

Friedrich-Alexander-Universität
Erlangen-Nürnberg



Controller Synthesis for Mapping Partitioned Programs on Array Architectures

Hritam Dutta, Frank Hannig and Jürgen Teich

{dutta, hannig, teich}@cs.fau.de

URL: <http://www12.informatik.uni-erlangen.de>

Department of Computer Science 12

Hardware-Software-Co-Design

University of Erlangen-Nuremberg

Am Weichselgarten 3

D-91058 Erlangen, Germany

Co-Design-Report 03-2005

June 18, 2005

Contents

1	Introduction and Related Work	4
2	Background	5
2.1	Definitions, Notations, and Transformations	5
2.2	Partitioning	5
2.3	Allocation and Scheduling	7
3	Control Generation	8
3.1	Why is the Problem of Control Generation Intractable?	9
3.2	Control Design Flow: Methodology	10
3.2.1	Determination of PE type	11
3.2.2	Determination of Scanning Code	12
3.2.3	Control Unit Generation	15
3.2.4	Propagation of Global Control Signals and Counter Values	19
4	A case study	20
5	Conclusions and Future Work	21
	References	22

Abstract

Processor arrays can be used as accelerators for a plenty of dataflow-dominant applications. In-nately these applications have almost no control flow, but the application of sophisticated partitioning and scheduling techniques in order to handle large scale problems and to balance local memory requirements with I/O-bandwidth has the disadvantage of a more complex control flow. Thus, efficient control path synthesis is one of the greatest challenges when compiling algorithms onto processor arrays. This paper presents an efficient methodology for the automated control path synthesis for the mapping of partitioned algorithms onto processor arrays. The major advantages observed in the presented methodology are seen in, (a) control generation for different partitioning techniques and arbitrary parallelepiped tiles, (b) combined use of a global and a local control strategy in order to reduce the control overhead, (c) up to 90 percent reduction in control path area and resources compared to existing approaches.

1 Introduction and Related Work

In the last decade, there has been a dramatic growth in research and development of massively parallel processor arrays both in academia and industry. The trends in lithography and process integration technology allow the implementation of hundreds of 32-bit microprocessors on a single die. Furthermore, the expensive design process for ASICs calls for an automated synthesis of such accelerators in form of array architectures. Also, the introduction of reconfigurable architectures allows to exploit the flexibility of software along with the performance of processor arrays. Processor array architectures provide an optimal platform for the parallel execution of number crunching loop programs from fields of digital signal processing, image processing, linear algebra, etc. However, due to a lack of mapping tools, these massively parallel processor architectures are not able to realize their full potential. The ultimate aim of such mapping tools is to map software loops, such as *for* or *while* loops in C programs, onto a hardware target subject to the performance constraints in latency, area, or power. The example of state of the art mapping tools are PICO-Express [1], and MMalpha [2].

The polytope model [3] is an intuitive methodology for loop parallelization and mapping of loop nests onto massively parallel architectures. The architectures in form of processor array may be implemented on FPGAs or coarse-grained programmable array architectures. The *space-time mapping* is an important transformation for obtaining full-size processor array descriptions from a given nested loop program. *Partitioning* is another necessary transformation for mapping loops onto reduced-size arrays in order to meet the resource constraints. Well known partitioning techniques are *Tiling* and *Clustering*. *Control generation* is a transformation which is responsible for the control path synthesis, which produces control signals to orchestrate the correct execution of the loop program. The systematic design of control units of full-size processor arrays was first introduced in [4]. Another procedure for the systematic definition of control signals for the class of conditional uniform recurrence equations (CUREs) was introduced in [5]. The first method is characterized by local control flow, problem size independence, and optimization of the number of required control variables. However, both the methodologies are restricted to simple space-time mappings obtained by a projection. Darté et al. [6] introduced a method for the automatic generation of control code in case of clustering with rectangular tiles and tight linear schedules. The main contribution of this paper is the introduction of a general methodology for control path generation on loop partitioning with congruent tiles with maximum reuse of control predicates in massively parallel architectures. But first in Section 2, we briefly introduce some definitions and important transformations. Afterwards in Section 3, a concise description of control generation is given, emerging problems are discussed, and our novel control generation methodology is presented. Finally, in Section 4 and 5, a case study of our methodology and conclusions are presented, respectively.

2 Background

2.1 Definitions, Notations, and Transformations

In this paper, the class of algorithms we are dealing with is a class of recurrence equations defined as follows:

Definition 2.1 (PLA). A piecewise linear algorithm consists of a set of N quantified equations, $S_1[I], \dots, S_i[I], \dots, S_N[I]$. Each equation $S_i[I]$ is of the form

$$\forall I \in \mathcal{I}_i : x_i[P_i I + f_i] = \mathcal{F}_i(\dots, x_j[Q_j I - d_{ji}], \dots) \text{ if } \mathcal{C}_i^1(I)$$

P_i, Q_j are constant rational indexing matrices and f_i, d_{ji} are constant rational vectors of corresponding dimension.

The domains \mathcal{I}_i are defined as *Linearly Bounded Lattices* [7]. With these definitions, several combinations of parallelizing transformations like *embedding* of variables, *localization* (vectorization), or *operator splitting* in the polytope model can be applied, for the sake of brevity we refer to [8].

Example 2.1 The matrix multiplication is taken as an example to illustrate our methodology. The product $C = A \cdot B$ of two square matrices $A, B \in \mathbb{Z}^{N \times N}$ is defined as $c_{ij} = \sum_{k=1}^N a_{ik} b_{kj} \quad \forall 1 \leq i \leq N \wedge 1 \leq j \leq N$. Let $N = 8$, then after application of the above mentioned transformations the following PLA (satisfying Definition 2.1) is obtained.

$$\begin{aligned} a[i, j, k] &= a[i, 0, k] \\ b[i, j, k] &= b[0, j, k] \\ z[i, j, k] &= a[i, j, k] \cdot b[i, j, k] \\ c[i, j, k] &= \begin{cases} c[i, j, k-1] + z[i, j, k] & \text{if } k > 0 \\ z[i, j, k] & \text{if } k = 0 \end{cases} \\ C_{out}[i, j, k] &= c[i, j, k] & \text{if } k = 7 \end{aligned}$$

The matrices A and B are embedded into the arrays as follows, $a[i, 0, k] = a_{ik}, b[0, j, k] = b_{kj}$. The index space is given by $\mathcal{I} = \{I = (i \ j \ k)^T \in \mathbb{Z}^3 \mid 0 \leq i, j, k \leq 7\}$.

2.2 Partitioning

Partitioning is a well known transformation which covers the index space of computation using congruent hyperplanes, hyperquaders, or parallelepipeds called *tiles* [9], [10]. For processor arrays (PAs), it is carried out in order to match a loop nest implementation to resource constraints in terms of available number of processing elements (PEs), local memory, and communication bandwidth. Well known partitioning techniques are multiprojection, LSGP (local sequential global parallel, often also referred as clustering or blocking) and LPGS (local parallel global sequential, also referred as tiling). Formally, partitioning divides the index space \mathcal{I} using congruent tiles such that it is decomposed into spaces \mathcal{J} and \mathcal{K} , i.e., $\mathcal{I} \mapsto \mathcal{J} \oplus \mathcal{K}$, where tiles may be defined by

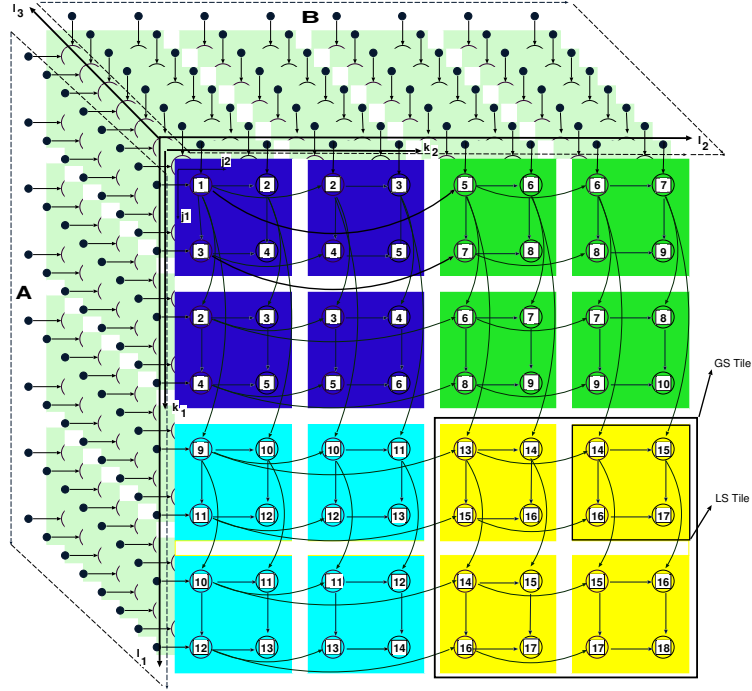


Figure 1: The iteration space of the partial localized co-partitioned matrix multiplication 8×8 example. Each arc denotes a data dependency. The numbers correspond to the time step of execution as specified by the space-time mapping in Eq. (3).

tiling matrix, P^1 . $\mathcal{J} \in \mathbb{Z}^n$ represents the points within the tile and $\mathcal{K} \in \mathbb{Z}^n$ accounts for regular repetition of the tiles, i.e., the origin of each tile. Hierarchical partitioning (often referred to as multi-blocking) methods use different hierarchies of tiling matrices to divide the index space. Co-partitioning is such an example of a 2-level hierarchical partitioning [11], where the index space is first partitioned into LS (local sequential) tiles, this tiled index space is tiled once more using GS (global sequential) tiles as shown in Fig. 1. Co-partitioning uses both LSGP and LPGS methods in order to balance local memory requirements with I/O bandwidth with the advantage of problem size independence. Formally, it is defined as splitting of an index space into spaces \mathcal{J} , \mathcal{K} and \mathcal{L} , i.e., $\mathcal{I} \mapsto \mathcal{J} \oplus \mathcal{K} \oplus \mathcal{L}^2$ using two congruent tiles defined by tiling matrices, P_{LS} and P_{GS} . $\mathcal{J} \in \mathbb{Z}^n$ represents the points within the LS tiles and $\mathcal{K} \in \mathbb{Z}^n$ accounts for the regular repetition of the origin of LS tiles (i.e., tiles marked with same shade in Fig. 1). $\mathcal{L} \in \mathbb{Z}^n$ accounts for the regular repetition of the GS tiles (i.e., bigger tiles marked with different shade in Fig. 1). Similarly an n -hierarchical partitioning method splits the index space \mathcal{I} into $n + 1$ spaces.

Example 2.2 *Co-partitioning and subsequent partial localization [12] of the matrix multiplication example is shown in Fig. 1. The PLA obtained on application of above mentioned transfor-*

¹ $\mathcal{J} \oplus \mathcal{K} = \{i = j + P \cdot k \mid j \in \mathcal{J} \wedge k \in \mathcal{K} \wedge P \in \mathbb{Z}^{n \times n}\}$

² $\mathcal{J} \oplus \mathcal{K} \oplus \mathcal{L} = \{i = j + P_{LS} \cdot k + P_{GS} \cdot l \mid j \in \mathcal{J} \wedge k \in \mathcal{K} \wedge l \in \mathcal{L} \wedge P_{LS}, P_{GS} \in \mathbb{Z}^{n \times n}\}$

mations is given in Eq. (1) and is shown in Fig. 1. The loop matrices used for tiling are

$$P_{GS} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad P_{LS} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

One can verify that the index point $I = (5, 7, 0)$ is uniquely mapped to $J = (1, 1)$, $K = (0, 1)$ and $L = (1, 1, 0)$ after co-partitioning³.

$$a[j_1, j_2, k_1, k_2, l_1, l_2, l_3] = \begin{cases} a[j_1, j_2 - 1, k_1, k_2, l_1, l_2, l_3] & \text{if } j_2 > 0 \\ a[j_1, j_1, k_1, k_2 - 1, l_1, l_2, l_3] & \text{if } j_2 + k_2 > 0 \wedge j_2 = 0 \\ a[j_1, j_1, k_1, k_2, l_1, l_2 - 1, l_3] & \text{if } j_2 + k_2 + l_2 > 0 \wedge \\ & j_2 + k_2 = 0 \wedge j_2 = 0 \\ A_{j_1+2 \cdot k_1+4 \cdot l_1, l_3} & \text{if } j_2 + k_2 + l_2 = 0 \wedge \\ & j_2 + k_2 = 0 \wedge j_2 = 0 \end{cases}$$

$$b[j_1, j_2, k_1, k_2, l_1, l_2, l_3] = \begin{cases} b[j_1 - 1, j_2, k_1, k_2, l_1, l_2, l_3] & \text{if } j_1 > 0 \\ b[j_1, j_2, k_1 - 1, k_2, l_1, l_2, l_3] & \text{if } j_1 + k_1 > 0 \wedge j_1 = 0 \\ b[j_1, j_2, k_1, k_2, l_1 - 1, l_2, l_3] & \text{if } j_1 + k_1 + l_1 > 0 \wedge \\ & j_1 + k_1 = 0 \wedge j_1 = 0 \\ B_{l_3, j_2+2 \cdot k_2+4 \cdot l_2} & \text{if } j_1 + k_1 + l_1 = 0 \wedge \\ & j_1 + k_1 = 0 \wedge j_1 = 0 \end{cases} \quad (1)$$

$$z[j_1, j_2, k_1, k_2, l_1, l_2, l_3] = a[j_1, j_2, k_1, k_2, l_1, l_2, l_3] \cdot b[j_1, j_2, k_1, k_2, l_1, l_2, l_3]$$

$$c[j_1, j_2, k_1, k_2, l_1, l_2, l_3] = \begin{cases} c[j_1, j_2, k_1, k_2, l_1, l_2, l_3 - 1] & \text{if } l_3 > 0 \\ + z[j_1, j_2, k_1, k_2, l_1, l_2, l_3] \\ z[j_1, j_2, k_1, k_2, l_1, l_2, l_3] & \text{if } l_3 = 0 \end{cases}$$

$$C[j_1, j_2, k_1, k_2, l_1, l_2, l_3] = c[j_1, j_2, k_1, k_2, l_1, l_2, l_3] \quad \text{if } l_3 = 7$$

for all $J = (j_1 \ j_2)^T \in \mathcal{J}$, $K = (k_1 \ k_2)^T \in \mathcal{K}$, and $L = (l_1 \ l_2 \ l_3)^T \in \mathcal{L}$, with

$\mathcal{J} = \{J \in \mathbb{Z}^2 \mid 0 \leq j_1, j_2 \leq 1\}$, $\mathcal{K} = \{K \in \mathbb{Z}^2 \mid 0 \leq k_1, k_2 \leq 1\}$, and

$\mathcal{L} = \{L \in \mathbb{Z}^3 \mid 0 \leq l_1, l_2 \leq 1 \wedge 0 \leq l_3 \leq 7\}$

2.3 Allocation and Scheduling

Linear transformations are used as *space-time mappings* in order to assign a processor p (space) and a sequencing index t (time) to index vectors [8]. In co-partitioning, the index points within the LS tiles are executed sequentially. All LS tiles within a GS tile are executed in parallel by the processor array. Therefore, the number of processors in the array is equal to the number of LS tiles within a GS tile. The GS tiles are executed sequentially.

³ j_3, k_3 are removed from the description as $j_3 = 0$, and $k_3 = 0$

$$\begin{aligned}
a[p_1, p_2, t] &= \begin{cases} a[p_1, p_2, t-1] & \text{if } j_2 > 0 \\ a[p_1, p_2-1, t-1] & \text{if } j_2 + p_2 > 0 \wedge j_2 = 0 \\ a[p_1, p_2, t-4] & \text{if } j_2 + p_2 + l_2 > 0 \wedge j_2 + p_2 = 0 \wedge j_2 = 0 \\ A_{2p_1+j_1+4l_1, l_3} & \text{if } j_2 + p_2 + l_2 = 0 \wedge j_2 + p_2 = 0 \wedge j_2 = 0 \end{cases} \\
b[p_1, p_2, t] &= \begin{cases} b[p_1, p_2, t-2] & \text{if } j_1 > 0 \\ b[p_1-1, p_2, t-1] & \text{if } j_1 + p_1 > 0 \wedge j_1 = 0 \\ b[p_1, p_2, t-8] & \text{if } j_1 + p_1 + l_1 > 0 \wedge j_1 + p_1 = 0 \wedge j_1 = 0 \\ B_{l_3, 2p_2+j_2+4l_2} & \text{if } j_1 + p_1 + l_1 = 0 \wedge j_1 + p_1 = 0 \wedge j_1 = 0 \end{cases} \\
z[p_1, p_2, t] &= a[p_1, p_2, t] \cdot b[p_1, p_2, t] \\
c[p_1, p_2, t] &= \begin{cases} c[p_1, p_2, t-16] + z[p_1, p_2, t] & \text{if } l_3 > 0 \\ z[p_1, p_2, t] & \text{if } l_3 = 0 \end{cases} \\
C[p_1, p_2, t] &= c[p_1, p_2, t] \quad \text{if } l_3 = 7
\end{aligned} \tag{3}$$

Definition 2.2 (*Space-time mapping for co-partitioning*). A space-time mapping in case of co-partitioning is an affine transformation of the form

$$\begin{pmatrix} p \\ t \end{pmatrix} = \begin{pmatrix} 0 & E & 0 \\ \lambda_J & \lambda_K & \lambda_L \end{pmatrix} \begin{pmatrix} J \\ K \\ L \end{pmatrix} \tag{2}$$

where $E \in \mathbb{Z}^{n_K \times n_K}$ is the identity matrix, $\lambda_J \in \mathbb{Z}^{1 \times n_J}$, $\lambda_K \in \mathbb{Z}^{1 \times n_K}$, $\lambda_L \in \mathbb{Z}^{1 \times n_L}$.

Other hierarchical partitioning schemes can be realized using an appropriate selection of an affine transformation characterizing scheduling and allocation of the index points. The problem of determining an optimal sequencing index (i.e., $\lambda_J, \lambda_K, \dots$) might be solved by Mixed Integer Linear Programming similar as in [13].

Example 2.3 An optimal space-time mapping for Ex. 2.2 on co-partitioning according to Def. 2.2 is

$$\begin{pmatrix} p_1 \\ p_2 \\ t \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 8 & 4 & 16 \end{pmatrix} \begin{pmatrix} J \\ K \\ L \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \text{ where } J = \begin{pmatrix} j_1 \\ j_2 \end{pmatrix}, K = \begin{pmatrix} k_1 \\ k_2 \end{pmatrix}, L = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix}$$

Therefore, a 2×2 processor array is obtained which executes the LS tiles in parallel and GS tiles sequentially. The PLA obtained after space-time mapping is shown in Eq. (3).

3 Control Generation

The control path generation is an intrinsic part for the automated generation of PAs. The data path synthesis can be easily obtained from the assignment statements in a PLA (see Eq. (3)) as

they are only processor p and time t dependent. The control path synthesis is more complicated as the control predicates depend on variables other than p and t . The iteration dependent if-conditionals occurring in a given PLA (see Eq. (1) and Eq. (3)) have to be replaced by control variables for efficient parallelization. Furthermore, scheduling and allocation of operations on a resource constrained architecture as determined by a partitioning method requires the generation of control signals. The input and/or output data to be read/write from/to border processors need control signals. Therefore, a step for control generation is needed that specifies the control units and the control signals of the processor array.

3.1 Why is the Problem of Control Generation Intractable?

Let a partitioned PLA and a space-time mapping as described in Section 2.3 be given, then the main challenge of control generation is to derive the control signals which compute the conditionals. The intractability of the control generation problem is illustrated in this section with help of co-partitioning. All the iteration dependent conditionals after co-partitioning can be represented in one of the following forms⁴:

$$A_J \cdot J \geq b_J \quad \wedge \quad A_K \cdot p \geq b_K \quad \wedge \quad A_L \cdot L \geq b_L \quad (4)$$

$$A_J \cdot J + A_K \cdot p + A_L \cdot L \geq b \quad (5)$$

e.g., $j_1 \geq 0$, $j_1 \geq 1 \wedge p_1 \geq 1$, $j_1 + p_1 \geq 1$, and other conditionals as seen in the right hand side in Eq. (3) all can be represented in one of the above two forms. The original iteration space co-ordinates, I are obtained as $J \oplus K \oplus L$. Therefore, the calculation of memory addresses and control predicates which are affine functions of I or other index variables can be done only if the following values are available.

- LS tile co-ordinates, J : For each given time step t and processor p , the LS co-ordinates J of the index points being executed have to be known.
- Processor co-ordinates, K or p : Each processor knows its co-ordinates as defined by the space-time mapping.
- GS tile co-ordinates, L : For each time step t and processor p , the GS co-ordinate L of the index point being executed is required.

The rigidity of the problem stems from the fact that given $(p \ t)^T$, the space $(J \ p \ L)^T$ needs to be calculated for the predicate computation from the following linear Diophantine equation

$$t - \lambda_K \cdot p = \lambda_J \cdot J + \lambda_L \cdot L \quad (6)$$

The equation has an unique solution for each relevant t and p . The following example illustrates the problem.

⁴Note, $p = K$ directly follows from the definition of the space-time mapping for co-partitioning, cp. Definition 2.2.

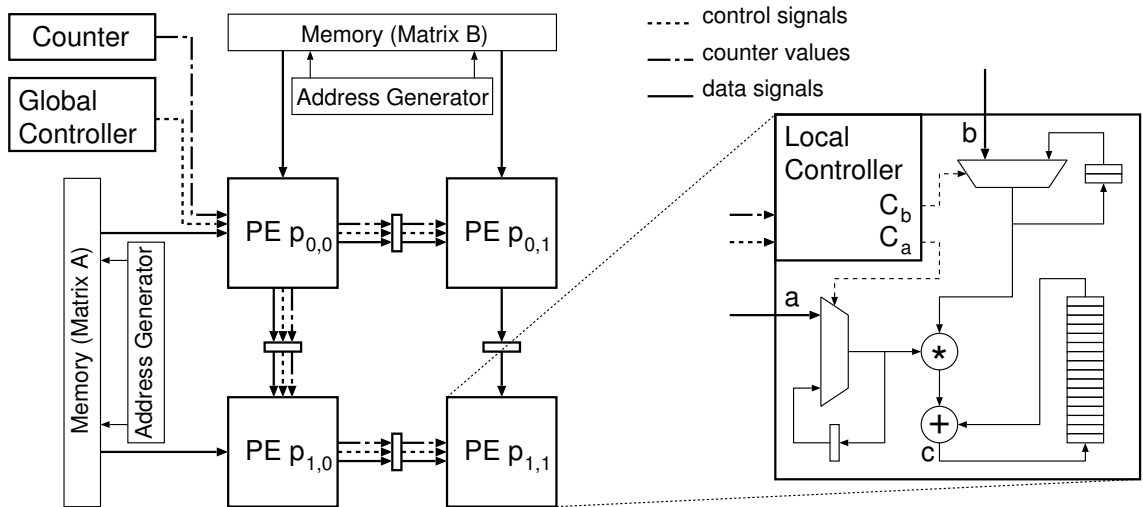


Figure 2: An example 2×2 processor array implementation of a co-partitioned matrix multiplication.

Example 3.1 *The index space in Fig. 3 (a) shows the execution ordering for an iteration space (tile) J . The execution ordering is obtained due to the schedule vector, i.e., $\lambda_J = (3 \ 4)$ and $t = \lambda_J \cdot J$. Suppose the conditional “if ($j_1 + j_2 = 2$)” needs to be executed. Then, given $t = 3j_1 + 4j_2$, for every time step t the equation has to be solved for j_1, j_2 . Afterwards, the “if” conditional can be computed using the calculated j_1, j_2 in the required predicate. Let, c_1 be the control variable corresponding to our given conditional. Then c_1 needs to be true at time step 8, 9, and 10.*

One possible direct approach for finding a solution of Eq. (6) is the usage of the *Smith Normal Form* [14]. However, the direct approach has disadvantages. First, for each processor p , a table of pre-calculated iteration variables (e.g., J, L) or control signals have to be maintained if solved at compile-time. This is infeasible in terms of area costs as the values have to be stored over all time steps of execution in the table. The other way of implementing a hardware solution of Eq. (6) in run-time instead of pre-calculation has very high hardware costs. In the next section we present a novel methodology for finding the requisite iteration variables and henceforth the correct control path synthesis.

3.2 Control Design Flow: Methodology

The methodology for control generation introduced in this section is not based on a direct solution approach, but based on the scanning of the index space in the order as given by the space-time mapping. The methodology for control generation proposed in this section encompasses all possible partitioning techniques (i.e., LPGS, LSGP, co-partitioning and other hierarchical partitioning methods) using congruent parallelepiped tiles. The only major assumption is the application of linear affine scheduling. This is not a disadvantage as it had been shown that linear scheduling under certain restrictions is nearly optimal [15]. Fig. 2 shows a 2×2 processor array realization of

the co-partitioned matrix multiplication example, where the iterative conditionals which depend on the processor index are implemented by a *local controller*. The conditionals (usually common over a group of processors) and counter (generating index variables) independent of the processor index are implemented in a *global controller* and *counter*, respectively. This leads to a considerable reduction in area cost of the control path. To determine these described components, our approach for control path generation is constituted of the following four steps.

Determination of PE types. This step finds processor regions of same type. This helps in classification of control predicates for local and global controllers.

Scanning of the partitioned polytope. In this step, a global counter (see Fig. 2) for producing values of the index space variables is determined.

Initialization of local and global control signals. In this step, local and global controllers (see Fig. 2) for the execution of control predicates are generated.

Propagation of control and iteration variables. The requisite delays and directions (see Fig. 2) required for the propagation of global counter and global control signals in the processor array are determined.

3.2.1 Determination of PE type

The main aim of the determination of PE types is

- To separate as many predicates as possible which can be executed by a global controller from those which must be locally computed. Without the determination of different PE types, the methodology would implement the local control model, i.e., all predicates would be computed in a local controller inside each PE.
- Classification of processor regions executing the same functions. The advantage appears in customizing local control units and leading to a customized hardware synthesis of data paths of PEs.

The following strategy is used for the separation of predicates to be implemented in a local or a global controller. Processor p based iteration dependent conditionals of the form as in Eq. (5) have to be implemented by a local controller in each PE, if $A_K \neq 0$. Control conditions of types as in Eq. (4) and in Eq. (5) if $A_K = 0$ may be implemented in a global controller. However, processor regions associated with global control signals have to be identified. This is done by an orthogonal projection of the set of inequalities defining the corresponding control space onto the subspace defined by processor variables, p ⁵.

Let Q be the number of statements having conditionals of type Eq. (4) or Eq. (5) (if $A_K = 0$). The statement S_q is associated with polyhedron \mathcal{P}_q as defined by $A_q \cdot p \geq b_q$ for $q = 1, \dots, Q$.

$${}^5(A_s \cdot p \geq b_s = \text{Proj}_p \left(\begin{pmatrix} A_J & A_K & A_L \\ A_J & 0 & 0 \\ 0 & A_K & 0 \\ 0 & 0 & A_L \end{pmatrix} \begin{pmatrix} J \\ p \\ L \end{pmatrix} \geq \begin{pmatrix} b \\ b_J \\ b_K \\ b_L \end{pmatrix} \right))$$

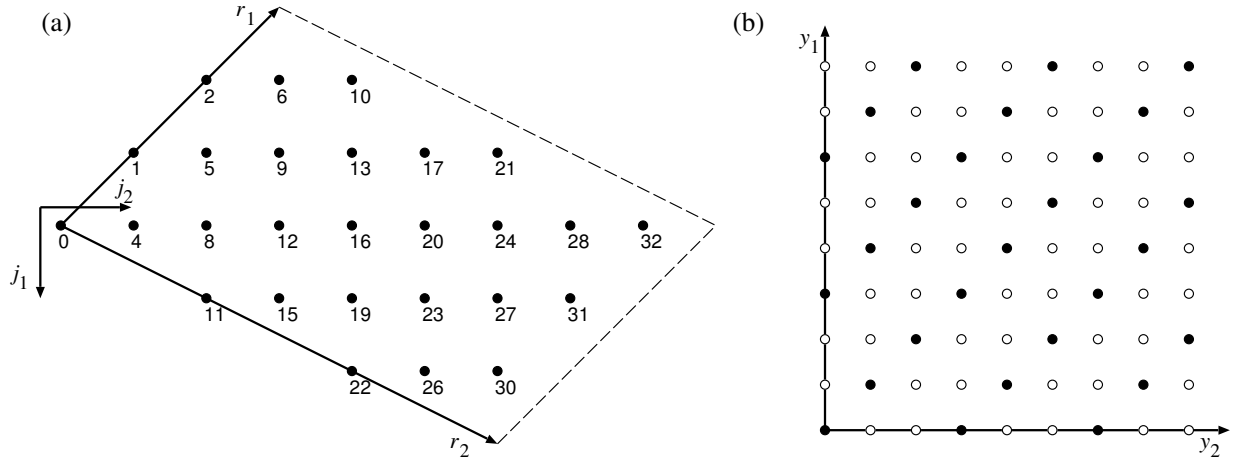


Figure 3: (a) Execution of index points within a tile. (b) Transformed domain.

The problem of identifying processor regions is now reduced to finding a non-intersecting set of k polyhedra, whose union covers the set $\bigcup_{q=1}^Q \mathcal{P}_q$. This is similar to the problem of code generation for multiple statements each defined over a different polyhedron. An efficient algorithm for the identification of k such processor spaces can be found in [16]. The Q statements are then reallocated to the k processor regions. Therefore, we obtain a PLA description (as in Eq. (3)) for each of the k PE types. The PLA in Eq. (3) for matrix multiplication has four PE region types characterized by PE1($p_1 = 0 \wedge p_2 = 0$), PE2($p_1 > 0 \wedge p_2 = 0$), PE3($p_1 = 0 \wedge p_2 > 0$), and PE4($p_1 > 0 \wedge p_2 > 0$), respectively.

3.2.2 Determination of Scanning Code

The purpose of this section is to synthesize a global counter which produces values of the required index variables (e.g., J , L for co-partitioning, J for LSGP, or K for LPGS) as specified by the schedule vector. The schedule vector is defined by a *loop matrix* [7].

Definition 3.1 A loop matrix $R = (r_1 \ r_2 \ \dots \ r_s) \in \mathbb{Z}^{s \times s}$ determines the ordering of index points J within a tile at time t . Index points in direction of r_1 are mapped side by side onto t , index points in direction of r_2 are separated by blocks of points in direction r_1 and so on. The ordering is similar to a sequential nested loop program where the loop index i_k corresponds to iterations in direction of r_k . The inner loop index is i_1 , and the outermost loop index is i_s .

To find a suitable schedule vector for co-partitioning, two loop matrices are required for J and L , respectively. However, for the sake of brevity, the following example illustrates our methodology for the counter generation of a LSGP tile using a single loop matrix.

Example 3.2 Fig. 3 (a) shows for a tile the relationship between the chosen loop matrix $R = \begin{pmatrix} -3 & 3 \\ 3 & 6 \end{pmatrix}$ and the derived schedule vector λ_J , $t = \lambda_J J = (3 \ 4) \begin{pmatrix} j_1 \\ j_2 \end{pmatrix}$. The lexicographic scanning in j_1 , j_2 can-

$$\begin{pmatrix} E & -T \\ -E & T \\ 0 & A \end{pmatrix} \begin{pmatrix} Y \\ J \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ b \end{pmatrix} \Leftrightarrow \begin{pmatrix} 1 & 0 & 2 & -1 \\ 0 & 1 & -1 & -1 \\ -1 & 0 & -2 & 1 \\ 0 & -1 & 1 & 1 \\ 0 & 0 & -6 & 3 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 6 & -3 \\ 0 & 0 & -3 & -3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ j_1 \\ j_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -26 \\ -26 \end{pmatrix} \quad (7)$$

not lead to an execution order as defined by the loop matrix R . Therefore, a transformation to the orthogonal domain defined by y_1, y_2 as depicted in Fig. 3 (b) has to be defined. Let the vectors $Y = (y_1 \ y_2)^T$ and $J = (j_1 \ j_2)^T$ represent the transformed orthogonal domain and the initial domain, respectively. The transformation is defined by $E \cdot Y = T \cdot J$, where E is the identity matrix and T is an appropriate transformation matrix (in this example $T = \begin{pmatrix} -2 & 1 \\ 1 & 1 \end{pmatrix}$). Subsequently, a lexicographic scanning in the transformed domain is implemented. In Fig. 3 (b), the transformed domain is shown where a lexicographic scanning of black points (images of index points in original domain) needs to be carried out. Therefore, a counter has to be obtained in terms of a FOR loop which counts in the transformed domain the image points (e.g., the black points in Fig. 3 (b)) and skips the holes (i.e., the white points in Fig. 3 (b)). The strides of the loop variables y_1, y_2 can be determined by finding the diagonal elements of the Hermite Normal Form (HNF) [14] of the transformation matrix T (here, $S = \text{HNF} \begin{pmatrix} 1 & 0 \\ -2 & 3 \end{pmatrix}$). The outer counter variables depend on the loop matrix which in turn determines the scheduling vector. The lower bounds (i.e., initialization values) of the loop variables is the lexicographic minimum of the loop variables in the following system of inequalities under the context of the outer loop variables [17], The set of inequalities $AJ \geq b$ defines the given initial tile whose points are to be scanned. Using Parametric Integer Programming (PIP) [18], the system of inequalities as in Eq. (7) can be used to find the lexicographic minimum of the inner loop variables (i.e., in this example y_1) under the context of the outermost loop as obtained from the corresponding row in $AT^{-1} \cdot y \geq b$ (i.e., $0 \leq 3y_2 \leq 26$). The variable lower in the following pseudocode⁶ is the lexicographic minimum of y_1 . Finally, the inverse of the transformation matrix is used to obtain the index variables in the original domain.

```

FOR ( $y_2 = 0; y_2 \leq 8; y_2 = y_2 + 1$ )
  lower =  $y_2 - 3(((2y_2) \div 3) \div 2)$ 
  FOR ( $y_1 = \text{lower}; y_1 \leq 8; y_1 = y_1 + 3$ )
     $j_1 = (-y_1 + y_2) \div 3;$ 
     $j_2 = (y_1 + 2y_2) \div 3;$ 

```

⁶The symbol “ \div ” denotes a modulo division.

Table 1: Counter output for Example 3.1. En, Res, "x" are the enable signal, reset signal, and stall states, respectively.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
y_1	0	3	6	x	1	4	7	x	2	5	8	0	3	6	x	1	4	7	x	2	5	8	0	3	6	x	1	4	7	x	2	5	8
y_2	0	0	0	x	1	1	1	x	2	2	2	3	3	3	x	4	4	4	x	5	5	5	6	6	6	x	7	7	7	x	8	8	8
j_1	0	-1	-2	x	0	-1	-2	x	0	-1	-2	1	0	-1	x	1	0	-1	x	1	0	-1	2	1	0	x	2	1	0	x	2	1	0
j_2	0	1	2	x	1	2	3	x	2	3	4	2	3	4	x	3	4	5	x	4	5	6	4	5	6	x	5	6	7	x	6	7	8
s	0	1	2	4	4	5	6	8	8	9	10	11	12	13	15	15	16	17	19	19	20	21	22	23	24	26	26	27	28	30	30	31	32
En	1	1	1	0	1	1	1	0	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	0	1	1	1	0	1	1	1
Res	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

ENDFOR

ENDFOR

The pseudocode is implemented as a ScanCounter which produces the index variables as specified by the scheduling order.

The questions that remain to be answered are: How can the above scanning code be synchronized with the time as determined by the schedule vector? Also, how can the transformation matrix, T be determined? The scanning code needs to be synchronized with the time as determined by the schedule vector. In this example, this corresponds to generating no values (stall states) at time steps $t = 3, 7, 14, 18, 25, 29$ as shown in Fig. 3 (a) and Table 1. This is accomplished by providing an enable signal to the ScanCounter which stops it at the requisite time steps. The enable mechanism's concept is shown in Fig. 4. Note that the enable mechanism is not required if there are no stall states (cycles). Furthermore, for power efficiency reasons the enable signal is used to switch off the PEs. The index points as determined by the loop matrix are output of the ScanCounter. The second counter counts over the time, which runs, without loss of generality, from 0 to t_{tile} , the last execution time of an index point within the tile.

If at time t , $s = 3j_1 + 4j_2 \neq t$, then the ScanCounter is halted by turning off the enable signal. If an enable mechanism is required or not can be determined by considering if the total time taken to compute a tile is a multiple of the number of points within the tile. The transformation matrix is given by

$$T = \frac{\sigma \cdot \text{adj}(R)}{\text{gcd}(r_{i,j})} \quad (8)$$

where $\sigma = |\det(R)|/\det(R)$ and $\text{gcd}(r_{i,j})$ is the greatest common divisor of all elements of the loop matrix. The above explanation has been carried out with reference to LSGP partitioning. The methodology can be extended to LPGS, co-partitioning and other hierarchical partitioning methods by passing the relevant loop matrices and schedule vectors in Algorithm 1. For instance, for LSGP, λ_K and the corresponding loop matrix, for co-partitioning λ_J , λ_L and two corresponding loop matrices. Therefore, for co-partitioning we obtain two counters producing J and L independently. Let a loop matrix and a derived schedule vector be given. Then, a corresponding counter can be generated by the following algorithm.

Algorithm 1: Counter generation

INPUT: Loop matrix R , Schedule vector λ, A, b .

OUTPUT: ScanCounter (as a FOR loop for J), Enable logic.

- Step 1: Determination of the transformation matrix T : If $\text{adj}(R) \neq R$, the transformation matrix is determined using Eq. (8), else $T = E$ where E is the identity matrix.
 - Step 2: Generation of scanning code:
 - Step 2.1: Determine the strides from the HNF of the transformation matrix T , i.e., $S = \text{HNF}(T)$.
 - Step 2.2: Determine upper and lower bounds of variables in the transformed domain defined by Y . The lower bounds are found as lexicographic minimum of the system of inequalities (7) under the context of the outermost loop. The upper bounds are found from $(AT^{-1})Y \geq b$.
 - Step 2.3: Determine values of counter variables in original domain \mathcal{J} by inverse transformation, i.e., $J = T^{-1}Y$.
 - Step 2.4: Write down the counter description in terms of a FOR loop.
 - Step 3: If the total time taken to compute a tile is not a multiple of the number of index points in the tile then generate an enable mechanism to synchronize the scanning code:
 - Step 3.1: Calculate $t_{tile} = \max_{J \in \mathcal{J}} \{\lambda_J J\}$, time needed to execute the tile.
 - Step 3.2: Generate the TimeCounter as in Fig. 4 with 0 as lower bound and t_{tile} as upper bound. The TimeCounter is incremented every clock cycle.
 - Step 3.3: The conditional unit as in Fig. 4 is configured, producing enable as true if and only if the time ($S = \lambda J$) corresponding to the ScanCounter matches the time t as specified by TimeCounter. The reset signal is produced when the TimeCounter reaches the upper bound t_{tile} .
 - Step 4: Update the ScanCounter every δ cycles, where δ is the iteration interval.
-

3.2.3 Control Unit Generation

This section deals with the automated control unit generation given the requisite predicates. After PE classification, each PE of different type has an individual behavioral specification. The behavioral specification of an example PE of q^{th} PE type ($p \in P_q$ (say)) can be the PLA in Eq. (9).

$$\begin{aligned}
 x_1[I] &= \begin{cases} \mathcal{F}_1^1(\dots, x_j[I - d_{1,1}] \dots) & \text{if } \tilde{I} \in \tilde{\mathcal{I}}_1^1 \text{ (L)} \wedge \hat{I} \in \hat{\mathcal{I}}_1^1 \text{ (G)} \\ \vdots & \vdots \\ \mathcal{F}_1^{W_1}(\dots, x_j[I - d_{w_1,1}], \dots) & \text{if } \tilde{I} \in \tilde{\mathcal{I}}_1^{W_1} \text{ (L)} \wedge \hat{I} \in \hat{\mathcal{I}}_1^{W_1} \text{ (G)} \\ \vdots & \vdots \end{cases} \\
 \vdots & \\
 x_K[I] &= \begin{cases} \mathcal{F}_K^1(\dots, x_j[I - d_{1,K}], \dots) & \text{if } \tilde{I} \in \tilde{\mathcal{I}}_K^1 \text{ (L)} \wedge \hat{I} \in \hat{\mathcal{I}}_K^1 \text{ (G)} \\ \vdots & \vdots \\ \mathcal{F}_K^t(\dots, x_j[I - d_{t,K}], \dots) & \text{if } \tilde{I} \in \tilde{\mathcal{I}}_K^t \text{ (L)} \wedge \hat{I} \in \hat{\mathcal{I}}_K^t \text{ (G)} \\ \vdots & \vdots \\ \mathcal{F}_K^{W_K}(\dots, x_j[I - d_{W_K,K}], \dots) & \text{if } \tilde{I} \in \tilde{\mathcal{I}}_K^{W_K} \text{ (L)} \wedge \hat{I} \in \hat{\mathcal{I}}_K^{W_K} \text{ (G)} \end{cases} \quad (9)
 \end{aligned}$$

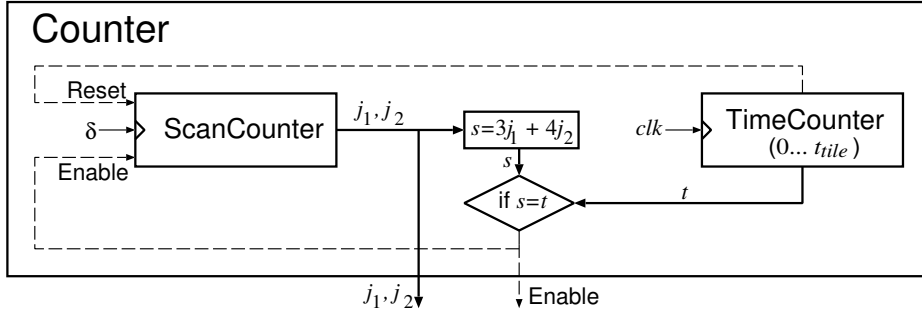


Figure 4: Enable mechanism: The ScanCounter implements the scanning pseudocode. The enable signal is turned off if $s \neq t$. The TimeCounter is incremented each clock cycle clk from zero to the time of execution of the last index point. The scan counter is incremented only every δ clock cycle, where δ denotes the *iteration interval* which is the number of time instances between the evaluation of two successive instances of a variable within one processing element.

with the index vector $I = (p \ t)^T$, where t is the time and $p = (p_1 \ p_2 \ \dots \ p_n)^T$ is the processor index, where normally $n = 1$ or 2 due to physical limitations. The “if” conditionals also known as *Housekeeping* code, describe the conditional execution of the recurrence equations. The “if” conditional for co-partitioning under type (L) are characterized by a processor index dependent equation (given $A_K \neq 0$) as in following Eq. (10) and therefore must be implemented in the local controller.

$$\text{if } \tilde{I} \in \tilde{\mathcal{I}} = \{\tilde{I} = (J \ p \ L)^T \in \mathbb{Z}^{3 \cdot n} \mid G\tilde{I} \geq g \Leftrightarrow A_J \cdot J + A_K \cdot p + A_L \cdot L \geq b\} \quad (10)$$

The “if” conditional under type (G) is described in the space $\hat{I} = (J \ L)^T$, explicitly describes those iterative conditionals that are independent of processor index p as shown in Eq. (11) and can therefore be implemented by a global controller.

$$\text{if } \hat{I} \in \hat{\mathcal{I}} = \{\hat{I} = (J \ L)^T \in \mathbb{Z}^{2 \cdot n} \mid G\hat{I} \geq g \Leftrightarrow A_J \cdot J \geq b_J \wedge A_L \cdot L \geq b_L\} \quad (11)$$

Let $G\tilde{I} \geq g$ be simplified so that it can be defined by a minimal number of m inequalities. Then the control needs to check whether the vector \tilde{I} is inside the polyhedron defined by the m inequalities. This is done by introducing m boolean variables $lctr^i$, which are ‘1’ only if $G_i\tilde{I} \geq g_i$. The control signals are generated for all LHS variables, x_k , $k = 1, \dots, K$. The following pseudo-code is the behavioral description of the generation of local control path and signals for processor p of type PE_q .

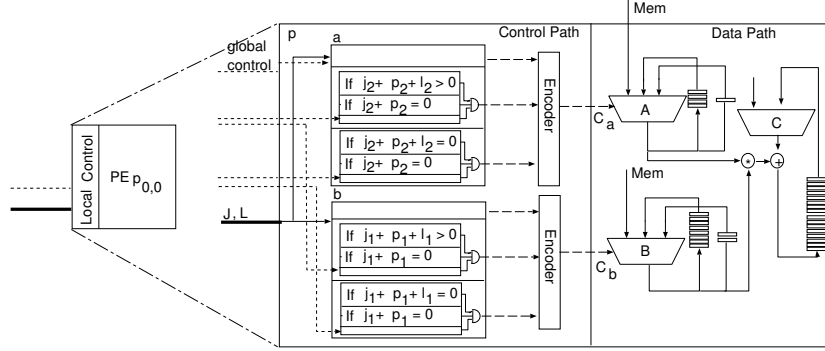


Figure 5: An example of a local control path of a PE as derived from the PLA in Eq. (12).

$$\begin{aligned}
lctr_{r,i}^l[I] &= \begin{cases} 1 & \text{if } G_{r,i}^l \cdot \tilde{I} \geq g_{r,i}^l \quad \forall r = 1, \dots, K \wedge i = 1, \dots, W_r \\ 0 & \text{else otherwise} \quad \wedge l = 1, \dots, m_{r,i} \end{cases} \\
lctr_{r,i} &= lctr_{r,i}^1 \wedge \dots \wedge lctr_{r,i}^{m_{r,i}} \quad \forall r = 1, \dots, K \wedge i = 1, \dots, W_r \\
&\quad \wedge m = m_{r,i} \\
C_1[I] &= \begin{cases} 0 & \text{if } lctr_{1,1} = 1 \wedge gctr_{1,1} = 1 \\ 1 & \text{if } lctr_{1,2} = 1 \wedge gctr_{1,2} = 1 \\ \vdots & \vdots \\ W_1 - 1 & \text{if } lctr_{1,W_1} = 1 \wedge gctr_{1,W_1} = 1 \end{cases} \\
&\vdots \\
C_K[I] &= \begin{cases} 0 & \text{if } lctr_{K,1} = 1 \wedge gctr_{K,1} = 1 \\ \vdots & \vdots \\ t - 1 & \text{if } lctr_{K,t} = 1 \wedge gctr_{K,t} = 1 \\ \vdots & \vdots \\ W_K - 1 & \text{if } lctr_{K,W_K} = 1 \wedge gctr_{K,W_K} = 1 \end{cases} \\
x_1[I] &= \text{SWITCH}(C_1[I] == 0, \mathcal{F}_1^1(\dots, x_j[I - d_{1,1}], \dots), \\
&\quad \dots, C_1[I] == W_1 - 1, \mathcal{F}_1^{W_1}(\dots, x_j[I - d_{W_1,1}], \dots)) \\
&\vdots \\
x_K[I] &= \text{SWITCH}(C_K[I] == 0, \mathcal{F}_K^1(\dots, x_j[I - d_{1,K}], \dots), \\
&\quad \dots, C_K[I] == W_K - 1, \mathcal{F}_K^{W_K}(\dots, x_j[I - d_{W_K,K}], \dots))
\end{aligned}$$

$I = (p t)^\top$, where $p \in PE_q$ and $\tilde{I} = (J p L)^\top$. J and L are index variables obtained from a global counter. $lctr_{r,i}$ denotes the local control signal for the i^{th} iterative conditional for variable x_r , which is obtained by AND relation of the m corresponding local control bits $lctr_{r,i}^l$. The mutual exclusivity of conditionals within a variable allows to encode the control variables in minimal bit encoding form in control variable $C_k[I]$. The global control signals, $gctr$ originating from the global controller are obtained by propagation from neighboring processors. The control signal is responsible for the selection of the appropriate input as dictated by the “if” statement.

\vdots
 /** Global control for type PEq**/

The hardware interpretation of the global controller for the matrix multiplication example is shown in Fig. 6. The global controller uses arithmetic and comparison operations for generating the control signals. Unlike the memory resources, the size of control unit is independent of the partitioning parameters. The local and global control units are problem size independent as the number of control variables are independent of the number of index points in any index space. The discussion in this section can be similarly modified for control generation of LSGP, LPGS, and other hierarchical partitioning methods. The propagation of the global control signals and counter variables to the individual PEs is discussed in the next section.

3.2.4 Propagation of Global Control Signals and Counter Values

The counter variables and the control signals are propagated through the processor array (see Fig. 2) instead of being broadcasted to the respective processor elements. The method presented below for finding appropriate propagation vectors and delay registers are discussed with reference to co-partitioning. However, a similar method can be used for other partitioning types. The global controller and counter are located next to the processor element which starts the execution. *Localization* (a transformation for converting global data dependencies into local dependencies) of control signals follows from $gctr(p, t) = gctr(0, t - \lambda_d)$ where λ_d is the number of delay registers or the number of time steps required by the global signals to travel to processor p . The equation is brought to a form where each PE receives the global signal from neighboring PEs as defined by $gctr(p, t) = gctr(p - d_p, t - \lambda_K \cdot d_p)$. The propagation vector, d_p is limited to $\{(1, 0), (0, 1), (1, 1), (-1, 0), (0, -1), (-1, 1), (1, -1), (-1, -1)\}$ for 2-d processor array. The selection of a propagation vector d_p for a PE(p) is obtained by looking at the start time of execution λ_s for each neighboring processor, $(p - d_p)$. The neighboring PE with λ_s less than start time of (p) and the difference being smallest is selected and the communication link is the propagation vector d_p . In case of a tie, the propagation vector is the same as the propagation vector for the neighboring PE if they are of same PE type. Otherwise the selection is done at random. This leads to regular circuit structure. Once the propagation vector d_p is found, the number of delay registers is found as $\lambda_K \cdot d_p$. For PE(0,0) the index variables and the global control signals are directly taken from the counter and global controller, respectively. The program for a PE after application of following algorithm incorporates the delay registers and propagation vectors.

Algorithm 2: Delay and propagation determination

INPUT: Processor space (\mathcal{P}), schedule vector (λ).

OUTPUT: Propagation vector (d_p), delay (λ_d). For all processing elements, $p \in \mathcal{P}$

For all possible propagation vectors, d_p ,

determine $\lambda_{(p-d_p)}$ (start time) using the schedule vector, i.e., $\lambda_{(p-d_p)} = \lambda_K \cdot (p - d_p)$.

Select d_p , s.t $\lambda_{p-d_p} < \lambda_p$ and $\lambda_{p-d_p} \geq \lambda_{p-s} \forall \lambda_{p-s} < \lambda_p$, where $s \in d_p$.

In case of tie, if p and $p - d_p$ are of the same PE type, then the propagation vector of p is the same as the propagation vector of PE $p - d_p$ else d_p is selected randomly.

ENDFOR

ENDFOR

4 A case study

Matrix multiplication was used as a case study for testing our methodology. The processor array specification is interpreted from the PLA after *control generation*. Afterwards, a VHDL implementation of the example co-partitioned matrix multiplication was carried out. The target FPGA architecture is a Xilinx Virtex XCV800. Table 2 summarizes the comparison of results from a multi-dimensional time implementation of matrix multiplication [19] and our implementation of co-partitioned matrix multiplication. The methodology for the generation of a controller for multi-

Table 2: Comparison of implementation for resource use by a single PE. Area complexity is expressed in terms of slices(4 LUTs).

Implementation	Control	Memory	Datapath	Clock
Multidimensional time	65 slices	2 RAM Blocks	26 slices	60 Mhz
Co-partitioned MM	12 slices	-	153 slices	58 Mhz

dimensional time implementation includes an automaton (counter) for scanning the space-time polyhedron within each PE thus accounting for the high cost of the control path as seen in Table 2. As compared, our methodology has a global counter which generates the iteration co-ordinates and the some of the common predicates thus leading to reduced area costs. Therefore, a control path area reduction of 90% is obtained. The absence of RAM blocks is explained by use of slices as registers to realize the local memory. That accounts for the large size of the data path. The global counter takes up 209 slices. The high cost of the global counter and controller is however an offset as by increasing the number of processor elements it is almost a constant cost.

5 Conclusions and Future Work

Our scheduling methodology for partitioning techniques enables the update of iteration co-ordinates in a global counter synchronous to the implementation as specified by the space-time mapping. Therefore, a considerable cost reduction in area of the control path is obtained as compared to the methodologies suggested in [6], [19] where a FSM (Finite State Machine) local to every PE is responsible for the generation of iteration co-ordinates. Furthermore, the control generation methodology can deal not only with LSGP, LPGS, multi-projection but also co-partitioning and other hierarchical partitioning methods with added advantage of using parallelepiped tiles of arbitrary shape. The area and speed trade-off obtained between calculation of control signals from

predicates or storage in a circular buffer needs to be studied with respect to resource constraints. The hardware interpretation of the program can be tuned to the architectures. For example, the propagation of global control signals can be optimized by having one global controller for each PE type in case of limited routing resources. Also, the optimal generation of address generation units and control units for PEs can be optimized by using an update scheme instead of re-computation. The results of our work are currently implemented in our design system [20].

References

- [1] Synfora, Inc.: (www.synfora.com)
- [2] Derrien, S., Risset, T.: Interfacing Compiled FPGA Programs: The MMAAlpha Approach. In: PDPTA. (2000)
- [3] Lengauer, C.: Loop Parallelization in the Polytope Model. In Best, E., ed.: CONCUR'93. Lecture Notes in Computer Science 715, Springer-Verlag (1993) 398–416
- [4] Teich, J., Thiele, L.: Control Generation in the Design of Processor Arrays. *Int. Journal on VLSI and Signal Processing* **3**(2) (1991) 77–92
- [5] Xue, J.: The Formal Synthesis of Control Signals for Systolic Arrays. PhD thesis, University of Edinburgh (1992)
- [6] Darte, A., Schreiber, R., Rau, B., Vivien, F.: Constructing and Exploiting Linear Schedules with Prescribed Parallelism. *ACM Transactions on Design Automation of Electronic Systems* **7**(1) (2002) 159–172
- [7] Teich, J., Thiele, L., Zhang, L.: Scheduling of Partitioned Regular Algorithms on Processor Arrays with Constrained Resources. *Journal of VLSI Signal Processing* **17**(1) (1997) 5–20
- [8] Hannig, F., Dutta, H., Teich, J.: Regular Mapping for Coarse-grained Reconfigurable Architectures. In: Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004). Volume V., Montréal, Quebec, Canada, IEEE Signal Processing Society (2004) 57–60
- [9] Wolfe, M.: High Performance Compilers for Parallel Computing. Addison-Wesley Inc. (1996)
- [10] Oldfield, J., Dorf, R.: Field Programmable Gate Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems. John Wiley & Sons, Chichester, New York (1995)
- [11] Eckhardt, U., Merker, R.: Hierarchical Algorithm Partitioning at System Level for an Improved Utilization of Memory Structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **18**(1) (1999) 14–24
- [12] Teich, J., Thiele, L.: Exact Partitioning of Affine Dependence Algorithms. In Deprettere, E.F., Teich, J., Vassiliadis, S., eds.: Embedded Processor Design Challenges. Volume 2268 of Lecture Notes in Computer Science (LNCS), Springer, Berlin (2002) 135–153
- [13] Hannig, F., Teich, J.: Design Space Exploration for Massively Parallel Processor Arrays. In Malyshkin, V., ed.: Parallel Computing Technologies, 6th International Conference, PaCT

- 2001, Proceedings. Volume 2127 of Lecture Notes in Computer Science (LNCS), Novosibirsk, Russia, Springer (2001) 51–65
- [14] Schrijver, A.: Theory of Linear and Integer Programming. Wiley – Interscience series in discrete mathematics. John Wiley & Sons, Chichester, New York (1986)
- [15] Darte, A., Khachiyan, L., Robert, Y.: Linear Scheduling is Nearly Optimal. Parallel Processing Letters **1**(2) (1991) 73–81
- [16] Quillere, F., Rajopadhye, S., Wilde, D.: Generation of Efficient Nested Loops from Polyhedra. International Journal of Parallel Programming **28**(5) (2000) 469–498
- [17] Bastoul, C.: Efficient Code Generation for Automatic Parallelization and Optimization. In: Int. Symposium on Parallel and Distributed Computing (ISPDC'03). (2003) 23–30
- [18] Feautrier, P.: Parametric Integer Programming. RAIRO Recherche Operationnelle **22** (1988) 243–268
- [19] Guillou, A., Quinton, P., Risset, T.: Hardware Synthesis for Multi-Dimensional Time,. IEEE computer Society, ASAP'2003 (2003)
- [20] PARO Design System Project: (www12.informatik.uni-erlangen.de/research/paro)