

# On Control Signals for Multi-Dimensional Time<sup>\*</sup>

DaeGon Kim<sup>1</sup>, Gautam<sup>1,2</sup>, and S. Rajopadhye<sup>1</sup>

<sup>1</sup> Colorado State University, Fort Collins, CO, U.S.A.

{kim,gautam}@cs.colostate.edu, Sanjay.Rajopadhye@colostate.edu

<sup>2</sup> IRISA, Rennes, France

**Abstract.** Affine control loops (ACLs) comprise an important class of compute- and data-intensive computations. The theoretical framework for the automatic parallelization of ACLs is well established. However, the hardware compilation of arbitrary ACLs is still in its infancy. An important component for an efficient hardware implementation is a control mechanism that informs each processing element (PE) which computation needs to be performed and when.

We formulate this *control signal problem* in the context of compiling arbitrary ACLs parallelized with a multi-dimensional schedule into hardware. We characterize the logical time instants when PEs need a control signal indicating which particular computations need to be performed. Finally, we present an algorithm to compute the minimal set of logical time instants for these control signals.

## 1 Introduction

It is well known that loops comprise the compute-intensive kernels of many applications. An important class of statically-analyzable loops is affine control loops (ACLs). Bastoul et. al. [1] report that over 99% in 7 out of 12 programs of the widely studied benchmark suites SpecFP and PerfectClub are affine control loops. Due to their high computational complexity, considerable research has aimed at deriving application specific circuits directly and automatically from ACLs and incorporated in both academic and commercial tools [2,3]. However, these tools have been restricted to a proper subset of ACLs, specifically, those that can be systematically transformed to loops that possess a 1-dimensional *affine schedule* and with uniform dependences<sup>3</sup> (or slight generalizations). One-dimensional affine schedules imply that only one loop carries dependences.

This paper deals with the full generality of arbitrary ACLs. The ACL may be arbitrarily nested, have arbitrary affine dependences and may have been parallelized with a multidimensional schedule [4]. Multidimensional schedules assign a time vector to computations which is interpreted as the logical instant at which they are executed. These time vectors are the iteration vectors corresponding

---

<sup>\*</sup> This research was supported in part, by the National Science Foundation, under the grant EI-030614: HiPHiPECS: High Level Programing of High Performance Embedded Computing Systems

<sup>3</sup> These conditions were motivated by the need to derive systolic architectures.

to outer sequential (time) loops in a loop nest. The total order between logical time instants is lexicographic. Multidimensional schedules present the following advantages over linear schedules (i) Some applications do not admit a linear schedule [5], and (ii) The architecture resulting from the parallelism exposed by linear schedules might not fit into the available hardware resources. With multidimensional schedules, we may limit the degree of parallelism.

The chosen parallelization is fine-grained such that the inner loops are parallel. The resulting loop nest is called an `FGP-ACL`. Parallel computations are mapped to distinct simple processing elements (`PEs`). Each `PE` may either be active or inactive for a certain logical time-vector. When it is active, it executes the assigned computation. The set of all valid time-vectors is called the *global time domain* of the specification and the set of all active time-vectors of a processor is called its *local time domain*.

The context of this work is the generation of custom hardware that realizes `PEs`, registers and memory banks local to each `PE` and an interconnection for data transfer across `PEs`. The target implementation for this custom hardware may either be an `ASIC` or an `FPGA`. Previous work [6] in this direction presents a methodology in which each `PE` implements a multi-dimensional counter to scan the global time domain, *i.e.*, an automaton enumerating valid time-vectors together with a test for the membership of time-vectors in the local time domain. The resource overhead for this scheme is significant.

Avoiding such an automaton involves (i) providing the control to every `PE` instructing it to resume or suspend its activity (start and stop are treated as special cases) and information about the statements it needs to compute at any particular time instant, and (ii) the computation of the array addresses for statements in the body of the loop nest. The related problem of developing an interconnection so that each `PE` can access the memory where data is stored has been addressed elsewhere [7].

This paper deals with the problem of providing the suspend and resume signals to each `PE` and the statements that need to be executed. In the presentation of this paper, we will specialize suspend and resume with respect to particular statements such that both these problems are solved together. We will refer to this as the *control signal problem*. The key contributions of this paper are (i) a precise formulation of control signal problem, (ii) characterization of time when control signals are necessary, and (iii) an algorithm computing an exact solution for the control signal problem.

The rest of this paper is organized as follows. The following section illustrates the control signal problem with an example. We introduce some concepts and notations for fine-grained parallel `ACLs` in Section 3. Section 4 precisely formulates the control signal problem, characterizes the time when `PEs` need to receive a control signal, and presents an algorithm to compute an exact solution for the control signal problem. Section 5 is a discussion on the propagation of the signals. Section 6 discusses related work, and finally we present our conclusions and future work in Section 7.

## 2 Illustrating Example

*Example 1.* Consider the following program.

```

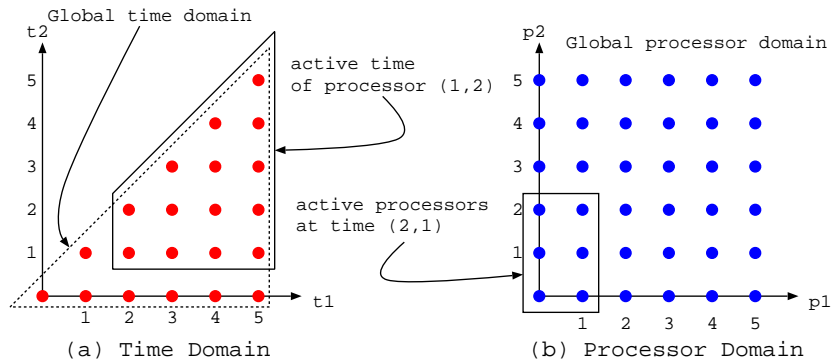
for t1 = 0 to n
  for t2 = 0 to t1
    forall p1 = 0 to t2
      forall p2 = 0 to t1
        ...

```

The first two loops are executed sequentially, and the remaining loops are parallel loops. We will interpret this program as follows: the two sequential loops represent time, and the parallel loops denote processors. For example, the processor  $(0,0)$  and  $(0,1)$  are active, i.e. execute statement, at time  $(1,0)$ . Given an iteration of the sequential loops, all the iterations of the parallel loops are executed at the same time. For instance, when  $t1 = n$  and  $t2 = n$ , all the  $n \times n$  processors are active.

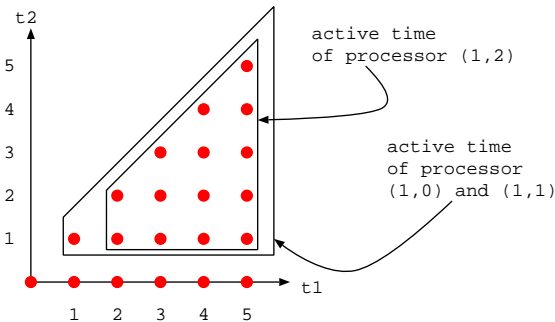
Now, we would like to address the question of which processors become active and idle at a given time. For example, the processor  $(0,0)$  resumes at time  $(0,0)$ , and both the processor  $(1,0)$  and  $(1,1)$  resume at time  $(1,1)$ .

In this program, the set of valid time coordinates for the entire program is called the *global time domain*. It is the iteration space of the time loops. Similarly, the set of all processor coordinates is called the *global processor domain*. It is the set of all the possible iterations over the global time domain. Figure 1 shows the global time and processor domain of this example for  $n = 5$ . Every processor is, however, not active all the time, i.e. for all the points in the global time domain. Each processor  $p$  has its own active time. Similarly, the set of active processors varies over the time. Figure 1 also illustrates the time when the processor  $(1,2)$  is active and the active processors at time  $(2,1)$ .



**Fig. 1.** Time and processor domain

It is easy to see when the processor (1, 2) should receive a resume (or suspend) signal from Figure 1(a). For each  $i$  such that  $2 \leq i \leq 5$ , it will become active at time  $(i, 1)$  and idle after  $(i, i)$ . However, it is not obvious which processors should receive resume (or suspend) signals at time  $(2, 1)$  from Figure 1(b). For instance, the processor (0, 0) need not receive a resume signal, because it was already active at the previous time step  $(2, 0)$ . Among the six processors in Figure 1(b), only three processors (1, 0), (1, 1) and (1, 2) have to receive a resume signal. As shown in Figure 2, the set of time coordinates when processor  $(p1, p2)$  is active is the intersection of the global time domain and  $\{t1, t2 \mid p1 \leq t2; p2 \leq t1\}$ . For instance, the earliest active time of processor (1, 2) is (2, 1). For each processor whose first coordinate is 0, it is always active at  $(t1, 0)$  if the processor is active at  $(t1, 1)$ . So, the other three processors in Figure 1 does not need to receive a resume signal at  $(2, 1)$  because these processors was active at the previous time step  $(2, 0)$ .



**Fig. 2.** Active time of processors (1, 0), (1, 1) and (1, 2) in the global time domain

### 3 Problem Formulation

The following transformations are involved in the hardware compilation of ACLs. Value based dependence analysis (or exact dataflow analysis) [8,9] on ACLs transforms it to a system of recurrence equations defined over polyhedral domains. On these equations, scheduling analysis [10,11,5,12,13,14,15,16] is performed which assigns an (multidimensional) execution time to each computation so that dependences are satisfied. The schedule may be such that multiple computations are executed at the same time step. Thus, there is inherent parallelism. An intermediate code generation step generates *fine-grained parallel* (FGP) ACLs. In FGP-ACLs, the outer loops are sequential (representing multidimensional time) and the inner loops are parallel. The array access functions in the statements within the body of the loop nest are in terms of the time and processor loop indices and size parameters. Hardware compilation is the mapping of parallel computations to distinct processing elements.

We will first present ACLs and the required mathematical background of integer polyhedra. Then we will discuss FGP-ACLs and define the terminology needed for our analysis.

### 3.1 ACLs (Affine Control Loops)

ACLs are loop nests of the form defined in Figure 3. Note that **Lower** (respectively, **Upper**) is the maximum (respectively, minimum) of a *finite number of* affine functions, respectively. Kuck [17] showed that the set of iterations with such bounds can be described as (a union of) integer polyhedra. Precisely, an iteration space of ACLs is a union of integral polyhedra parameterized by the size parameters  $s$ .

ACL Grammar	
ACL	: Stmt *
Stmt	: Loop   Assignment
Loop	: for Index=Lower ... Upper ACL
Assignment	: ArrayRef = Expression;

**Fig. 3.** ACL grammar. **Index** is an identifier; **Lower** (respectively, **Upper**) is the maximum (respectively, minimum) of affine functions of outer indices and size parameters; **ArrayRef** is a reference to a multidimensional array whose access expression is an affine function of an iteration vector and the size parameters; and **Expression** is an arbitrary expression comprising finitely many **ArrayRefs** with a strict, side-effect free function.

**Definition 1.** An *integer polyhedron*  $\mathcal{P}$  is a set of integer solutions of finitely many affine inequalities with integer coefficients. These inequalities are called *constraints of  $\mathcal{P}$* . A *parameterized integer polyhedron* is an integer polyhedron some of whose indices are interpreted as symbolic constants.

The iteration space of ACLs can be written in the following form:

$$\left\{ \begin{pmatrix} z \\ s \end{pmatrix} \mid (M \ M_s) \begin{pmatrix} z \\ s \end{pmatrix} \geq c, \begin{pmatrix} z \\ s \end{pmatrix} \in \mathbb{Z}^{n_z+n_s} \right\} \quad (1)$$

where  $n_z$  is the depth of the loop nest and  $n_s$  is the number of size parameters.

### 3.2 Fine grained parallel ACLs

In the fine grained parallel ACLs each statement  $\mathcal{S}$  is surrounded by  $k_{\mathcal{S}}$  sequential loops and  $d_{\mathcal{S}}$  parallel loops. Our analysis is per each statement and we will drop the subscript  $\mathcal{S}$ . An iteration vector of sequential loops represents time, and the order of these  $k$ -vectors is defined by the lexicographic order.

**Definition 2.** Let  $u = (u_1, \dots, u_n)^T$  and  $v = (v_1, \dots, v_n)^T$  be  $n$ -dimensional vectors. The **lexicographic order**  $\prec$  is defined by the following rule:

$u \prec v$  if there exists  $i$  such that  $u_j = v_j$  for all  $1 \leq j < i$ ,  $u_i < v_i$ , and  $1 \leq i \leq n$ .

The remaining  $d$  innermost loops represents processing elements. We distinguish time and processor indices, and throughout the paper, time and processor indices are denoted by  $t$  and  $p$ , respectively. So, the iteration space of parallel ACLs can be written as follows:

$$\left\{ \begin{pmatrix} t \\ p \\ s \end{pmatrix} \mid (M_t \ M_p \ M_s) \begin{pmatrix} t \\ p \\ s \end{pmatrix} \geq c, \begin{pmatrix} t \\ p \\ s \end{pmatrix} \in \mathbb{Z}^{k+d+n_s} \right\} \quad (2)$$

### 3.3 Time and processor domain

Here, we introduce some concepts related to fine grained parallel ACLs.

**Definition 3.** The **global time domain**  $\mathcal{GT}$  is the set of all the sequential loop iterations. The **local time domain**  $\mathcal{LT}(p)$  is the set of the sequential loop iterations at which a given processor  $p$  is active.

**Definition 4.** A **local processor domain**  $\mathcal{LP}(t)$  at a given  $t \in \mathcal{GT}$  is the set of active processors at time instant  $t$ . The **global processor domain**  $\mathcal{GP}$  is the set of  $\bigcup_t \mathcal{LP}(t)$  for all  $t \in \mathcal{GT}$ .

In fact, these concepts have already been introduced in section 2 with our example. The global time and processor domain of Example 1 is  $\{(t1, t2) \mid 0 \leq t2 \leq t1 \leq n\}$  and  $\{(p1, p2) \mid 0 \leq p1 \leq n; 0 \leq p2 \leq n\}$ , respectively.  $\mathcal{GT}$  can be computed directly from the sequential loops, and  $\mathcal{GP}$  can be derived by projecting the ACL iteration space onto processor space, i.e. elimination of time indices. The local processor domain at time instant  $(t1, t2)$  is  $\{(p1, p2) \mid 0 \leq p1 \leq t2; 0 \leq p2 \leq t1\}$ . This can be regarded as a polyhedron parameterized by the time  $t$  and size parameters  $s$ . In fact,  $\mathcal{LP}(2, 1)$  in Figure 1 is computed by replacing  $t1$  with 2 and  $t2$  with 1. Similarly, the local time domain can be seen as a polyhedron parameterized by the processor  $p$  and  $s$ . So,  $\mathcal{LT}(p)$  of Example 1 is  $\{(t1, t2) \mid 0 \leq p1 \leq t2 \leq n; 0 \leq p2 \leq t1 \leq n\}$ .

*Property 1.*  $\mathcal{LT}(p)$  and  $\mathcal{LP}(t)$  are polyhedra parameterized by  $p$  and  $t$ , respectively. More precisely,  $\mathcal{LT}(p)$  is a polyhedron in  $k$ -dimensional space parameterized by  $p$  and  $s$ , and  $\mathcal{LP}(t)$  is a polyhedron in  $d$ -dimensional space parameterized by  $t$  and  $s$ .

## 4 Analysis

Now, we want to characterize the set of processors that receive a *resume* and *suspend* signal, as well as the time instants expressed as a function of their coordinates, at which this happens.

**Definition 5.** The *immediate precedent*, denoted by  $pre(t)$ , of  $t \in \mathcal{GT}$  is an element  $t'$  of  $\mathcal{GT}$  satisfying that (i)  $t' \prec t$ , and (ii) there is no element  $t'' \in \mathcal{GT}$  such that  $t' \prec t'' \prec t$ . The *immediate successor* of  $t \in \mathcal{GT}$ , denoted by  $next(t)$ , is  $t'$  such that  $t = pre(t')$ .

Note that  $pre(t)$  and  $next(t)$  are defined only in the context of  $\mathcal{GT}$ , not  $\mathcal{LT}$ .

*Property 2.* A processor  $p$  must receive a resume signal at time instant  $t \in \mathcal{GT}$  if either (i)  $t \in \mathcal{LT}(p)$  and  $t$  is the lexicographic minimum of  $\mathcal{GT}$ , or (ii)  $t \in \mathcal{LT}(p)$  and  $pre(t) \notin \mathcal{LT}(p)$ . Similarly, processor  $p$  must receive a suspend signal at time  $t \in \mathcal{GT}$  if either (i)  $t \in \mathcal{LT}(p)$  and  $t$  is the lexicographic maximum of  $\mathcal{GT}$ , or (ii)  $t \in \mathcal{LT}(p)$  and  $next(t) \notin \mathcal{LT}(p)$ .

The above property conceptually gives a precise time when a signal arrives at a certain processor. We now characterize this set of time instants.

**Definition 6.** A *face* of a polyhedron  $\mathcal{P}$  is an intersection of  $\mathcal{P}$  with a subset of inverses of its constraints. A face  $\mathcal{F}$  is called a **facet** if there is no face  $\mathcal{F}'$  such that  $\mathcal{F} \subset \mathcal{F}' \subset \mathcal{P}$ .

Note that a facet of an integral polyhedron may be “thick” in the sense that a single hyperplane cannot contain it. Here, we distinguish two kinds of facets.

**Definition 7.** Let  $\mathcal{P}$  be a polyhedron in  $n$ -dimensional space. Let  $e_n$  be the  $n$ -th unit vector, i.e., an  $n$ -vector whose last element is 1 and the other elements are 0. A facet  $\mathcal{F}$  is a **lower facet** if  $p - e_n \notin \mathcal{P}$  for all  $p \in \mathcal{F}$ . Similarly, a facet  $\mathcal{F}$  is an **upper facet** if  $p + e_n \notin \mathcal{P}$  for all  $p \in \mathcal{F}$ . Let  $\mathbb{LF}(P)$  and  $\mathbb{UF}(P)$  denote the set of all the lower and upper facets of  $P$ , respectively.

We illustrate these concepts with the following example.

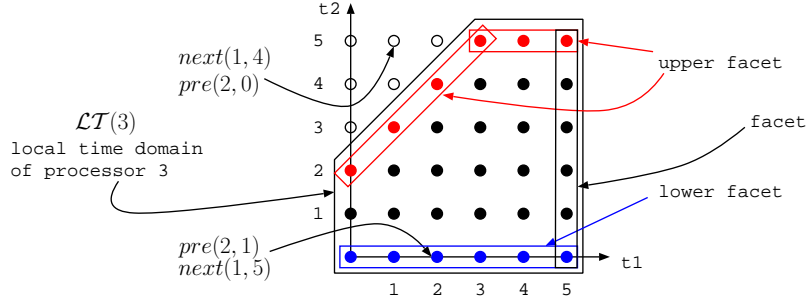
*Example 2.* Consider the following parallel ACL.

```

for t1 = 0 to n
  for t2 = 0 to n
    forall p1 = 0 to t1+n-t2
      ...

```

The  $\mathcal{GT}$  and  $\mathcal{LT}(3)$  of this example are given in Figure 4 for  $n = 5$ . This figure also illustrates facets of the local time domain with one lower facet and two upper facets. For each point  $t \notin \mathbb{LF}(\mathcal{GT})$ , its immediate precedent is simply  $t - (0, 1)$ . Similarly, for  $t \notin \mathbb{UF}(\mathcal{GT})$ , its immediate successor is  $t + (0, 1)$ . However,



**Fig. 4.** Global and local time domain of Example 2 for  $n = 5$ ; one lower facet and two upper facet of the local time domain

$pre(t)$  of  $t \in \mathbb{LF}(\mathcal{GT})$  is found in some element of  $\mathbb{UF}(\mathcal{GT})$ . Similarly,  $next(t)$  of  $t \in \mathbb{UF}(\mathcal{GT})$  is found in some element of  $\mathbb{LF}(\mathcal{GT})$ .

Note that a processor receives a signal at a time instant which belongs to its local time domain. The following lemma gives the first characterization for the time when a signal arrives at a processor.

**Lemma 1.** *If a processor  $p$  receives a resume (resp. suspend) signal at time  $t$ ,  $t$  is in a lower (resp. upper) facet of  $\mathcal{LT}(p)$ .*

*Proof.* Let  $e_k \in \mathbb{Z}^k$  be  $(0, \dots, 0, 1)^T$ . Note that  $t \in \mathcal{LT}(p)$  and  $pre(t) \notin \mathcal{LT}(p)$ . So,  $t - e_k \notin \mathcal{LT}(p)$ . Therefore,  $t$  belongs to a lower facet of  $\mathcal{LT}(p)$ .

The proof for a suspend signal is similar.

This lemma says that a processor  $p$  receives a signal only if time  $t$  belongs to a lower and upper facet of  $\mathcal{LT}(p)$ . In other words, if  $t \in \mathcal{GT}$  is not in lower and upper facets of  $\mathcal{LT}(p)$ , the processor  $p$  does not need to receive any signals. Hence, a safe solution is to generate a signal for *each point* in the lower and upper facets of  $\mathcal{LT}(p)$ .

*Solution 1.* A **naive** way to control a processor  $p$  is to generate resume signals at all points in  $\mathbb{LF}(\mathcal{LT}(p))$  and suspend signals over  $\mathbb{UF}(\mathcal{LT}(p))$ .

Sometimes, this naive solution might be an exact solution. For instance, processor (1, 2) in Figure 1 must receive signals for all the points in the lower and upper facets of its local time domain. Furthermore, in Example 1, this naive solution becomes an exact solution for all the processors except the five processors, (0, 0), (0, 1), (0, 2), (0, 3) and (0, 4). While processor (1, 2) receives the resume and suspend signals over the lower and upper facet of  $\mathcal{LT}(1, 2)$  respectively, processor (0, 0) needs to receive only two signals, a resume signal at (0, 0) and a suspend signal at (5, 5), because  $\mathcal{LT}(0, 0)$  is equal to  $\mathcal{GT}$ . With the fact that the number of suspend signals have to be the number of resume signals, the following theorem explains why the naive solution becomes an exact solution for almost all processors in Example 1.

**Theorem 1.** *Let  $p \in \mathcal{GP}$ .*

(a) *If  $t \in \mathcal{GT}$  is in a lower facet of  $\mathcal{LT}(p)$  and not in a lower facet of  $\mathcal{GT}$ , the processor  $p$  must receive a resume signal at time  $t$ .*

(b) *If  $t \in \mathcal{GT}$  is in an upper facet of  $\mathcal{LT}(p)$  and not in an upper facet of  $\mathcal{GT}$ , the processor  $p$  must receive a suspend signal at time  $t$ .*

*Proof.* (a) Suppose  $t$  is in a lower facet of  $\mathcal{LT}(p)$  and not in a lower facet of  $\mathcal{GT}$ . Let  $e_k \in \mathbb{Z}^k$  be  $(0, \dots, 0, 1)^T$ . Note that  $\mathcal{GT}$  is a polyhedron in  $k$ -dimensional space. Since  $t$  is not in a lower facet of  $\mathcal{GT}$ ,  $t - e_k \in \mathcal{GT}$ . On the other hand,  $t - e_k \notin \mathcal{LT}(p)$ , because  $t$  is in a lower facet of  $\mathcal{LT}(p)$ . Since  $t - e_k$  is  $pre(t)$ , processor  $p$  must receive a resume signal at time  $t$ .

The proof for (b) is symmetric.

Let us apply this theorem to Example 1. For a processor whose first coordinate is greater than 0, the lower facet of its local time domain shares no point with the lower facet of  $\mathcal{GT}$ . So, for each point in the lower facet, the processor must receive a resume signal. Also, there is a corresponding suspend signal for each resume signals. Also, note that the number of points in the lower facets is equal to that of points in the upper facets. By Lemma 1, each point in the upper facet must also receive a resume signal.

Theorem 1 proves that every point in  $\mathbb{LF}(\mathcal{LT}) \setminus \mathbb{LF}(\mathcal{GT})$  and  $\mathbb{UF}(\mathcal{LT}) \setminus \mathbb{UF}(\mathcal{GT})$  needs a signal. Now, we focus on the remaining part,  $\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT})$  and  $\mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT})$ . A processor need not receive a resume signal at time  $t$  when  $pre(t) \in \mathcal{LT}(p)$ . For instance, the processor  $(0, j)$  for  $0 \leq j \leq 5$  in Example 1 needs only one resume signal because, for each point  $t \in \mathbb{LF}(\mathcal{LT})$  but one,  $pre(t)$  belongs to its local time domain. When a time instant  $t$  is in a lower facet of  $\mathcal{GT}$ ,  $pre(t)$  is in a upper facet of  $\mathcal{GT}$ . So, the precise answer for the question of what point need a resume signal can be obtained from (i)  $\mathbb{LF}(\mathcal{LT}(p)) \cap \mathbb{LF}(\mathcal{GT})$  and (ii)  $\mathbb{UF}(\mathcal{LT}(p)) \cap \mathbb{UF}(\mathcal{GT})$ . Algorithm 1 illustrates how the precise solution can be obtained by comparing these two sets for the 2-dimensional control signal problem. The operations in the algorithm are well supported [18,19].

Now, consider example 2 with Figure 4. Its  $\mathcal{GT}$  and  $\mathcal{LT}$  are  $\{t1, t2 \mid 0 \leq t1 \leq n; 0 \leq t2 \leq n\}$  and  $\{t1, t2 \mid 0 \leq t1 \leq n; 0 \leq t2 \leq n; t2 \leq t1 + n - p1\}$ , respectively. Here, we compute only the set of points that require a suspend signal, and that of a resume signal will be left as an exercise. First, there are two upper facets of  $\mathcal{LT}$ ,  $\{t1, t2 \mid 0 \leq t1 \leq p1 - 1; t2 = t1 + n - p1\}$  and  $\{t1, t2 \mid p1 \leq t1 \leq n; t2 = n\}$ . Note that either of them can be empty set and the point  $(p1, n)$  is removed from the first set. In fact, the first set is  $\mathbb{UF}(\mathcal{LT}) \setminus \mathbb{UF}(\mathcal{GT})$ , and the second is  $\mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT})$ . Hence, a suspend signal is needed at every point in the first set. To compute the exact points in the second set,  $\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT})$  is required, and it is  $\{t1, t2 \mid 0 \leq t1 \leq n; n - p1 \leq t1; t2 = 0\}$ . Adding a ray  $(1, 0)$  yields  $\{t1, t2 \mid 0 \leq t1 \leq n; n - p1 \leq t1; t2 > 0\}$ , and shifting it by  $(-1, 0)$  yields  $\{t1, t2 \mid -1 \leq t1 \leq n - 1; n - p1 - 1 \leq t1; t2 > 0\}$ . From subtracting this from  $\mathbb{UF}(\mathcal{LT}) \setminus \mathbb{UF}(\mathcal{GT})$ , we get  $\{t1, t2 \mid 0 \leq t1 \leq p1 - 1; t2 = t1 + n - p1\}$  and  $\{t1, t2 \mid t1 = n; t2 = n\}$  if this point  $(n, n)$  belongs to  $\mathcal{LT}$ . The main intuition is that  $pre(t)$  of  $t \in \mathbb{UF}(\mathcal{GT})$  lies in  $\mathbb{LF}(\mathcal{GT})$ , and consequently,  $next(t)$  of  $t \in \mathbb{LF}(\mathcal{GT})$  lies in  $\mathbb{UF}(\mathcal{GT})$ .

---

**Algorithm 1** Algorithm for 2-dimensional time

---

INPUT :  $\mathcal{LT}$  - local time domain,  $\mathcal{GT}$  - global time domainOUTPUT :  $\mathcal{R}$  - the set of points requiring resume signals,  $\mathcal{S}$  - the set of points requiring suspend signals1: compute  $\mathbb{LF}(\mathcal{LT})$ ,  $\mathbb{UF}(\mathcal{LT})$ ,  $\mathbb{LF}(\mathcal{GT})$  and  $\mathbb{UF}(\mathcal{GT})$ 2:  $\mathcal{R} \leftarrow \mathbb{LF}(\mathcal{LT}) \setminus \mathbb{LF}(\mathcal{GT})$ ;  $\mathcal{S} \leftarrow \mathbb{UF}(\mathcal{LT}) \setminus \mathbb{UF}(\mathcal{GT})$ 3: compute  $\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT})$  and  $\mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT})$ 4: add a ray  $(0, -1)$  to  $\mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT})$ , shift it by  $(1, 0)$ , and subtract it from  $\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT})$ 5: add a ray  $(0, 1)$  to  $\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT})$ , shift it by  $(-1, 0)$ , and subtract it from  $\mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT})$ 6: add the result of step 4 to  $\mathcal{R}$ ; add the result of step 5 to  $\mathcal{S}$ 7: return  $\mathcal{R}$  and  $\mathcal{S}$ 

---

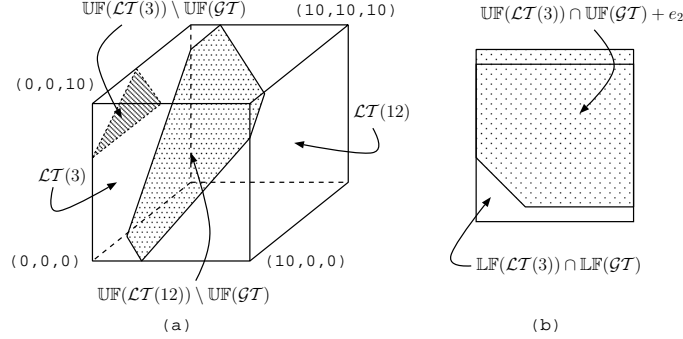
We now introduce another example that has non-trivial behavior, and will help understand the algorithm for the general case.

*Example 3.* Consider the following simple extension of Example 2 with a third time loop.

```
for t1 = 0 to n
  for t2 = 0 to n
    for t3 = 0 to n
      forall p1 = 0 to t1+t2+n-t3
        ...
```

Figure 5(a) shows the  $\mathcal{GT}$  of this example for  $n = 10$ , as well as  $\mathcal{LT}(3)$  and  $\mathcal{LT}(12)$ . First, consider the times when processor 3 needs a resume signal. As shown in Figure 5(a),  $\mathbb{LF}(\mathcal{LT}(3)) \cap \mathbb{LF}(\mathcal{GT})$  is the entire bottom square  $\{t1, t2, t3 \mid t3 = 0; 0 \leq t1, t2 \leq 10\}$ . Figure 5(b) visualizes the 3D generalization of step 5 in the algorithm 1. For every point  $t$  in  $\mathbb{LF}(\mathcal{LT}(3)) \cap \mathbb{LF}(\mathcal{GT})$  except its lower facet,  $pre(t)$  is “directly above”  $t$  after shifting. However,  $pre(t)$  of the point  $t$  on the line segment between  $(0, 0, 0)$  and  $(10, 0, 0)$  is on the line segment between  $(0, 0, 0)$  and  $(10, 0, 0)$ . The first line segment can be seen as a lower facet of a lower facet of  $\mathcal{GT}$  (we call this the second lower facet of  $\mathcal{GT}$ ). Similarly, the line segment between  $(0, 0, 0)$  and  $(10, 0, 0)$  is the second upper facet of  $\mathcal{GT}$ . This explains why the algorithm for general case recurses on the dimensions of the polyhedra it manipulates. So, processor 3 needs a resume signal at the following set  $\{t1, t2, t3 \mid t3 = 0; 0 \leq t1; 1 \leq t2; t1 + t2 \leq 4\} \cup \{(0, 0, 0)\}$ .

Now, consider the processor 12. The intersection of  $\mathbb{LF}(\mathcal{LT}(12))$  and  $\mathbb{LF}(\mathcal{GT})$  is  $\mathbb{LF}(\mathcal{LT}(12))$  itself,  $\{t1, t2, t3 \mid 0 \leq t1, t2 \leq 10; t1 + t2 \geq 2; t3 = 0\}$ . The intersection of  $\mathbb{UF}(\mathcal{LT}(12))$  and  $\mathbb{UF}(\mathcal{GT})$  is  $\{t1, t2, t3 \mid t1 \leq 10; t2 \leq 10; t1 + t2 \geq 12; t3 = 10\}$ . Step 5 in algorithm 1 tells us that a resume signal is needed in  $\{t1, t2, t3 \mid 0 \leq t1 \leq 10; 1 \leq t2 \leq 10; t1 + t2 \geq 2; t1 + t2 \leq 12; t3 = 0\}$ . We emphasize that *only the intersection of the second lower facets of  $\mathcal{LT}$  and  $\mathcal{GT}$  is compared with only the intersection of the second upper facets of  $\mathcal{LT}$*



**Fig. 5.** (a) Global time domain of Example 3;  $\mathcal{LT}(3)$  contains  $\mathcal{LT}(11)$  (b) and  $\mathbb{LF}(\mathcal{LT}(3)) \cap \mathbb{LF}(\mathcal{GT})$  and shifted  $\mathbb{UF}(\mathcal{LT}(3)) \cap \mathbb{UF}(\mathcal{GT})$  by  $(0, 1, 0)$

and  $\mathcal{GT}$  although there are the two second lower facets of  $\mathcal{LT}$ . The final result is  $\{t_1, t_2, t_3 \mid 0 \leq t_1 \leq 10; 1 \leq t_2 \leq 10; t_1 + t_2 \geq 2; t_1 + t_2 \leq 12; t_3 = 0\} \cup \{(2, 0, 0)\}$ .

The general case is addressed in algorithm 2. The difference from algorithm 1 is a recursive function for computing the exact points in  $\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT})$  and  $\mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT})$ . However, depending on which face of  $\mathcal{GT}$  that  $t$  belongs to,  $pre(t)$  or  $next(t)$  is in a different face of  $\mathcal{GT}$ .

**Theorem 2.** Algorithm 2 is (i) correct and (ii) exact:

(i) *Correctness:* If a processor  $p$  needs a resume (resp. suspend) signal at  $t \in \mathcal{GT}$ , then  $t$  belongs to the set  $R$  (resp.  $S$ ).

(ii) *Exactness:* If  $t \in \mathcal{GT}$  belongs to the set  $R$  (resp.  $S$ ), then a processor  $p$  needs a resume (resp. suspend) signal at time  $t$ .

*Proof.* (i) *Correctness:* Suppose that a processor  $p$  needs a resume signal at time  $t \in \mathcal{GT}$ . Then,  $t \in \mathcal{LT}(p)$  and  $pre(t) \notin \mathcal{LT}(p)$ . By lemma 1,  $t \in \mathbb{LF}(\mathcal{LT})$ . If  $t \notin \mathbb{LF}(\mathcal{GT})$ , then  $t \in \mathbb{LF}(\mathcal{LT}) \setminus \mathbb{LF}(\mathcal{GT})$ . In this case,  $t$  is taken into  $R$  by step 3. Otherwise,  $t \in \mathbb{LF}(\mathcal{GT})$ . First, consider the case that  $t$  is not the lexicographic minimum of  $\mathcal{GT}$ . Then,  $pre(t)$  exists, and let  $t = (t_1, \dots, t_k)$  and  $pre(t) = (t'_1, \dots, t'_k)$ . By the definition of  $pre(t)$ , there exists an integer  $f \geq 1$  such that (i)  $t_i = t'_i$  for all  $1 \leq i \leq f - 1$ , (ii)  $t_f - 1 = t'_f$ , and (iii)  $t' + e_i \notin \mathcal{GT}$  for all  $f < i \leq k$ . Now, consider when  $t$  is determined to be taken or not, i.e. the face that  $t$  belongs to. By the condition (iii), it belongs to  $(k - f)$ -th upper facet of  $\mathcal{GT}$ . That is, the  $f$ -th recursive call determines whether  $t$  is taken or not. By  $pre(t) \notin \mathcal{LT}$  and the fact that  $pre(t) + e_f$  and  $t$  have the common elements up to  $f$ -th element,  $t$  is taken into  $R$ . This is the only operation that affects the determination of  $t$ . Finally, consider the case when  $t$  is the lexicographic minimum of  $\mathcal{GT}$ . In this case,  $t$  belongs to the last lower facet of  $\mathcal{GT}$ . This is the base case. So  $t$  is taken into  $R$ . Therefore, Algorithm 2 is correct.

(ii) *Exactness:* Suppose that algorithm 2 picks time  $t \in \mathcal{GT}$  into  $R$ . By the construction of  $R$ , it is easy to see that every element of  $R$  belongs to  $\mathbb{LF}(\mathcal{LT})$ . Now, let  $t \in R$  be  $(t_1, t_2, \dots, t_k)$ .

---

**Algorithm 2** Algorithm for  $k$ -dimensional time domain (general case)

---

INPUT :  $\mathcal{LT}$  - local time domain,  $\mathcal{GT}$  - global time domainOUTPUT :  $\mathcal{R}$  - the set requiring resume signals,  $\mathcal{S}$  - the set requiring suspend signals1: compute  $\mathbb{LF}(\mathcal{LT})$ ,  $\mathbb{UF}(\mathcal{LT})$ ,  $\mathbb{LF}(\mathcal{GT})$  and  $\mathbb{UF}(\mathcal{GT})$ 2:  $\mathcal{R} \leftarrow \mathbb{LF}(\mathcal{LT}) \setminus \mathbb{LF}(\mathcal{GT})$ ;  $\mathcal{S} \leftarrow \mathbb{UF}(\mathcal{LT}) \setminus \mathbb{UF}(\mathcal{GT})$ 3: compute  $\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT})$  and  $\mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT})$ 4:  $\mathcal{R}'$  and  $\mathcal{S}' \leftarrow \text{Boundary}(\mathbb{LF}(\mathcal{LT}) \cap \mathbb{LF}(\mathcal{GT}), \mathbb{UF}(\mathcal{LT}) \cap \mathbb{UF}(\mathcal{GT}), \mathbb{LF}(\mathcal{GT}), \mathbb{UF}(\mathcal{GT}), k, k)$ 5:  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}'$ ;  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}'$ 6: return  $\mathcal{R}$  and  $\mathcal{S}$ Procedure: **Boundary**INPUT :  $L, U, GL$  and  $GU$  - unions of polyhedra,  $d$  - an integer,  $k$  - total dimensionsOUTPUT :  $R$  and  $S$  - unions of polyhedra1: if  $d = 1$ 2: return  $L$  and  $U$ 3: else -  $d > 1$ 4:  $L' \leftarrow \mathbb{LF}(L) \cap \mathbb{LF}(GL)$ ;  $U' \leftarrow \mathbb{UF}(U) \cap \mathbb{UF}(GU)$ 5:  $R \leftarrow \{L \setminus \text{shift}(\text{array}(U, -e_i, \forall d \leq i \leq k), e_{d-1})\} \setminus L'$ 6:  $S \leftarrow \{U \setminus \text{shift}(\text{array}(L, e_i, \forall d \leq i \leq k), -e_{d-1})\} \setminus U'$ 7:  $\mathcal{R}'$  and  $\mathcal{S}' \leftarrow \text{Boundary}(L', U', \mathbb{LF}(GL), \mathbb{UF}(GU), d-1, k)$ 8:  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}'$ ;  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}'$ 9: return  $\mathcal{R}$  and  $\mathcal{S}$ 

---

Case 1:  $t$  is picked at step 2. Then,  $t \notin \mathbb{LF}(\mathcal{GT})$ . So,  $t - e_k \in \mathcal{GT}$ , i.e.,  $\text{pre}(t) = t - e_k$ . Since  $t \in \mathbb{LF}(\mathcal{LT})$ ,  $t - e_k \notin \mathcal{LT}$ . Hence,  $\text{pre}(t) \notin \mathcal{LT}$ .

Case 2:  $t$  is picked at step 2 of **Boundary**. Then,  $t$  is the lexicographic minimum of  $\mathcal{GT}$ . Therefore, a processor  $p$  needs a resume signal.

Case 3:  $t$  is picked at step 5 of **Boundary**. Suppose that  $t$  is picked at the  $f$ -th recursive call. Then,  $t$  is in the  $f$ -th lower facet of both  $\mathcal{LT}$  and  $\mathcal{GT}$ . Since  $t$  is in the  $f$ -th lower facet of  $\mathcal{GT}$ , i.e.,  $t$  belongs to the  $i$ -th lower facet of  $\mathcal{GT}$  for all  $1 \leq i \leq f$ ,  $t - e_i \notin \mathcal{GT}$  for all  $k - f + 1 \leq i \leq k$ . Also,  $t$  is not in  $(f + 1)$ -th lower facet of  $\mathcal{GT}$ . So, there exists a point  $(t_1, \dots, t_{k-f+1} - 1, t'_{k-f}, \dots, t'_k)$  in  $\mathcal{GT}$ . Hence,  $\text{pre}(t)$  is the form of  $(t_1, \dots, t_{k-f+1} - 1, t''_{k-f}, \dots, t''_k)$ . By the definition of  $\text{pre}(t)$ ,  $\text{pre}(t) + e_i \notin \mathcal{GT}$  for all  $k - f + 1 \leq i \leq k$ . So,  $\text{pre}(t)$  is in the  $f$ -th upper facet of  $\mathcal{GT}$ . Since  $t$  is picked,  $\text{pre}(t)$  is not in the  $f$ -th facet of  $\mathcal{LT}$ . Since  $\text{pre}(t)$  is in the  $f$ -th upper facet of  $\mathcal{GT}$ , the only possibility is that  $\text{pre}(t) \notin \mathcal{LT}$ .

A proof for the suspend signal is symmetric.

#### 4.1 Extension

For the sake of explanation, we had presented our analysis with the view that  $\mathcal{LT}$  is a single polyhedron. However, our analysis and algorithm are more general and are valid for the case when  $\mathcal{LT}$  is a union of polyhedra. When  $\mathcal{GT}$  is a union of polyhedra, the algorithm can be applied to the convex hull  $C(\mathcal{GT})$  of  $\mathcal{GT}$ . In this case, the algorithm may not be exact and return a superset of required signals. Precisely, it will always return the points that belong to  $\mathbb{LF}(\mathcal{GT}) \setminus \mathbb{LF}(C(\mathcal{GT}))$  and  $\mathbb{UF}(\mathcal{GT}) \setminus \mathbb{UF}(C(\mathcal{GT}))$ .

## 5 Discussion on Propagation of Control

In our analysis, we have characterized the precise set of time vectors at which the resume (or suspend) signal is needed. Here, we will discuss, with the help of an example a possible mechanism to propagate these signals to the desired PE.

*Example 4.* Consider the following ACL and its corresponding FGP-ACL for matrix multiplication obtained by the parallelization presented in [6].

```

for i = 1 to M
  for j = 1 to P
    C[i,j] = 0;
    for k = 1 to N
      C[i,j] += A[i,k]*B[k,j];
    for t1 = 2 to M+P
      forall p = max(1,t1-M)
        to min(P,t1-1)
          C[t1-p,p]=0;
          for t2 = 1 to N
            forall p = max(1,t1-M)
              to min(P,t1-1)
                C[t1-p,p]+=A[t1-p,t2]*B[t2,p];

```

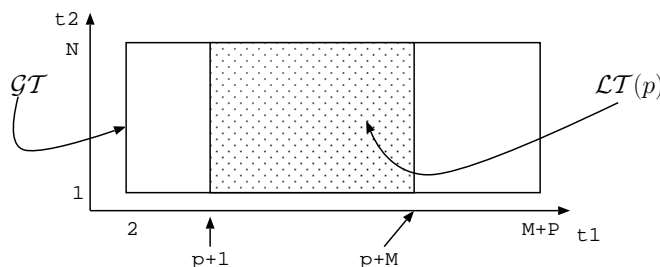


Fig. 6. Global and Local time domains for matrix multiplication

The global time domain is enumerated by the loop nest in the FGP-ACL given above. It is shown in figure 6 along with the local time domain for the PE,  $p$ .

This is a simple design, and by our analysis, there is only one resume and one suspend signal for every PE. These are precisely for starting and stopping the processing element and are needed at the time instants  $(p+1, 1)$  and  $(p+M, N)$  respectively. From the figure, one may verify that the mapping,  $\omega$ , of logical times to concrete time, denoted by  $t^c$  say, is  $\omega(t_1, t_2)^T = N(t_1 - 2) + t_2$  such that the processing commences at the concrete time instant  $t^c = 1$ .

A possible mechanism to *create* and *convey* signals to the relevant PEs would be with the help of a controller connected to the first PE ( $p = 1$ ) and propagation through intermediate PEs. For the first PE, the resume (start) signal is needed at  $t^c = 1$  which is received directly from the controller. The next PE needs this signal at  $t^c = N + 1$ . This too is created by the controller, however, it is propagated through the first PE. If the latency involved with this propagation is a single time instant, the signal is created by the controller at the concrete

time instant  $t^c = N$ . Below, we have given the concrete time instants when the controller creates resume and suspend signals for the PE  $p$ .

$$\begin{aligned}\text{Resume}(p) &= 1 + (N - 1)(p - 1) \\ \text{Suspend}(p) &= MN + (N - 1)(p - 1)\end{aligned}$$

The validity requirement for such a scheme is that an unbounded number of signals are not created at the same concrete time. Any further discussion on the validity is beyond the scope of this paper.

## 6 Related Work

In the context of hardware synthesis, the control signal issue in multidimensional time was recently addressed by Guillou et. al. [6]. They proposed an solution where each PE has a clock enumerating all the points in the global time domain. The clock is an automaton [20] scanning the global time domain. Although it provides a complete solution for control signals and memory addresses, it also introduces a significant overhead to each PE. Although matrix multiplication requires each PE only one adder and multiplier, each PE intrinsically has a loop scanning the global time domain (and possibly some guards).

Bowden et. al. [21] proposed a control signal distribution scheme in linear systolic array. They assumed that a solution for the control signal problem is given as a quadratic function of the processor coordinate. The basic idea is that architecture itself, not each PE, enumerates the global time domain as a whole. They did not address the question of how to derive the quadratic function.

## 7 Conclusions

The theory and techniques for automatic derivation of systolic architectures has a long and rich history. Although systolic arrays are impractical, much of this theory had an impact in the context of automatic parallelization. Several extensions were proposed, notably those that manipulated more general specifications (ACLs), and towards multidimensional schedules. Yet, the hardware compilation of these general specifications has remained an open problem. The contribution of this paper is aimed at the completion of this circle and provides required results for compiling arbitrary ACLs directly to silicon.

In this paper, we formulated the control signal problem; characterized the time instants when a control signal is necessary; and proposed an algorithm to compute exact solution of the control signal problem. Finally, we discussed possible ways to incorporate a solution into hardware so that each PE receives a correct signal at the correct time.

Future work involves devising efficient distribution schemes for control, leading to the automatic generation of the “optimal” control mechanism for multi-dimensional schedules. The generation of array access functions is also an open problem.

## References

1. Bastoul, C., Cohen, A., Girbal, S., Sharma, S., Temam, O.: Putting polyhedral loop transformations to work. In: LCPC 2003. (2003) 209–225
2. Schreiber, R., Aditya, S., Mahlke, S., Kathail, V., Ramakrishna-Rau, B., Conquist, D., Sivaraman, M.: Pico-npa: High level synthesis of nonprogrammable hardware accelerators. *Journal of VLSI Signal Processing* (2001) (to appear) (preliminary version presented at ASAP 2000.
3. Guillou, A.C., Quilleré, F., Quinton, P., Rajopadhye, S., Risset., T.: Hardware design methodology with the alpha language. In: FDL'01. (2001)
4. Feautrier, P.: Some efficient solutions to the affine scheduling problem: Part II. multidimensional time. *Int. J. of Parallel Program.* **21**(6) (1992) 389–420
5. Feautrier, P.: Some efficient solutions to the affine scheduling problem: Part i. one-dimensional time. *Int. J. of Parallel Program.* **21**(5) (1992) 313–348
6. Guillou, A.C., Quinton, P., Risset, T.: Hardware synthesis for multi-dimensional time. In: ASAP 2003. (2003) 40–50
7. Gautam, Renganarayana, L., Rajopadhye, S.: GRAIL: A generic reconfigurable affine interconnection lattice. submitted (2006)
8. Feautrier, P.: Dataflow analysis of array and scalar references. *Int. J. of Parallel Programming* **20**(1) (1991) 23–53
9. Pugh, W.: A practical algorithm for exact array dependence analysis. *Communications of the ACM* **35**(8) (1992) 102–114
10. Karp, R.M., Miller, R.E., Winograd, S.V.: The organization of computations for uniform recurrence equations. *JACM* **14**(3) (1967) 563–590
11. Lamport, L.: The parallel execution of DO loops. *Communications of the ACM* (1974) 83–93
12. Darte, A., Robert, Y., Vivien, F.: *Scheduling and Automatic Parallelization*. Birkhäuser (2000)
13. Rajopadhye, S.V., Purushothaman, S., Fujimoto, R.: On synthesizing systolic arrays from recurrence equations with linear dependencies. In: *Foundations of Software Technology and Theoretical Computer Science*. Volume 241. (1986) 488–503 Later appeared in *Parallel Computing*, June 1990.
14. Quinton, P., Dongen, V.V.: The mapping of linear equations on regular arrays. *J. of VLSI Signal Processing* **1**(2) (1989) 95–113
15. Gautam, Rajopadhye, S., Quinton, P.: Scheduling reductions on realistic machines. In: SPAA '02: Symposium on Parallel algorithms and architectures. (2002) 117–126
16. Lim, A.W., Cheong, G.I., Lam, M.S.: An affine partitioning algorithm to maximize parallelism and minimize communication. In: *Int. Conf. on Supercomputing*. (1999) 228–237
17. Kuck, D.L.: *Structure of Computers and Computations*. John Wiley & Sons, Inc., New York, NY, USA (1978)
18. Bagnara, R., Hill, P.M., Zaffanella, E.: *The Parma Polyhedra Library User's Manual*. Dept of Mathematics, University of Parma, Prama, Italy. version 0.9 edn. (2006) Available at <http://www.cs.unipr.it/ppl/>.
19. Wilde, D.: A library for doing polyhedral operations. Technical Report PI-785, IRISA (1993)
20. Boulet, P., Feautrier, P.: Scanning polyhedra without do-loops. In: PACT'1998. (1998) 4–11
21. Bowden, S., Wilde, D., Rajopadhye, S.V.: Quadratic control signals in linear systolic arrays. In: ASAP 2000. (2000) 268–275