Generation of Distributed Loop Control⋆

Marcus Bednara, Frank Hannig, and Jürgen Teich

University of Paderborn, D-33098 Paderborn, Germany,
{bednara, hannig, teich}@date.upb.de,
URL: http://www-date.upb.de

Abstract. We present a new methodology for controlling the space-time behavior of VLSI and FPGA-based processor arrays. The main idea is to generate simple local control elements which take control over the active-ness of each attached processor element. Each control element thereby propagates a "start" and a "stop execution" signal to its neighbors. We show that our control mechanism is much more efficient than existing approaches because 1) only two control signals (start/stop) are required, 2) no extension of the computation space is necessary. 3) By the local propagation of just one start/stop signal, energy is saved as processing elements are only active between the time they have received the start signal and the time they have received the stop signal. Our methodology is applicable to one- and multi-dimensional processor arrays and is based on local control signal propagation. We provide a theoretical analysis of the overhead caused by the control structure.

1   Introduction

Parallel processor arrays and mapping methodology for such architectures are becoming more and more important, especially due to the advent of reusable reconfigurable hardware such as FPGAs [1,20], and due to the increasing amount of computing power implementable on a single chip (SoC-technology). Only if we are able to map computation intensive algorithms onto such architectures efficiently are we able to exploit the benefit of the given technology.

This paper is a contribution to implement computation intensive algorithms as specified by loop-like algorithms such as *piecewise regular algorithms* [16,17] or *uniform recurrence equations* [7] in hardware, see also [2,6].

The major task of mapping a given algorithm with indexed computations onto hardware is to assign space (processor index) and time to each computation in a computational domain that is often called *index space*. In the area of VLSI processor arrays, linear index transformations [8,10,11] are typically used and known as *space-time mappings*. Similar approaches are also used in the area of parallelizing compilers for supercomputers with linear transformations such as *loop skewing*, *loop tiling*, and *loop permutation* [19].

The problem addressed in this paper is in the context of controlling the operations of a given algorithm when being linearly transformed by a space-time mapping. Let the index space be described by a bounded polyhedron (polytope), then a space-time mapping leads to a skewed polytope. In such a polytope, the operations assigned to each processing element do not necessarily start executing at the same time 0. Hence, one could either apply techniques of control generation such as described in [13, 14, 21] that have in common that the index space is extended by dummy operations such that all processors start at the reset time $t = 0$. In order to control correct execution, a control signal is generated for each bounding hyperplane of the transformed index space such that predicates in control signals help a processing element to identify whether and what operation it has to perform at each time step.

This is, however, inefficient in the sense that the number of control signals generated depend on the number of bounding hyperplanes of the algorithm and that much energy is consumed by extension of index spaces and by the introduction of propagated control signals.

Here, we present a new idea of efficient controlling the boundaries of a space-time transformed index space polytope that introduces only two additional signals that indicate at each processing element the first time step and the last time step it will have to execute an operation. These two signals are propagated locally to neighbor processing elements. The approach is energy-efficient in the sense that a processing element only consumes energy during the time interval between reception of its start and its stop signal.

The rest of the paper is structured as follows. First, in Section 2, we show how the principle works for 1-D arrays (2-D index spaces). In Section 3, we extend this generation of boundary control hierarchically to higher dimensions including optimization techniques to reduce the control overhead in terms of number of control elements. Finally, results are presented in Section 4.

2 Control Mechanism for Linear Processor Arrays

First, we present our approach to controlling the activity of linear processor arrays. Let $\mathcal{I}$ denote a two-dimensional convex index space defined as follows:

$$\mathcal{I} = \{I \in \mathbb{Z}^d \mid AI \geq b\}, \quad A \in \mathbb{Z}^{m \times d}, \ b \in \mathbb{Z}^m \tag{1}$$

and let $d = 2$. Hence, $\mathcal{I}$ is the set of integral points in the intersection of m two-dimensional halfspaces: $\mathcal{I} = \bigcap_{i=0}^{m-1} H_i$. For each halfspace $H_i = \{I \in \mathbb{Z}^2 \mid A_i I \geq b_i\}$, $i = 0 \ldots m-1$, its *bounding hyperplane* P_i is $P_i = \{I \in \mathbb{Z}^2 \mid A_i I = b_i\}$ where $A_i = (A_{i,p} \ A_{i,t})$ is the i-th row vector of matrix A and b_i is the i-th element of vector b. For each index vector $I = (p \ t)^{\mathrm{T}} \in \mathcal{I}$, p denotes the *processor index* and t the (discrete) *time index* or simply time. Hence, a space-time mapping of the operations inside the polytope is assumed to be given.

If the element $A_{i,t} \neq 0$, P_i can be written as a linear equation:

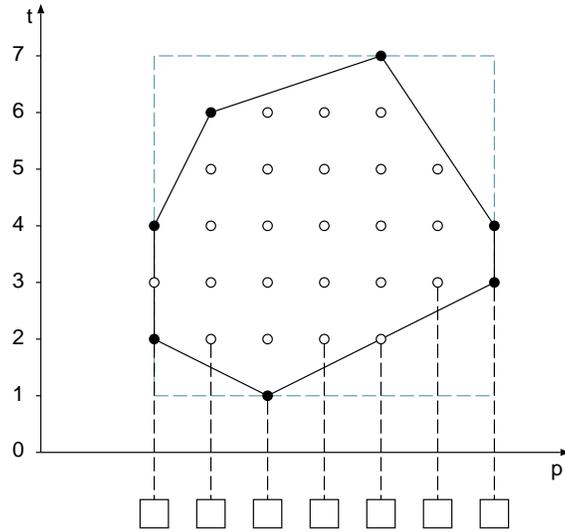$$t = T_i(p) := -\frac{A_{i,p}}{A_{i,t}}p + \frac{b}{A_{i,t}} \tag{2}$$

Fig. 1. Index space $\mathcal{I}$ with bounding hyperplanes. Each integer point corresponds to the computation of a loop body.

In Eq. (2), t denotes the time at which the bounding hyperplane P_i is crossed in dependence of the processor index p. If $A_{i,t} = 0$, then p is constant.

Fig. 1 shows an index space $\mathcal{I}$ with 7 bounding hyperplanes. Each integral point inside or on the border of the index space corresponds to a computation of a loop-body of a given algorithm. For the purpose of this paper, it does not matter how complex and what type of computation is specified at each index point. In Fig. 1, we have already given a space-time mapping: The p-axis denotes the processor index, and t denotes the time of a computation. By the projection of the polytope along the time-axis, it can be seen that the corresponding processor arrays consists of 7 processing elements.

2.1 Boundary Control

It can be seen in Fig. 1 that some processor elements start their first computation at different times than others. A similar observation holds for the last time, a processor performs a computation inside the index space polytope. Hence, control is necessary that tells each processing element when to start executing and when to stop executing.

In previous approaches to *control generation* such as [13–15], it was discovered that the space-time-transformed index space of an algorithm that corresponds to a synchronously clocked processor array reset at time 0 must be in the shape of a *right prism* as indicated by dashed lines in Fig. 1 for the index space shown there. The authors proposed to specify an *extension of index space* to such a right prism hull and to provide control signals that propagate control signals from

the border of the array that provide a processing element with the information whether it computes an operation inside $\mathcal{I}$ or not. The complexity of this kind of boundary control in terms of the number of control signals is m. Another disadvantage of the control proposed in [14, 21] is that the m generated control signals are propagated in the complete space $\mathcal{I}$, hence causing an overhead of $m|\mathcal{I}|$ propagations that cause a lot of energy consumption.

In the following, we develop our main idea of *boundary control* by propagating just a signal start and a single stop signal to the processing elements, and only once during the complete algorithm execution. Before that, some definitions are in order.

We assume that $\mathcal{I}$ is an integral polytope. In this case, all its vertices of $\mathcal{I}$ are integral points. Otherwise, we may apply the *cutting plane algorithm* according to [12] to compute the integer hull.[1] The *vertex set* V of $\mathcal{I}$ is defined as follows:

$$V = \mathcal{I} \cap \left\{ \bigcup_{0 \leq i,j < m, i \neq j} (P_i \cap P_j) \right\}. \tag{3}$$

For example, the vertex set of the index space shown in Fig. 1 is highlighted by black dots.

In the following, let the processor index of $v_i \in V$ be denoted p_i and the time index be denoted t_i ($v_i = (p_i \ t_i)^{\mathrm{T} \in \mathcal{I}}$). We define

$$\mathcal{I}_{t_{\min}} = \{v_i \in V : \forall v_j \in V, v_j \neq v_i : t_i \leq t_j\}$$
$$\mathcal{I}_{t_{\max}} = \{v_i \in V : \forall v_j \in V, v_j \neq v_i : t_i \geq t_j\}$$
$$\mathcal{I}_{p_{\min}} = \{v_i \in V : \forall v_j \in V, v_j \neq v_i : p_i \leq p_j\}$$
$$\mathcal{I}_{p_{\max}} = \{v_i \in V : \forall v_j \in V, v_j \neq v_i : p_i \geq p_j\}$$

Hence, $\mathcal{I}_{t_{\min}}$ denotes the set of vertices with the property that there is no vertex with a smaller time step, and so on. Let $I_{t_{\min}}$ ($I_{t_{\max}}, I_{p_{\min}}, I_{p_{\max}}$) be an arbitrary element of $\mathcal{I}_{t_{\min}}$ ($\mathcal{I}_{t_{\max}}, \mathcal{I}_{p_{\min}}, \mathcal{I}_{p_{\max}}$). Finally, we define two predicates TLO and THI on index vectors as follows:

$$\mathrm{TLO}(I) \quad \Leftrightarrow \quad I = \begin{pmatrix} p \\ t \end{pmatrix} \in \mathcal{I} \ \wedge \ \forall \begin{pmatrix} p \\ t' \end{pmatrix} \in \mathcal{I} \ : \ t \leq t'$$

$$\mathrm{THI}(I) \quad \Leftrightarrow \quad I = \begin{pmatrix} p \\ t \end{pmatrix} \in \mathcal{I} \ \wedge \ \forall \begin{pmatrix} p \\ t' \end{pmatrix} \in \mathcal{I} \ : \ t \geq t'$$

In other words, $\mathrm{TLO}(I)$ ($\mathrm{THI}(I)$) is true for an index point $I = (p \ t)^{\mathrm{T}}$ iff $I \in \mathcal{I}$ and there is no point $I' = (p \ t')^{\mathrm{T}}$ in $\mathcal{I}$ at the same processor index p with a smaller (bigger) time step than t.

---

[1] Although the cutting plane algorithm has exponential complexity, it is in general possible to apply the algorithm for low-dimensional index spaces (e.g., $d = 2, 3$), and the number of half-spaces is small in practical applications.
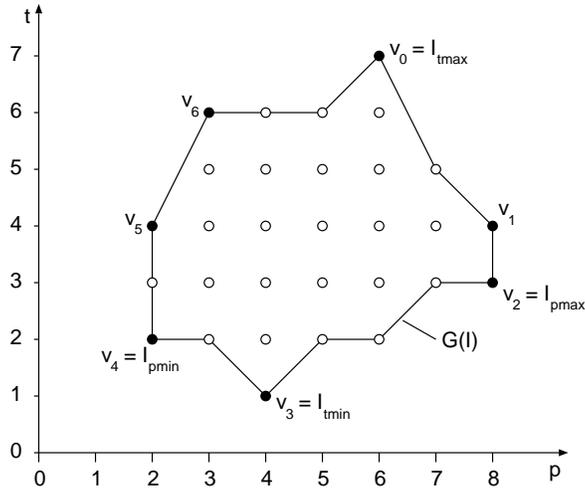
Fig. 2. Index space $\mathcal{I}$ with vertex set $V = \{v_0, \ldots, v_6\}$ and boundary control graph $G(\mathcal{I})$.

We use these predicates in order to define the *boundary control graph* $G(\mathcal{I}) = (V_G(\mathcal{I}), E_G(\mathcal{I}))$ with vertices $V_G(\mathcal{I})$ and edges $E_G(\mathcal{I})$:

$$V_G(\mathcal{I}) = \{I \in \mathcal{I} \mid \mathrm{TLO}(I) \vee \mathrm{THI}(I)\}$$

$$E_G(\mathcal{I}) = \left\{ \left(I_1 = \begin{pmatrix} p_1 \\ t_1 \end{pmatrix} \in V_G(\mathcal{I}), I_2 = \begin{pmatrix} p_2 \\ t_2 \end{pmatrix} \in V_G(\mathcal{I}) \right) \mid \right.$$
$$((|p_2 - p_1| = 1 \ \wedge \ (\mathrm{TLO}(I_1) = \mathrm{TLO}(I_2))) \vee$$
$$\left. (p_2 = p_1 \ \wedge \ ((\mathrm{TLO}(I_1) = \mathrm{THI}(I_2)) \ \vee \ (t_1 = t_2)))) \right\}$$

In Fig. 2, a sample index space $\mathcal{I}$ and the corresponding graph $G(\mathcal{I})$ is shown. Since $\mathrm{TLO}(I_{t_{\min}})$ and $\mathrm{THI}(I_{t_{\max}})$ holds, $I_{t_{\min}}, I_{t_{\max}} \in V_G(\mathcal{I})$.

The idea of boundary control is now to traverse the boundary control graph node by node, starting in a node $I_{t_{\min}} \in \mathcal{I}_{t_{\min}}$ in two directions (L for left and R for right) as indicated in Fig. 3 until a node $I_{t_{\max}} \in \mathcal{I}_{t_{\max}}$ is reached. These two paths can be represented as vectors, where each vector component is a pair $(I_i, I_j) \in V_G(\mathcal{I}) \times V_G(\mathcal{I})$ that denotes a *directed* edge from I_i to I_j:

$$L = (l_0, \ldots, l_{k-1}) \ : \ \forall i \in \{0, \ldots, k-1\} \ : \ l_i = (I_{i,1}, I_{i,2})$$
$$\wedge \ \{I_{i,1}, I_{i,2}\} \in E_G(\mathcal{I}) \ \wedge \ I_{i+1,1} = I_{i,2} \ \wedge \ I_{0,1} = I_{t_{\min}}$$
$$\wedge \ I_{k-1,2} = I_{t_{\max}} \ \wedge \ \exists j \in \{0, \ldots, k-1\} \ : \ I_{j,1} = I_{p_{\min}};$$

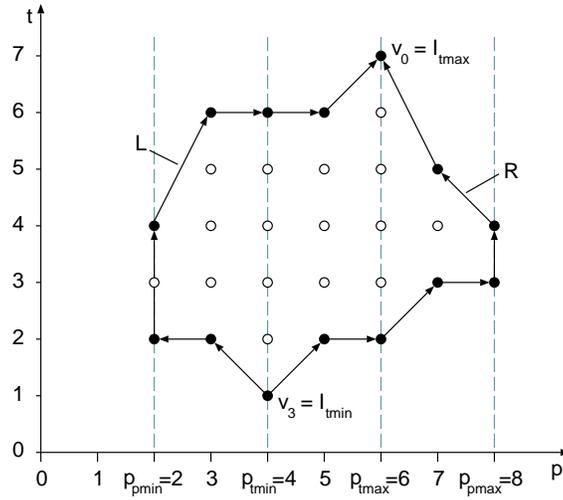$$R = (r_0, \ldots, r_{k'-1}) \ : \ \forall i \in \{0, \ldots, k'-1\} \ : \ r_i = (I_{i,1}, I_{i,2})$$

Fig. 3. Index space $\mathcal{I}$ with boundary control paths L and R.

$$\wedge \ \{I_{i,1}, I_{i,2}\} \in E_G(\mathcal{I}) \ \wedge \ I_{i+1,1} = I_{i,2} \ \wedge \ I_{0,1} = I_{t_{\min}}$$
$$\wedge \ I_{k'-1,2} = I_{t_{\max}} \ \wedge \ \exists j \in \{0, \dots, k'-1\} \ : \ I_{j,1} = I_{p_{\max}}.$$

The traces L and R are called *boundary control paths*. Each vector element of a boundary control path represents the flow of a control signal from a *control element* to the next. The implied hardware structure of control elements is defined in the next subsection.

2.2 Control Hardware

All index vectors $I = (p \ t)^{\mathrm{T}}$ with the same p-coordinate are mapped on the same processor element. Thus, the processor elements of operations at index points $I_{t_{\min}}$, $I_{t_{\max}}$, $I_{p_{\min}}$, $I_{p_{\max}}$ are mapped onto processor indices $p_{t_{\min}}$, $p_{t_{\max}}$, $p_{p_{\min}}$, $p_{p_{\max}}$, respectively.

The hardware structure of our control methodology is defined as follows: To each processing element, we associate a dedicated controller called *control element* (CE). Each control element has exactly two control input/output pairs. Control elements are interconnected locally with neighbor control elements in the form of a *control chain* as indicated in Fig. 4, showing the control chain for the index space in Fig. 2.

Thereby, each trace L and R represents a control signal flow from $p_{t_{\min}}$ to $p_{t_{\max}}$. A control signal that initiates the execution of a processing element is passed from $p_{t_{\min}}$ to $p_{p_{\min}}$ and $p_{p_{\max}}$. Each processor element in between starts its operation once receiving this start signal. When a control signal is passed
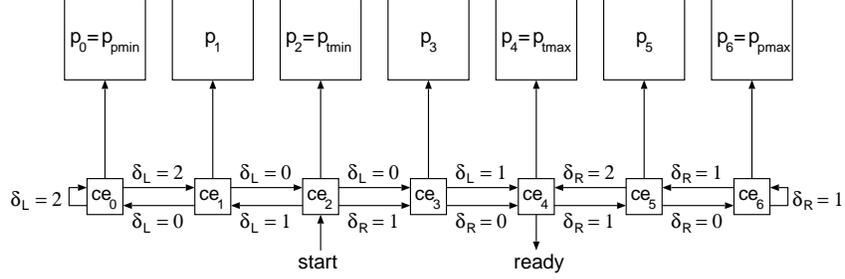
Fig. 4. Control chain for the index space in Fig.2. The data links between the processor elements are not shown here.

back from $p_{p_{\min}}$ and $p_{p_{\max}}$ to $p_{t_{\max}}$, the processor elements in between stop their execution (stop signal).

The CE at location $p_{t_{\min}}$ has a start-input which must be set active for one clock cycle when the computation of the array starts. Two control signals are duplicated from the start signal and propagated to the first and the last CE in the control chain (at $p_{p_{\min}}$ and $p_{p_{\max}}$) and then back to the CE $p_{t_{\max}}$. The control signals are delayed in each CE, and both control signals must arrive at $p_{t_{\max}}$ at the same time step. Thus, the sum of the delays must be the same for both control signal paths. By determining an appropriate delay for both control signals when passing from one CE to the next, we obtain a control mechanism for controlling the boundary of an index space $\mathcal{I}$.

Each CE thus handles two control signals, thus it has two control input/output pairs, except for the CEs connected to $p_{t_{\min}}$ and $p_{t_{\max}}$ which are slightly different. A control signal on an input is passed to the corresponding output after a delay of δ cycles. An additional control output (*enable*) is connected to the attached processor element. This enable signal is set active when the first (start) control signal arrives and is reset when the second control signal (stop) arrives. The processor element is active as long as the enable signal is set and stopped else.

2.3 Computation of δ-Delays

The number δ of time steps a control signal must be delayed before passed to the next CE depends on the number of isotemporal hyperplanes the corresponding edge $e = (v, w) \in E_G(\mathcal{I})$ crosses the index space $\mathcal{I}$. Thus, we define functions $delay_L : 0, \ldots, k-1 \rightarrow \mathbb{N}^3$ and $delay_R : 0, \ldots, k'-1 \rightarrow \mathbb{N}^3$ that return the triples (i, j, δ_L) and (i, j, δ_R) with the following semantics: A control signal passed from the CE at p_i to the CE at p_j must be delayed for δ_L (δ_R) time steps. Remember that each vector element l_i (r_i) of the vector L (R) is of the form $((p_{i,2} \ t_{i,2})^{\mathrm{T}}, (p_{i,1} \ t_{i,1})^{\mathrm{T}})$.

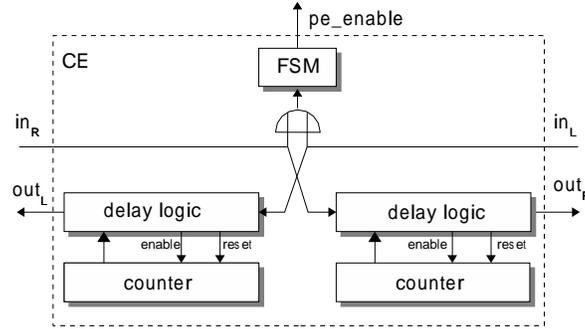$$delay_L(i) = (p_{i,1}, p_{i,2}, (t_{i,2} - t_{i,1}))$$

Fig. 5. Internal logic of a control element

$$delay_R(j) = (p_{j,1}, p_{j,2}, (t_{j,2} - t_{j,1}))$$

Computing both $delay_L$ and $delay_R$ on their whole domains gives us the complete structure of the control chain and all necessary δ_L and δ_R delays of the CEs.

2.4 Architecture of the Control Elements

A refinement of the internal circuitry of a control element is shown in Fig. 5. The CE consists of two binary counters used for realizing the delay of the control signals. Each counter is controlled by a delay logic that starts the counter when a control signal event occurs at the in_L (in_R) input. The counter is stopped and reset after δ_L (δ_R) clock cycles and the out_L (out_R) output is set active for one clock cycle. A small state machine (FSM) is used to generate the signal pe_enable, which is set active when the in_L (in_R) signal arrives and is reset when the in_R (in_L) signal arrives.

The CE architecture differs from that shown in Fig. 5 at processor locations $p_{t_{\min}}$ and $p_{t_{\max}}$ since they have an additional start input or ready output. In the case of $p_{t_{\min}} = p_{t_{\max}}$, we use another type of control element that has both start and stop signals.

The computed delays for the space-time transformed index space introduced in Fig. 2 are represented by the slope of the edges of the boundary control paths in Fig. 3. I.e., the number of isotemporal hyperplanes crossed by each edge corresponds to the δ-delay. The δ-delays are also annotated to the interconnect of the control elements in Fig. 4.

3 Boundary Control for Multi-Dimensional Arrays

The methodology of boundary control presented in Section 2 is applicable only to two-dimensional index spaces (1-D arrays). Here, we show how the concept may be extended to multi-dimensional arrays by a hierarchical application of the principle introduced earlier.
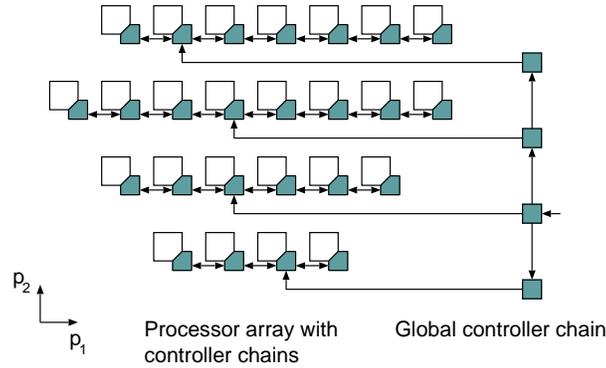
p_2

p_1

Processor array with   Global controller chain
controller chains

Fig. 6. Controller structure of a two-dimensional processor array. The data links between the processor elements are not shown here.

3.1 Multi-dimensional Control

An extension of our model to algorithms with index spaces of higher dimensions is possible due to the observation that each n-dimensional convex polytope can be decomposed into a finite set of $n-1$-dimensional convex polytopes by slicing it using a hyperplane partitioning. Such a slicing algorithm will be used in this section to extend the method of boundary control to higher-dimensional index spaces.

For example, a space-time mapped 3-D index polytope corresponds to a two-dimensional processor array. A hyperplane partitioning in 3-D space divides the 2-D processor space into a set of 1-D (linear arrays), see, e.g., in Fig. 6. For each of the partitions, we may apply the boundary control of the 2-D polytopes as introduced in Section 2. Finally, an additional controller is required that generates the *start*-signal for each of the linear control chains. For this purpose, we introduce another global controller chain. A sample control structure of a two-dimensional processor array is shown in Fig. 6.

The number of required additional control elements obviously depends on the orientation of the slices of the original polytope. If the slices were oriented in vertical direction, in Fig. 6, 8 additional control elements instead of 4 would be required. Other hyperplane directions used for slicing the 3-D space would lead to different directions of controller chains and different numbers of required global control elements.

3.2 Hyperplane Partitioning Algorithm

Our slicing algorithm is applicable for any convex index space of arbitrary dimension $d > 1$. Actually, for illustrative reasons, the algorithm is described here only for $d = 3$. The algorithm decomposes a given index space $\mathcal{I}$ into a finite number of parallel, $d - 1$ dimensional hyperplanes, such that

- the hyperplane normal vector chosen is not equal to the time axis, and
- the number of hyperplane partitions is minimal.[2]

The last condition minimizes the number of additional control elements.

Optimal Hyperplane Direction

Given an index space $\mathcal{I}$ according to Def. 1. Let the index space be space-time transformed. This means that without loss of generality, we may assume that the index points $I = (p_0 \ p_1 \ \ldots \ p_{d-1} \ t)^{\mathrm{T}}$ have $d-1$ leading processor indices and the time t as the d-th coordinate.

The first step in our algorithm is to find the slice-minimal orientation of the hyperplanes. Let $v = (v_0 \ v_1 \ \ldots \ v_{d-1} \ v_t)$ be the normal vector of feasible hyperplanes (being orthogonal to the time axis). This problem is very similar to finding an optimal schedule vector for processor arrays, see [4, 18], and can be solved as follows: Find a vector v that minimizes the maximum number of hyperplanes, i.e., that minimizes $\max\{v(I_2 - I_1) \mid AI_1 \geq b \ \wedge \ AI_2 \geq b\}$. This problem can be represented in its dual form as one single linear program as follows:

$$
\begin{aligned}
\text{minimize } & -(y_2 + y_1)b \\
\text{subj. to } & y_1 A = -v \quad y_1 \geq 0 \quad y_1 \in \mathbb{Q}^{1 \times m} \\
& y_2 A = +v \quad y_2 \geq 0 \quad y_2 \in \mathbb{Q}^{1 \times m} \\
& v_t = 0.
\end{aligned}
\tag{4}
$$

This linear program may be solved in polynomial time.

For example, for the processor space shown in Fig. 6, the optimal hyperplane obtained is $v = (0 \ 1 \ 0)$ leading to a hyperplane with 4 control elements.

The purpose of the top-level linear array of control elements is to inject only start signals to the local linear control chains that are obtained by the hyperplane partitioning. This is due to the fact that the control elements at the lowest level generate stop signals to finish operations which are not needed at higher levels. Hence, the control hierarchy is obtained by having one processing element at which the start signal is injected. This element, see e.g., in Fig. 6, is obtained as belonging to a hyperplane with a processing element that starts at a globally minimal time step. All neighbor elements obtain a delayed start signal. On the top-level, this delay is given by the difference between the minimal time step of one hyperplane and the minimal time step of the index space in the neighbor partition.

In case of more than 3 dimensions, the hyperplane method can be applied iteratively to obtain a control hierarchy with one linear array at the top-level, each element starts a linear control array at the next lower level, and so on, until a control chain at the lowest level is started.

---

[2] In the following, we denote the intersection of a hyperplane with a given index space *slice*.

For completeness, we only specify next how we obtain the description of the slices.

Description of Slices

With a found hyperplane direction v, we can process on the next lower hierarchical level by a) computing all non-empty slices, and b) for each slice, apply the boundary control to each slice of dimension $d-1$.

In the following, we simply present an algorithm for computing the descriptions of each slice, given an index space $\mathcal{I}$ and a hyperplane partitioning vector v.

All non-empty slices satisfy:

$$vI = z$$
$$AI \geq b.$$

The first slice is obtained by solving the linear program

$$z_{\min} = \min\{vI \mid AI \geq b\}.$$

The last slice is obtained by solving the linear program

$$z_{\max} = \max\{vI \mid AI \geq b\}.$$

By the convex nature of the index space, there is a non-empty slice for each value $z_{\min} \leq z \leq z_{\max}$. Each hyperplane description is given by $\{I \in \mathbb{Z}^d \mid vI = z \land AI \geq b\}$. In order to apply the control to the next lower level of hierarchy, we need to eliminate a variable of I to obtain a description with $d-1$ variables. This can be done easily by choosing an arbitrary variable of the index vector I, e.g., p_0, solve $vI = z$ for p_0, i.e.,

$$p_0 = -\frac{v_1}{v_0}p_1 - \frac{v_2}{v_0}p_2 - \cdots - \frac{v_{d-1}}{v_0}p_{d-1},$$

and replace p_0 in $AI \geq b$ with the right hand side of the above equation.

4 Results

In this section, the methodology of our approach is shown for a realistic example. As instance, we consider the well-known problem of LU-decomposition. In the LU-decomposition, a given matrix $C \in \mathbb{R}^{N \times N}$ is decomposed into $C = A \cdot B$, where $A \in \mathbb{R}^{N \times N}$ is a lower triangular and $B \in \mathbb{R}^{N \times N}$ is an upper triangular matrix. This problem may be formulated by a recursive algorithm [9]. A corresponding piecewise regular algorithm is given by the following set of quantified equations

$$b\,[i,j,k] \quad \leftarrow \quad \begin{cases} c\,[i,j,k-1] & \text{if } i = k \\ b\,[i-1,j,k] & \text{else} \end{cases}$$

$$a\,[i,j,k] \quad \leftarrow \quad \begin{cases} c\,[i,j,k-1]\,/b\,[i,j,k] & \text{if } j=k \\ a\,[i,j-1,k] & \text{else} \end{cases}$$

$$c\,[i,j,k] \quad \leftarrow \quad c\,[i,j,k-1] - a\,[i,j,k] \cdot b\,[i,j,k]$$

where the index space is given by

$$\mathcal{I} \;=\; \left\{ I = \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in \mathbb{Z}^3 \; \middle| \; \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \\ k \end{pmatrix} \geq \begin{pmatrix} 0 \\ 1-N \\ 0 \\ 1-N \\ 0 \\ 1-N \end{pmatrix} \right\}.$$

For illustration purposes let be $N = 5$ in the following. As mentioned before, linear transformations as in Equation (5) are used as *space-time mappings* [10,11] in order to assign a *processor index* $p \in \mathbb{Z}^{n-1}$ (space) and a *sequencing index* $t \in \mathbb{Z}$ (time) to index vectors $I \in \mathcal{I}$.

$$\begin{pmatrix} p \\ t \end{pmatrix} \;=\; \begin{pmatrix} Q \\ \lambda \end{pmatrix} I \tag{5}$$

In Eq. (5), $Q \in \mathbb{Z}^{(n-1)\times n}$ and the *schedule vector* $\lambda \in \mathbb{Z}^{1 \times n}$.

Now, consider that we have obtained an optimal space-time mapping by exploring the design space in terms of cost and performance. In [6], we have presented efficient pruning techniques for the search of optimal projection vectors (space-time mappings). We only summarize the main ideas: 1) Only consider co-prime vectors, 2) only consider co-prime vectors that have the properties that at least two points in $\mathcal{I}$ are projected onto each other. This leads to a search space of co-prime vectors in a convex polytope called difference-body of points in $\mathcal{I}$. Finally, in this reduced search space, we can exploit symmetry to exclude search vectors $v = -v'$ such that typically, only few projection vector candidates v have to be investigated. In order to count the number of points in a projected index space, *Ehrhart polynomials* [3, 5] may be evaluated to count the number of points (control elements) in the projected space.

Let $u = (0\ 1\ 0)^{\mathrm{T}}$ be a projection vector and $\lambda = (0\ 1\ 2)$ be a schedule vector candidate during exploration. Herewith, index points I are mapped onto processors p at time t

$$\begin{pmatrix} p \\ t \end{pmatrix} \;=\; \begin{pmatrix} 1\ 0\ 0 \\ 0\ 0\ 1 \\ 0\ 1\ 2 \end{pmatrix} I.$$

This point set can also be described by the following polytope $\mathcal{I}_{pt}$

$$\mathcal{I}_{pt} \;=\; \left\{ I = \begin{pmatrix} i \\ k \\ t \end{pmatrix} \in \mathbb{Z}^3 \; \middle| \; \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -2 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} i \\ k \\ t \end{pmatrix} \geq \begin{pmatrix} -4 \\ -4 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right\}.$$
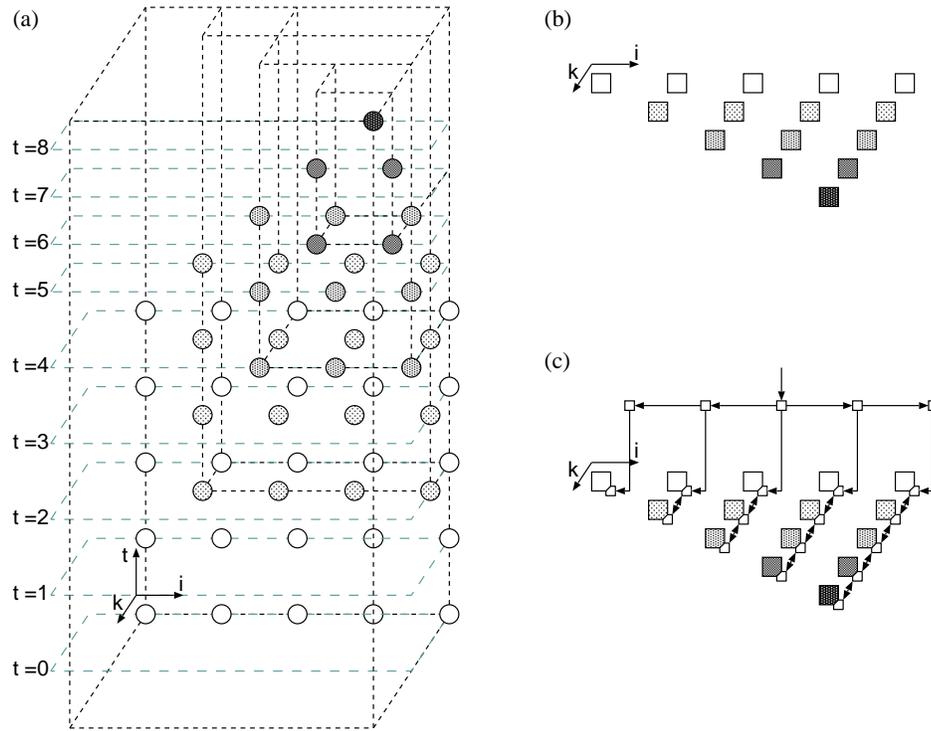
Fig. 7. In (a), the mapped index space of the LU-decomposition is shown, where $N = 5$, $u = (0\ 1\ 0)^{\mathrm{T}}$, and $\lambda = (0\ 1\ 2)$. The isotemporal hyperplanes are visualized. The matching processor array is shown in (b). In (c), the control structure for the processor array is shown.

The transformed index space is visualized in Fig. 7 (a), its shape is like a skewed pyramid. The triangular shape of the processor array is shown in Fig. 7 (b). For example, at time step $t = 0$ and $t = 1$ only five processor elements are making computations (marked as white points in Fig. 7), at time $t = 2$ and $t = 3$ in total nine processors are used, at time $t = 4$ twelve processors, at time $t = 5$ seven processors, and so on.

Next, an optimal hyperplane direction is determined. By solving the corresponding linear program (see Eq. (4)), the normal vector $v = (1\ 0\ 0)$ is obtained as an optimal solution vector. Using this direction all processor with $k = 0$ can be started at time 0. Therefore, a broadcast is sent to all the white processor elements in Fig. 7 (c). In Fig. 8 (a), an exemplary slice for $v = (1\ 0\ 0)$ and $i = 4$ of the mapped index space is shown. The boundary control paths are

$$
L = \left(\left(\begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \\ 4 \end{pmatrix} \right), \left(\begin{pmatrix} 4 \\ 0 \\ 4 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \\ 5 \end{pmatrix} \right), \left(\begin{pmatrix} 4 \\ 1 \\ 5 \end{pmatrix}, \begin{pmatrix} 4 \\ 2 \\ 6 \end{pmatrix} \right), \ldots \right.
$$

$$\ldots, \left(\begin{pmatrix} 4 \\ 2 \\ 6 \end{pmatrix}, \begin{pmatrix} 4 \\ 3 \\ 7 \end{pmatrix} \right), \left(\begin{pmatrix} 4 \\ 3 \\ 7 \end{pmatrix}, \begin{pmatrix} 4 \\ 4 \\ 8 \end{pmatrix} \right) \right)$$

for the left path and

$$R = \left(\left(\begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \\ 2 \end{pmatrix} \right), \left(\begin{pmatrix} 4 \\ 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 4 \\ 2 \\ 4 \end{pmatrix} \right), \left(\begin{pmatrix} 4 \\ 2 \\ 4 \end{pmatrix}, \begin{pmatrix} 4 \\ 3 \\ 6 \end{pmatrix} \right), \ldots \right.$$

$$\ldots, \left(\begin{pmatrix} 4 \\ 3 \\ 6 \end{pmatrix}, \begin{pmatrix} 4 \\ 4 \\ 8 \end{pmatrix} \right) \right)$$

for the right path. The δ-delays that are necessary for controlling the boundary of the slice are computed as follows for the left path L

$$\begin{aligned}
\text{delay}(0) &= (p_{40}, p_{40}, 4 - 0), & \delta_L &= 4 \\
\text{delay}(1) &= (p_{40}, p_{41}, 5 - 4), & \delta_L &= 1 \\
\text{delay}(2) &= (p_{41}, p_{42}, 6 - 5), & \delta_L &= 1 \\
\text{delay}(3) &= (p_{42}, p_{43}, 7 - 6), & \delta_L &= 1 \\
\text{delay}(4) &= (p_{43}, p_{44}, 8 - 7), & \delta_L &= 1,
\end{aligned}$$

and for the right path R

$$\begin{aligned}
\text{delay}(0) &= (p_{40}, p_{41}, 2 - 0), & \delta_R &= 2 \\
\text{delay}(1) &= (p_{41}, p_{42}, 4 - 2), & \delta_R &= 2 \\
\text{delay}(2) &= (p_{42}, p_{43}, 6 - 4), & \delta_R &= 2 \\
\text{delay}(3) &= (p_{43}, p_{44}, 8 - 6), & \delta_R &= 2.
\end{aligned}$$

The controlled processors are shown in Fig. 8 (b), the edges between the controller elements are annotated by the delays δ_L and δ_R respectively. At time 0 the processor element p_{40} is enabled by its controller. After two time steps ($\delta_R = 2$) the next processor element p_{41} is enabled. Every second time unit the start signal is propagated two the right neighbor and the corresponding processor is enabled. The first stop signal is produced at time step 4 where processor p_{40} is disabled. Then, one processor element from p_{41} to p_{44} is turned off at each subsequent time step.

The utilization of our new control methodology for the LU-decomposition algorithm shows several advantages very well compared to the classically approaches of control generation [13, 14, 21]. An extension of a given index space to a right prism is not necessary. In our case of the LU-decomposition, only approximately half of the number of data transfers (propagations) are necessary. The energy reduction is evident.
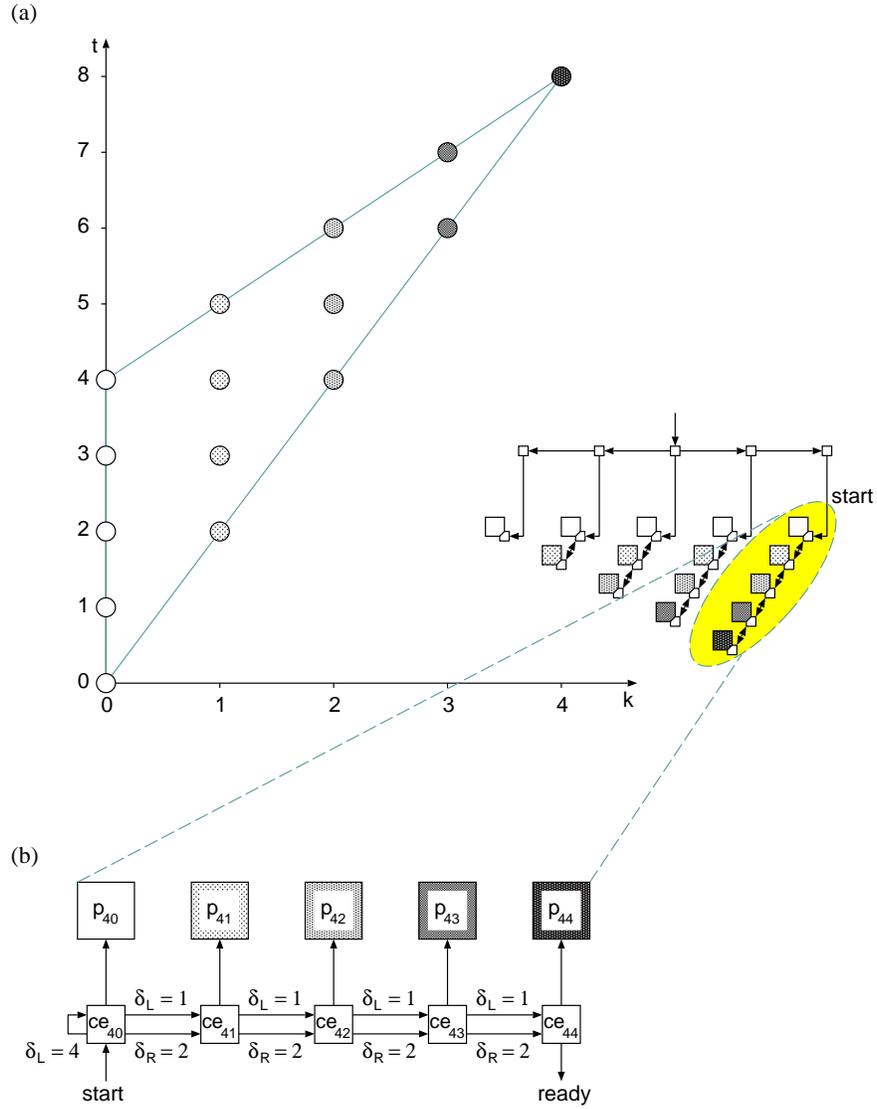
(a)

(b)

Fig. 8. In (a), a slice for $i = 4$ of the mapped index space $\mathcal{I}_{pt}$ is visualized. In (b), the corresponding controller structure of the processor array is shown. The edges between the controller elements are annotated by the δ-delays. The lower edges between the control elements propagate every second time step a start signal to its right neighbor. After the delay $\delta_L = 4$, subsequently the stop signal is sent from one control element to its right neighbor.

5   Conclusion

In this paper, we have introduced a new method for controlling the operations of loop-like algorithms when mapped onto processor arrays. The idea is to reconstruct the border of an index polytope of computation by propagation of a single start and a single stop event locally to neighbor processors that identifies the execution intervals of each processor. The effort amounts to the fixed amount of two control signals. Contrary to existing approaches, our approach is much more area- and energy-conscious.

Our method can be seen complementary to existing control generation approaches that are still necessary, e.g., for replacing iteration dependent predicates of operations *inside* a given index space by predicates on locally propagated control signals.

The presented approach can easily be extended for a more general class of index spaces, so-called *linearly bounded lattices*:

$$\mathcal{I} \;=\; \{I \in \mathbb{Z}^n \mid I = M\kappa + c \;\wedge\; A\kappa \geq b\}$$

where $\kappa \in \mathbb{Z}^l$, $M \in \mathbb{Z}^{n \times l}$, $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{m \times l}$ and $b \in \mathbb{Z}^m$. Like throughout this paper, $\{\kappa \in \mathbb{Z}^l \mid A\kappa \geq b\}$ defines an integral convex polyhedron or in case of boundedness a polytope in $\mathbb{Z}^l$. This set is affinely mapped onto iteration vectors I using an affine transformation ($I = M\kappa + c$).

Our new distributed loop control methodology is integrated as part of the PARO design system and can be used during the process of automated synthesis of regular circuits. PARO is a design system project for modeling, transforming, optimization, and processor synthesis for the class of piecewise linear algorithms. For further information on the PARO design system project, check the following website: `http://www-date.upb.de/research/paro/`.

References

1. Atmel Inc. *AT6000 FPGA Configuration Guide.*
2. Marcus Bednara and Jürgen Teich. Synthesis of FPGA Implementations from Loop Algorithms. In *Proc. of the First International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA01)*, pages 1–7, Las Vegas, Nevada, June 2001.
3. Philippe Clauss and Vincent Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *Journal of VLSI Signal Processing*, 19(2):179–194, July 1998.
4. Alain Darte, Leonid Khachiyan, and Yves Robert. Linear Scheduling is Nearly Optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.
5. Eugène Ehrhart. *Polynômes arithmétiques et Méthode des Polyèdres en Combinatoire*, volume 35 of *International Series of Numerical Mathematics*. Birkhäuser Verlag, Basel, 1. edition, 1977.
6. Frank Hannig and Jürgen Teich. Design Space Exploration for Massively Parallel Processor Arrays. In Victor Malyshkin, editor, *Parallel Computing Technologies, 6th International Conference, PaCT 2001, Proceedings*, volume 2127 of *Lecture Notes in Computer Science (LNCS)*, pages 51–65, Novosibirsk, Russia, September 2001. Springer.

7. Richard M. Karp, Raymond E. Miller, and Shmuel Winograd. The Organization of Computations for Uniform Recurrence Equations. *Journal of the Association for Computing Machinery*, 14(3):563–590, 1967.

8. Robert H. Kuhn. Transforming Algorithms for Single-Stage and VLSI Architectures. In *Workshop on Interconnection Networks for Parallel and Distributed Processing*, pages 11–19, West Layfaette, Indiana, April 1980.

9. Sun Yuan Kung. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, New Jersey, 1987.

10. Christian Lengauer. Loop Parallelization in the Polytope Model. In Eike Best, editor, *CONCUR'93*, Lecture Notes in Computer Science 715, pages 398–416. Springer-Verlag, 1993.

11. Dan I. Moldovan. On the Design of Algorithms for VLSI Systolic Arrays. In *Proceedings of the IEEE*, volume 71, pages 113–120, January 1983.

12. Alexander Schrijver. *Theory of Linear and Integer Programming*. Whily – Interscience series in discrete mathematics. John Wiley & Sons, Chichester, New York, 1986.

13. Jürgen Teich. *A Compiler for Application-Specific Processor Arrays*. Shaker (Reihe Elektrotechnik). Zugl. Saarbrücken, Univ. Diss, ISBN 3-86111-701-0, Aachen, Germany, 1993.

14. Jürgen Teich and Lothar Thiele. Control Generation in the Design of Processor Arrays. *Int. Journal on VLSI and Signal Processing*, 3(2):77–92, 1991.

15. Jürgen Teich and Lothar Thiele. Control Generation in the Design of Processor Arrays. In Josef A. Nossek, editor, *Parallel Processing on VLSI Arrays*. Kluwer Academic Publishers, 1992.

16. Jürgen Teich and Lothar Thiele. Partitioning of Processor Arrays: A Piecewise Regular Approach. *INTEGRATION: The VLSI Journal*, 14(3):297–332, 1993.

17. Lothar Thiele. On the Design of Piecewise Regular Processor Arrays. In *Proc. IEEE Symp. on Circuits and Systems*, pages 2239–2242, Portland, 1989.

18. Lothar Thiele. Scheduling of Uniform Algorithms with Resource Constraints. *Journal of VLSI Signal Processing*, 10:295–310, 1995.

19. Michael Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, Redwood City, California, 1996.

20. Xilinx Inc. `http://www.xilinx.com/partinfo/ds003-2.pdf`.

21. Jingling Xue. *The Formal Synthesis of Control Signals for Systolic Arrays*. PhD thesis, Univ. of Edinburgh, March 1992.