^1 \neq y^2, & (c2) \\ f(y^1, \alpha^1) = f(y^2, \alpha^2). & (c3) \end{cases}$$

(b) Multiplicity Problem (MP)

Figure 5.2: Reordering and Multiplicity Problems for used in communication model identification in PPNs derived from dynamic programs.

The definition of the LmP set used in Figure 5.2(a) is given in Figure 5.3(a). It is a parametric integer linear problem (PILP) similar to the problem given in Definition 2.6.2. The differences between the two formulations of LmP problems are that in the problem shown in Figure 5.3(a) the mapping function is parameterized and this problem finds lexicographically minimal Consumer's iteration points *and* parameter vector $\vec{\alpha}$. For example, consider the P2/P3 pair (and its corresponding ports `p2` and `p4`) formed by statements S2 and S3 from the dSAC shown in Figure 5.1(a).

The LmP problem for the P2/P3 pair is illustrated in Figure 5.3(b). The solution of this problem is $\text{LmP}(\mathbf{D}(f_{p4p2})) = \{(i_3, j_3, \alpha_i, \alpha_j) \in \mathbb{Z}^4 \mid i_3 = i_2 \wedge j_3 = j_2 \wedge \alpha_i = i_2 \wedge \alpha_j = j_2 \wedge 1 \leq i_2 \leq j_2 \leq 4\}$.

objective :

$$(\vec{y}_m, \vec{\alpha}_m) = lexmin\ \{f^{-1}(\vec{x})\},$$

subject to :

$$\begin{cases} \vec{y} \in \mathbf{D}(f), \\ \vec{x} = f(\vec{y}, \vec{\alpha}). \end{cases}$$

(a) Formulation of the LmP problem

$$\begin{cases} i_3 \geq \alpha_i, j_3 = \alpha_j, \\ i_2 = \alpha_i, j_2 = \alpha_j, \\ 1 \leq i_3 \leq j_3 \leq 4, \\ 1 \leq i_2 \leq j_2 \leq 4. \end{cases}$$

(b) An example of LmP problem

Figure 5.3: Formulation of the problem with an example used to find Lexicographically minimal Preimage (LmP) of the Consumer iteration points while deriving a PPN from a dynamic program.

Examples of applying RP and MP problems to our running example depicted in Figure 1.5(a) are shown in Figure 5.4. The RP and MP problems are formulated for the P2/P3 pair formed by statements S2 and S3 from the Figure 5.1(a), respectively. Clearly, the RP problem shown in Figure 5.4(a) does not have an integer solution, because constraints $(c3)$ and $(c4)$ contradict each other. Therefore, the communication model of P2/P3 pair is **in-order**. The MP problem illustrated in Figure 5.4(b) has an integer solution: for some $i_3^1 \neq i_3^2$, there can be $\alpha_i^1 = \alpha_i^2$ which satisfy $(c4)$, and, thus, the communication model of the channel has a **multiplicity**. Therefore, the communication model of P2/P3 pair is *IOM*.

$$\begin{cases} i_3^1 = \alpha_i^1, j_3^1 = \alpha_j^1, \\ i_3^2 = \alpha_i^2, j_3^2 = \alpha_j^2, \\ (i_3^1, j_3^1) \ll (i_3^2, j_3^2), \\ (\alpha_i^1, \alpha_j^1) \gg (\alpha_i^2, \alpha_j^2). \end{cases}$$

(a) RP for P2/P3 pair

$$\begin{cases} i_3^1 \geq \alpha_i^1, j_3^1 = \alpha_j^1, & (c1) \\ i_3^2 \geq \alpha_i^2, j_3^2 = \alpha_j^2, & (c2) \\ (i_3^1, j_3^1) \neq (i_3^2, j_3^2), & (c3) \\ (\alpha_i^1, \alpha_j^1) = (\alpha_i^2, \alpha_j^2). & (c4) \end{cases}$$

(b) MP for P2/P3 pair

Figure 5.4: Examples of RP and MP problems for P2/P3 pair.

5.3 Discussion and Summary

In this chapter, we have presented a novel procedure for communication models identification while deriving PPNs from the dynamic programs defined in Section 2.2. The procedure, needed to convert multidimensional arrays used for data communication after the Data Dependence analysis step, distinguishes between four patterns of communicating data and identifies the most general one which can realize all dependency patterns in different instances of a dynamic program. This identification procedure solves the optimization problem by finding the most optimal realization for each P/C pair.

The novel idea that we have used in our communication model identification procedure is the parameterized mapping functions. We derive them from a dSAC specification of an initial dynamic application using the Fuzzy Array Dataflow Analysis. Parameterized mapping functions are used to describe all possible dependency relations that exist in all instances of a dynamic program for each P/C pair. Parameters in these functions are used to uniformly specify a set of unique mapping functions which exist for every instance of dynamic program, thus, capturing unknown information at compile-time. Using parameterized mapping functions, we reformulated the problems presented in Section 2.6: the *Reordering Problem* (RP) and the *Multiplicity Problem* (MP) which are used to identify communication models in a derived PPN.