

# Design Space Exploration for Massively Parallel Processor Arrays<sup>\*</sup>

Frank Hannig and Jürgen Teich

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

**Abstract.** In this paper, we describe an approach for the optimization of dedicated co-processors that are implemented either in hardware (ASIC) or configware (FPGA). Such massively parallel co-processors are typically part of a heterogeneous hardware/software-system. Each co-processor is a massive parallel system consisting of an array of processing elements (PEs). In order to decide whether to map a computational intensive task into hardware, existing approaches either try to optimize for performance or for cost with the other objective being a secondary goal. Our approach presented here, instead, a) considers multiple objectives simultaneously. For a given specification, we explore *space-time-mappings* leading to different degrees of parallelism and cost, and different optimal hardware solutions. b) We show that the hardware cost may be efficiently determined in terms of the chosen space-time mapping by using state-of-the-art techniques in polyhedral theory. c) Finally, we introduce ideas to drastically reduce dimension and size of the search space of mapping candidates. d) The feasibility of our approach is shown for two realistic examples.

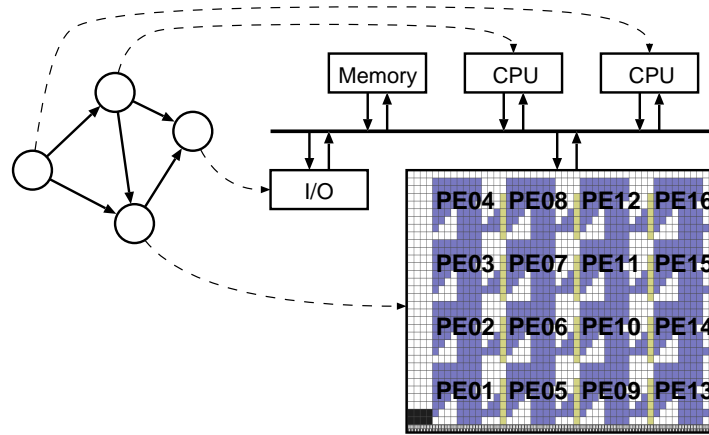
## 1 Introduction

Technical analysts foresee the dilemma of not being able to focus next generation hardware complexity because of a lack of mapping tools. On the other hand, the next generation of ULSI chips will allow to implement arrays of  $10 \times 10$  32-bit micro-processors on a single die and more. Hence, parallelization techniques and compilers will be of utmost importance in order to map computational-intensive algorithms efficiently to these processor arrays.

Through this advance in technology, also reconfigurable hardware, sometimes also called *configware* such as FPGAs (field-programmable gate-arrays) [OD95], becomes more and more attractive as co-processors for the following three reasons: 1) Chips with up to 10 million gate counts allow to implement arithmetic co-processors with hundreds of processing elements, e.g., for image processing and linear algebra algorithms, see, e.g., in Fig. 1. Shown is an FPGA place-

---

<sup>\*</sup> Supported in part by the German Science Foundation (DFG) Project SFB 376 “Massively Parallel Computation”.



**Fig. 1.** Heterogeneous application, architecture, and hardware/software partition including a massively parallel co-processor implemented in hardware (ASIC) or configure (FPGA).

ment visualized by the tool BoardScope by Xilinx [Xil01] with a square array of processing elements (PEs), each consisting of an array-multiplier, an adder, registers, and some control logic. 2) Configure has the major advantage of being able to reuse silicon for time-variant co-processor functions by means of reconfiguration. 3) Support for regular designs: standards such as the Java API JBits [Xil01] allow to specify the regular design within Java-loops such that lower-level mapping may be accomplished efficiently and independent of the problem-size.

In the eighties and early nineties, higher-level mapping techniques for so-called *systolic arrays* have been in its fancy. They pretty much dealt with the problem of mapping a certain algorithm specified by a loop program onto a parallel processor array such as a systolic array, and architectural extensions thereof with time-dependent and control-dependent processor functions [Tei93]. By the use of linear space-time mappings, the relationship between a regular array of communicating PEs and the temporal execution of operations of loop algorithms has been described. Unfortunately, dedicated hardware chips proposed for certain algorithms were too rigid, implementing just a single problem, or too slow and expensive due to long time-to-market.

With the above mentioned advances of silicon technology, and the advent of configure, the necessity of mapping tools for parallel hardware processors has been rethought and its application scope and processor capabilities broadened. Some important recent approaches include the PICO-N system by Hewlett-Packard [SAR<sup>+</sup>00] that specifies a methodology for synthesizing an array of customized VLIW processors starting with a loop program with uniform data dependencies and VHDL code at the RTL-level. From a given irregular program, parts are automatically extracted, mapped to hardware, and finally, the specification is modified to make use of this accelerator. Another approach that embeds regular array design into heterogeneous hardware/software targets is the

Compaan system [KRDL00]. There, Matlab applications are transformed into a network of sequential, communicating processes where each process is responsible for computing some variables of a nested loop program.

In this realm, our paper deals with the specific problem of exploring cost/performance tradeoffs when mapping a certain class of loop-specified computations called *piecewise regular algorithms* [Tei93] onto a dedicated co-processor. The main new ideas of our approach are summarized as follows:

- Simultaneous consideration of multiple objectives: For a given *piecewise regular algorithm*, we explore *space-time-mappings*<sup>1</sup> leading to different degrees of parallelism and cost, and different optimal hardware solutions. Existing approaches such as [FM98] consider solutions that find a schedule first (time-mapping) such to minimize latency and minimize cost as a secondary goal, or the other way round. Such design points are not necessarily so-called *Pareto-optimal* [Par96] points.
- Efficient computation of objectives: We show that hardware cost may be efficiently determined in terms of the chosen space-time mapping by using state-of-the-art techniques in polyhedral theory.
- Search space reduction: We introduce several ideas to drastically reduce dimension and size of the search space of mapping candidates.

The rest of the paper is structured as follows. Section 2 introduces the class of algorithms we are dealing with. In Section 3, the exploration algorithm for finding Pareto-optimal space-time mappings is given. There, the objective functions for cost and performance (latency) are explained including the reduction of the search space. Finally, results are presented in Section 4.

## 2 Notation and Background

### 2.1 Algorithms

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

**Definition 1.** (*Piecewise Regular Algorithm*). *A piecewise regular algorithm contains  $N$  quantified equations*

$$S_1 [I], \dots, S_i [I], \dots, S_N [I]$$

*Each equation  $S_i [I]$  is of the form*

$$x_i [I] = f_i (\dots, x_j [I - d_{ji}], \dots)$$

*where  $I \in \mathcal{I}_i \subseteq \mathbb{Z}^n$ ,  $x_i [I]$  are indexed variables,  $f_i$  are arbitrary functions,  $d_{ji} \in \mathbb{Z}^n$  are constant data dependence vectors, and  $\dots$  denote similar arguments.*

<sup>1</sup> Although we are able to handle also more general classes of algorithms and mappings, introducing them here would unnecessarily complicate the notation and hinder to present the main ideas of the exploration approach.

The domains  $\mathcal{I}_i$  are called index spaces, and in our case defined as follows:

**Definition 2.** (*Linearly Bounded Lattice*). A linearly bounded lattice denotes an index space of the form

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

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  $b \in \mathbb{Z}^m$ .  $\{\kappa \in \mathbb{Z}^l \mid A\kappa \geq b\}$  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 ( $I = M\kappa + c$ ).

Throughout the paper, we assume that the matrix  $M$  is square and of full rank. Then, each vector  $\kappa$  is uniquely mapped to an index point  $I$ . Furthermore, we require that the index space is bounded.

For illustration purposes throughout the paper, the following simple example is used.

*Example 1.* Consider a piecewise regular algorithm which consists of three quantified indexed equations

$$\begin{aligned} a[i, j] &= f(a[i-1, j]), & \forall (i, j)^T = I \in \mathcal{I} \\ b[i, j] &= g(b[i, j-1]), & \forall (i, j)^T = I \in \mathcal{I} \\ c[i, j] &= a[i, j] \text{ op } b[i, j], & \forall (i, j)^T = I \in \mathcal{I}. \end{aligned}$$

The data dependence vectors are  $d_{aa} = (1, 0)^T$ ,  $d_{bb} = (0, 1)^T$ ,  $d_{ac} = (0, 0)^T$ , and  $d_{bc} = (0, 0)^T$ . The index space is given by

$$\mathcal{I} = \left\{ I \in \mathbb{Z}^2 \mid \begin{pmatrix} 1 & -1 \\ -3 & -5 \\ 3 & 4 \\ -4 & 5 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \geq \begin{pmatrix} -3 \\ -63 \\ 26 \\ -14 \end{pmatrix} \right\}.$$

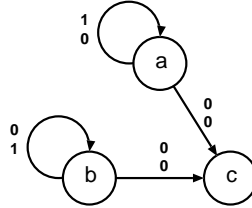
Computations of piecewise regular algorithms may be represented by a *dependence graph* (DG). The DG of the algorithm of Example 1 is shown in Fig. 3 (a). The DG expresses the partial order between the operations. Each variable of the algorithm is represented at every index point  $I \in \mathcal{I}$  by one node. The edges correspond to the data dependencies of the algorithm. They are *regular* throughout the algorithm, i.e.  $a[i, j]$  is directly dependent on  $a[i-1, j]$ . The DG specifies implicitly all legal execution orderings of operations: if there is a directed path in the DG from one node  $a[J]$  to a node  $c[K]$  where  $J, K \in \mathcal{I}$ , then the computation of  $a[J]$  must precede the computation of  $c[K]$ .

Henceforth, and without loss of generality<sup>2</sup>, we assume that all indexed variables are embedded in a common index space  $\mathcal{I}$ . Then, the corresponding dependence graphs can be represented in a reduced form.

**Definition 3.** (*Reduced Dependence Graph*). A reduced dependence graph (RDG)  $G = (V, E, D, \mathcal{I})$  of dimension  $n$  is a network where  $V$  is a set of nodes and  $E \subseteq V \times V$  is a set of edges. To each edge  $e = (v_i, v_j)$  there is associated a dependence vector  $d_{ij} \in \mathbb{Z}^n$ .

<sup>2</sup> All described methods can also be applied for each quantification individually.

*Example 2.* In Fig. 2, the RDG of the algorithm introduced in Example 1 is shown.



**Fig. 2.** Reduced dependence graph.

## 2.2 Space-Time Mapping

Linear transformations as in Equation (1) are used as *space-time mappings* [Mol83,Len93] 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  $I \in \mathcal{I}$ .

$$\begin{pmatrix} p \\ t \end{pmatrix} = \begin{pmatrix} Q \\ \lambda \end{pmatrix} I \quad (1)$$

In Eq. (1),  $Q \in \mathbb{Z}^{(n-1) \times n}$  and  $\lambda \in \mathbb{Z}^{1 \times n}$ . The main reasons for using linear allocation and scheduling functions is that the data flow between PEs is local and regular which is essential for VLSI implementations. The interpretation of such a linear transformation is as follows: The set of operations defined at index points  $\lambda \cdot I = \text{const.}$  are scheduled at the same time step. The index space of allocated processing elements (*processor space*) is denoted by  $\mathcal{Q}$  and is given by the set  $\mathcal{Q} = \{p \mid p = Q \cdot I \wedge I \in \mathcal{I}\}$ . This set can also be obtained by choosing a projection of the dependence graph along a vector  $u \in \mathbb{Z}^n$ , i.e. any coprime<sup>3</sup> vector  $u$  satisfying  $Q \cdot u = 0$  [Kuh80] describes the allocation equivalently.

Allocation and scheduling must satisfy that no data dependencies in the DG are violated. This is ensured by the well-known *causality constraint*

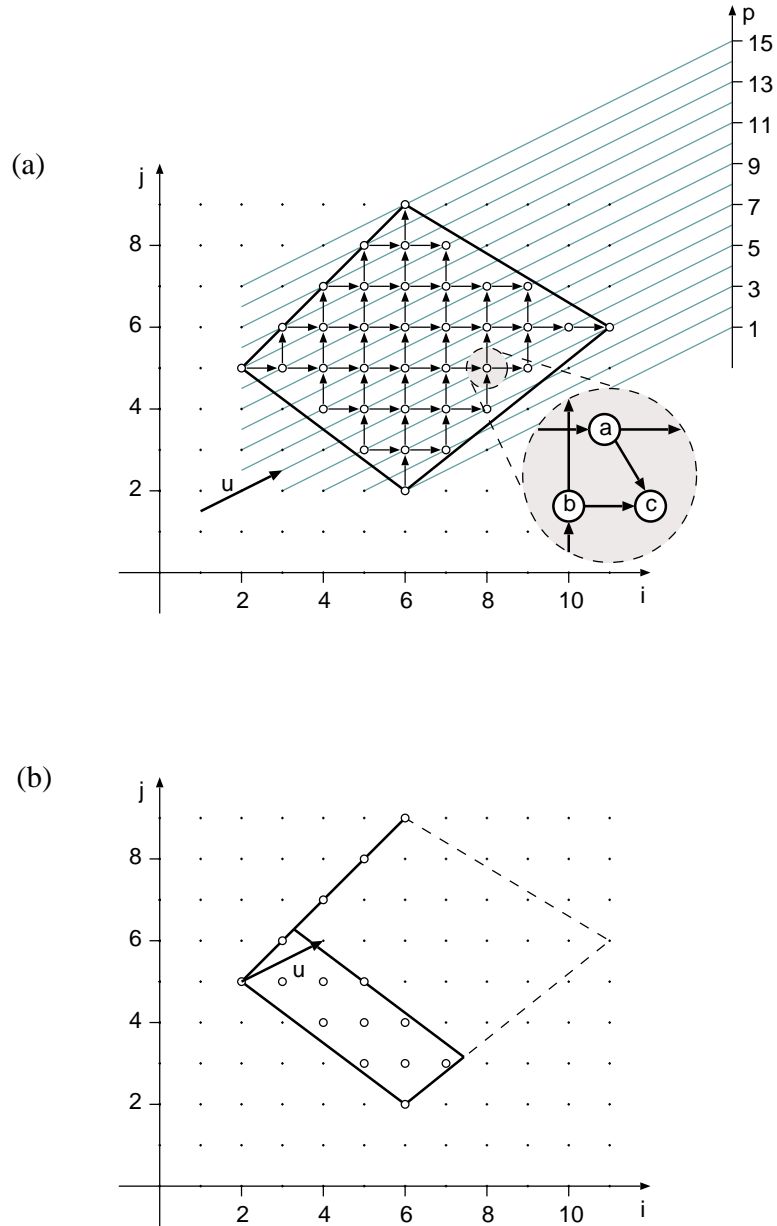
$$\lambda \cdot d_{ij} \geq 0 \quad \forall (v_i, v_j) \in E. \quad (2)$$

A sufficient condition for guaranteeing that no two or more index points are assigned to a processing element at the same time step is given by

$$\text{rank} \begin{pmatrix} Q \\ \lambda \end{pmatrix} = n. \quad (3)$$

Using the projection vector  $u$  satisfying  $Q \cdot u = 0$ , this condition is equivalent to  $\lambda \cdot u \neq 0$  [Rao85].

<sup>3</sup> A vector  $x$  is said to be *coprime* if the absolute value of the greatest value of the greatest common divisor of its elements is one.

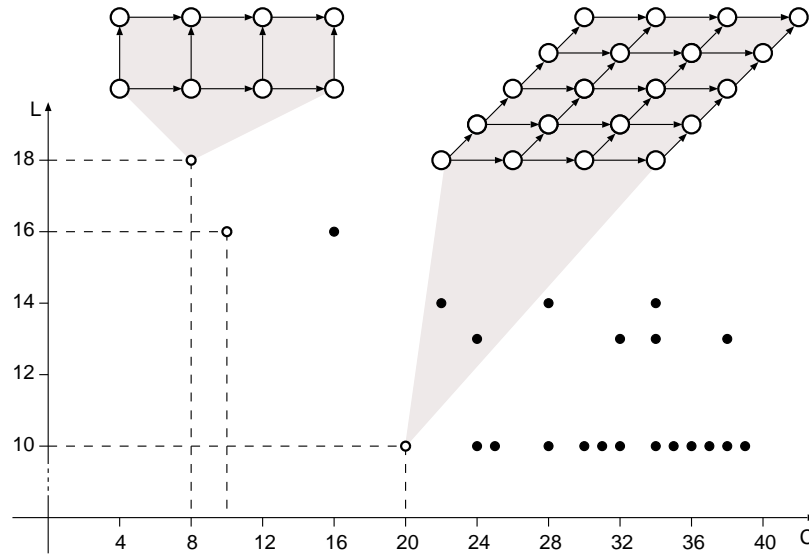


**Fig. 3.** In (a), the dependence graph of the algorithm introduced in Example 1 is shown. Also an allocation given by a projection vector  $u$  is illustrated. Counting the number of processors is equal to counting the number of integral points in a transformed polytope shown in (b) which may be accomplished using Ehrhart polynomials [CL98].

### 3 Methodology

Based on the class of piecewise regular algorithms, we want to explore space-time mappings systematically in order to find optimal implementations. Thereby, we want to simultaneously minimize several objectives, a multiobjective optimization problem (MOP). In this paper, we consider the two objectives *latency*  $L(Q, \lambda)$  as a measure for the performance, and *cost*  $C(Q, \lambda)$  of a processor array.

As  $C$  and  $L$  are dependent on  $Q$  and  $\lambda$ , the search space contains  $n \times n$  parameters. But as already mentioned, a linear allocation can be described equivalently through a coprime projection vector  $u$ . Thus, the dimension of the search space can be reduced to  $2 \times n$  (vector  $u$ , vector  $\lambda$ ).



**Fig. 4.** Pareto-points for the matrix multiplication example in the objective space of latency ( $L$ ) and cost ( $C$ ).

Fig. 4 shows a typical tradeoff curve between cost and performance for a matrix multiplication algorithm. Different pairs of latency and cost correspond to different space-time mappings. As we are concerned with a MOP, there is not only one optimal solution but typically a set of optimal solutions, so called *Pareto-optimal* solutions. Our MOP consists of two objective functions  $C(u, \lambda)$  and  $L(u, \lambda)$ , where the parameters  $u$  and  $\lambda$  are denoted as decision variables. The optimization goal is to simultaneously minimize  $C(u, \lambda)$  and  $L(u, \lambda)$  within a search space of feasible space-time mappings.

**Definition 4.** (*Search Space, Decision Vector*). Let  $x = (u \ \lambda)^T \in \mathbb{Z}^{2n}$  denote a decision vector and  $\mathbf{X}$  denote decision space of all vectors  $x$  satisfying Eq. (1), (2) and (3).

**Definition 5.** (*Pareto-optimality*). For any two decision vectors  $\mathbf{a}, \mathbf{b} \in \mathbf{X}$ ,  $\mathbf{a}$  dominates  $\mathbf{b}$  iff  $(C(\mathbf{a}) < C(\mathbf{b}) \wedge L(\mathbf{a}) \leq L(\mathbf{b})) \vee (C(\mathbf{a}) \leq C(\mathbf{b}) \wedge L(\mathbf{a}) < L(\mathbf{b}))$ . A decision vector  $\mathbf{x} \in \mathbf{X}$  is said to be non-dominated regarding a set  $\mathbf{A} \subseteq \mathbf{X}$  iff  $\nexists \mathbf{a} \in \mathbf{A} : \mathbf{a}$  dominates  $\mathbf{x}$ . Moreover,  $\mathbf{x}$  is said to be Pareto-optimal iff  $\mathbf{x}$  is non-dominated regarding  $\mathbf{X}$ .

In Fig. 4, the objective space of an example discussed later in Section 4 is shown. The white points correspond to Pareto-optimal solutions because they are not dominated by any other point. Dominated points are shown in black.

Now, we are able to formulate our exploration algorithm. For a given RDG  $G = (V, E, D, \mathcal{I})$  and a set  $U$  of projection vectors  $u$ , our exploration methodology works as follows: First, the cost  $C$  for a given projection vector  $u$  is determined. For this allocation, the minimal latency  $L$  is computed. Afterwards, we determine if the design point is non-dominated with respect to the actual set of Pareto-optimal solutions. If it is non-dominated, the decision vector  $(u \ \lambda)^T$  is added to the Pareto-optimal set, denoted  $\mathcal{O}$  in the following. Subsequently, the set  $\mathcal{O}$  has to be updated if the new decision vector dominates some other vectors in  $\mathcal{O}$ . In the following algorithm, the main ideas of our exploration methodology are described.

#### EXPLORE

```

IN:   RDG, set  $U$  of projection vector candidates
OUT:  Pareto-optimal set  $\mathcal{O}$ 
BEGIN
  FOR each candidate  $u \in U$  DO
     $C \leftarrow \text{determineNoOfPEs}(u)$ 
     $L \leftarrow \text{minimize}_\lambda \{L(u, \lambda)\}$ 
    IF  $(u \ \lambda)^T$  is non-dominated with respect to  $\mathcal{O}$  THEN
       $\mathcal{O} \leftarrow \mathcal{O} \cup \{(u \ \lambda)^T\}$ 
      update( $\mathcal{O}$ )
    ENDIF
  ENDFOR
END
```

Next, we briefly describe how the cost  $C$  and the latency  $L$  may be computed. Afterwards, we describe how to reduce the set  $U$  of candidate vectors  $u$  that must be investigated.

### 3.1 Cost

For a regular processor array, we are able to approximate the cost, as being proportional to the processor count.

$$C(u, \lambda) = \#PE(u) \cdot (c_{\text{FU}} + c_{\text{Rg}}(\lambda) + c_{\text{Wire}}(u)) \quad (4)$$

In Eq. (4),  $\#PE(u)$  denotes the number of projected index points when projecting the index space  $\mathcal{I}$  along  $u$  (see, e.g., Fig. 3 (a)). The cost for functional

units, registers and wiring is denoted by  $c_{\text{FU}}$ ,  $c_{\text{Rg}}$  and  $c_{\text{Wire}}$ . In the following, we assume that processor arrays are resource-dominant: This means that  $c_{\text{FU}} \gg c_{\text{Rg}}(\lambda) + c_{\text{Wire}}(u)$ . Under these assumptions, we obtain the approximation:

$$C(u, \lambda) \approx C(u) = \#PE(u) \cdot c_{\text{FU}} \quad (5)$$

As a consequence, the cost of an array is independent of the schedule and proportional to the number of points in the projected polytope  $\mathcal{I}$ . This is also the reason why we are able to investigate only the projection vector candidates  $u \in U$  and minimize the latency  $L$ .

It remains to determine the number of processor elements for a given linear allocation. Here, a geometrical approach recently proposed in [Cla96] is applied, for illustration, see Fig. 3. In (a), the index space of the algorithm described in Example 1 and a projection vector  $u = (2 \ 1)^T$  is shown. This linear allocation leads to an array of 15 processors. This number of processor elements can be determined by a transformation of the given polytope  $\mathcal{I}$ . The number of integral points inside this transformed polytope is equal to the number of processor elements obtained by the projection along  $u$ . In [Cla96], it has been shown that this problem is equal to a counting problem of the number of integral points in a transformed polytope, see e.g. the polytope shown in Fig. 3 (b) for the algorithm of Example 1. The number of processors using the projection vector  $u = (2 \ 1)^T$  results in 15 different projected PEs. This is exactly the number of integral points inside the polytope shown in Fig. 3 (b), see [Cla96] for details. A state-of-the-art solution to the final counting problem is to use so-called *Ehrhart polynomials*<sup>4</sup> [CL98].

### 3.2 Latency

In this section, a short description is given how the latency for a given piecewise regular algorithm and a given schedule vector  $\lambda$  is determined. For approximation of the latency, the following term is used

$$L = \max_{I \in \mathcal{I}} \{\lambda \cdot I\} - \min_{I \in \mathcal{I}} \{\lambda \cdot I\} = \max_{I_1, I_2 \in \mathcal{I}} \{\lambda \cdot (I_2 - I_1)\}.$$

The latency minimization problem in algorithm EXPLORE may be formulated as a mixed integer linear program (MILP) [Thi95,TTZ97]. This well-known method is used here during exploration as a subroutine. In this MILP, the number of resources inside each processing element can be limited (determining  $c_{\text{FU}}$ ). Also given is the possibility that an operation can be mapped onto different resource types (module selection), and pipelining is also possible. As a result of the MILP, we obtain:

- the minimal latency  $L$ ,
- the according optimal schedule vector  $\lambda$ ,
- the iteration interval<sup>5</sup>  $P$ ,

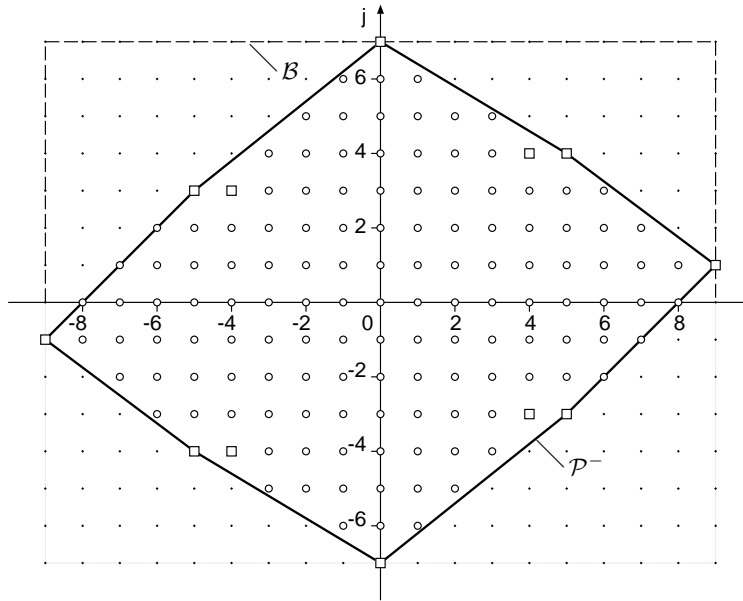
<sup>4</sup> Due to space limits, we omit the details of this procedure.

<sup>5</sup> The iteration interval  $P$  of an allocated and scheduled piecewise regular algorithm is the number of time instances between the evaluation of two successive instances of a variable within one processing element [Thi95].

- the start times of each  $v_i \in V$ , within the iteration interval
- the selected resource type for each  $v_i \in V$ .

Here, only the latency is used for rating the performance. The other values, however, are necessary for simulation and synthesis. We will present a detailed example of this procedure in Section 4.

In the following, we introduce two new additional methods how to reduce the search space for Pareto-optimal space-time mappings.



**Fig. 5.** Difference body of the convex polytope from Fig. 3 (a).

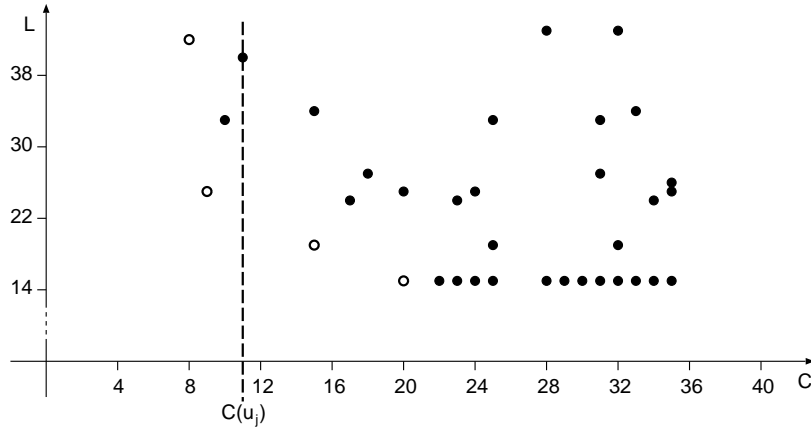
### 3.3 Projection vector candidates

Let  $\mathcal{I} \subset \mathbb{Z}^n$  be a linearly bounded lattice according to Definition 2. In the following, we investigate projection vectors for the polytope  $\mathcal{P} = \{\kappa \in \mathbb{Z}^n \mid A\kappa \geq b\}$ . By our assumption that the lattice matrix  $M$  has full rank, projection vectors  $u' \in \mathbb{Z}^n$  for  $\mathcal{P}$  may be transformed to a corresponding projection vector  $u \in \mathbb{Z}^n$  in  $\mathcal{I}$  by  $u = Mu'$ .

For the exploration, it is necessary to determine a set  $U$  of projection vector candidates. This search space may be bounded as follows: Note that a projection vector may not be optimal if not at least two points  $\kappa_1, \kappa_2 \in \mathcal{P}$  are projected onto each other:

$$\kappa_1 - \kappa_2 = \alpha u', \quad \alpha \in \mathbb{Z}. \quad (6)$$

Hence, the search space may be bounded by the set of possible differences of



**Fig. 6.** Pareto-points obtained through design space exploration for the algorithm introduced in Example 1.

two points in  $\mathcal{P}$ , the so-called *difference body*  $\mathcal{D}$  of  $\mathcal{P}$  [WD89], which again is a polytope.

$$\mathcal{D} = \{\kappa \in \mathbb{Z}^n \mid \kappa = \kappa_1 - \kappa_2 \wedge \kappa_1, \kappa_2 \in \mathcal{P}\}.$$

The *dual*  $\mathcal{P}^-$  of  $\mathcal{D}$  is convex and symmetric about the origin (see, e.g., in Fig. 5 for the polytope  $\mathcal{P}$  in Fig. 3 (a)). From duality,  $\mathcal{P}^- = \{\kappa \in \mathbb{Z}^n \mid A^- \kappa \geq b^-\}$  is the intersection of closed half-spaces. Furthermore, let  $\mathcal{B} \subset \mathbb{Z}^n$  be the smallest  $n$ -dimensional box (*bounding box*) containing  $\mathcal{P}^-$ .

In the following, a procedure for the reduction of suitable projection vector candidates is described:

- Compute all vertices  $\mathcal{V}$  of the polytope  $\mathcal{P}$ .
- For each pair  $v_i, v_j \in \mathcal{V}$  compute the vertex difference  $v_i - v_j$ . The set of vertex differences is denoted by  $\mathcal{V}^-$ .
- Determine the dual representation of  $\mathcal{V}^-$ . This is the convex polytope  $\mathcal{P}^-$ . Also determine the bounding box  $\mathcal{B}$  of  $\mathcal{V}^-$ .
- Iterate over all points  $u' \in \mathcal{B}$ . For the reason  $\mathcal{P}^-$  is symmetric about the origin, also  $\mathcal{B}$  is symmetric about the origin. Due to symmetry, it is only necessary to consider, e.g., for the first component of  $u'$  all positive values. Furthermore, the selected projection vectors  $u'$  have to be coprime. Finally, test if  $u'$  is in  $\mathcal{P}^-$ . If  $u' \in \mathcal{P}^-$ , the condition in Eq. (6) that at least two point mapped onto each other is satisfied.

*Example 3.* Reconsider the polytope shown in Fig. 3 (a) with the vertices

$$v_1 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 6 \\ 9 \end{pmatrix}, v_3 = \begin{pmatrix} 11 \\ 6 \end{pmatrix}, v_4 = \begin{pmatrix} 6 \\ 2 \end{pmatrix}.$$

All differences  $v_i - v_j$ ,  $i, j \in [1, 4]$ ,  $i \neq j$  are marked in Fig. 5 as white small boxes.  $\mathcal{P}^-$  is bounded by the black,  $\mathcal{B}$  by the dashed line. Due to symmetry, only the upper half-space has to be explored. All coprime integral points  $(i \ j)^T$ ,  $i \in [-9, 9]$ ,  $j \in [0, 7]$  which lie inside  $\mathcal{P}^-$  are projection vector candidates.

### 3.4 Further reduction of the search space

The order in our exploration algorithm to determine the cost first has the advantage that possibly the search space can be reduced further by adding a more restrictive constraint to the MILP for latency minimization: Let  $\mathcal{O}$  be the set of so far determined Pareto-points (see Fig. 6). The dashed line denotes the computed cost of a design point  $(u_j \lambda_j)^T$ . If this design point shall be Pareto-optimal, obviously  $L(\lambda_j)$  must be smaller or equal to the latency  $L(\lambda_i)$  of all such points  $o_i \in \mathcal{O}$  for which the cost  $C(u_i)$  is smaller or equal to  $C(u_j)$ :

```

IF ( $\exists o_i = (u_i \lambda_i)^T \in \mathcal{O} \mid C(u_i) \leq C(u_j)$ ) THEN
  let  $o_i \in \mathcal{O}$  be the Pareto-point for which
   $\max_{o_i \in \mathcal{O}} \{C(u_i) \mid C(u_i) \leq C(u_j)\}$  holds
  IF ( $C(u_i) < C(u_j)$ ) THEN
    add constraint  $L(\lambda_j) < L(\lambda_i)$  to MILP
  ELSE
    IF ( $C(u_i) = C(u_j)$ ) THEN
      add constraint  $L(\lambda_j) \leq L(\lambda_i)$  to MILP
    ENDIF
  ENDIF
ENDIF

```

## 4 Results

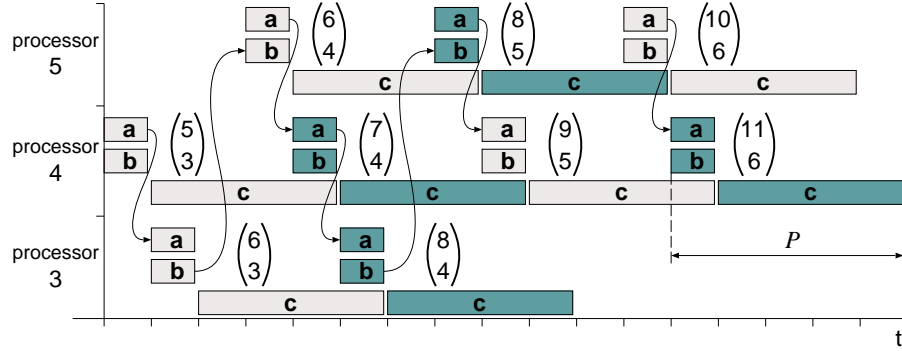
First, space-time mappings for the algorithm introduced in Example 1 are explored. The bounding box (Fig. 5) contains 295 integral points as candidates for projection vectors. When symmetry is explored and only coprime vectors are considered,  $U$  is reduced to 45 candidates. For each of these projection vectors, the cost  $C$  is determined. Subsequently, the latency is minimized. The results are visualized in Fig. 6, the Pareto-optimal solutions are the white points and presented in Table 1. The MILP was solved for execution times of 1 unit for  $f(a)$

**Table 1.** Pareto-points of the design space exploration for the algorithm in Example 1.

$u$	$\lambda$	$C$	$L$
$(1 \ 0)^T$	$(4 \ 1)$	8	42
$(1 \ 1)^T$	$(2 \ 2)$	9	25
$(2 \ 1)^T$	$(1 \ 2)$	15	19
$(3 \ 1)^T$	$(1 \ 1)$	20	15

and  $g(b)$ . For  $op$ , we considered 4 time units. From the solution of the MILP, we obtain the schedule vector  $\lambda$ , the iteration interval  $P$  and as well all starting times for each operation within the iteration interval. In the following, we take a closer look at the solution for  $u = (2 \ 1)^T$ . The corresponding iteration interval

is 4 and the starting points are  $\tau(a) = 0$ ,  $\tau(b) = 0$ , and  $\tau(c) = 1$ . In Fig. 7, the scheduling for the processors  $p = 3, 4$ , and 5 is shown. The data dependencies between adjacent index points are visualized by arcs.



**Fig. 7.** Bar chart of scheduled algorithm.

The second example is a matrix multiplication algorithm. The product  $C = A \cdot B$  of two matrices  $A \in \mathbb{R}^{N_1 \times N_3}$  and  $B \in \mathbb{R}^{N_3 \times N_2}$  is defined as follows

$$c_{ij} = \sum_{k=1}^{N_3} a_{ik} b_{kj} \quad \forall 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2.$$

A corresponding piecewise regular algorithm is given by

*input operations*

$$\begin{aligned} a[i, 0, k] &\leftarrow a_{ik} & 1 \leq i \leq N_1 \wedge 1 \leq k \leq N_3 \\ b[0, j, k] &\leftarrow b_{kj} & 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3 \\ c[i, j, 0] &\leftarrow 0 & 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \end{aligned}$$

*computations*

$$\begin{aligned} a[i, j, k] &\leftarrow a[i, j-1, k] & 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3 \\ b[i, j, k] &\leftarrow b[i-1, j, k] & 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3 \\ z[i, j, k] &\leftarrow a[i, j, k] \cdot b[i, j, k] & 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3 \\ c[i, j, k] &\leftarrow c[i, j, k-1] + z[i, j, k] & 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3 \end{aligned}$$

*output operations*

$$c_{ij} \leftarrow c[i, j, N_3] \quad 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2$$

where the index space is

$$\mathcal{I} = \{I = (i \ j \ k)^T \in \mathbb{Z}^3 \mid 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3\}.$$

The input operations  $a$  and  $b$  are mapped each to one resource of type *input*. The execution times of these operations are zero. This is equivalent to a multi-cast without delay to a set of index points. For the multiplication (variable  $z$ ), an execution time of 4 time units is considered, whereby the multiplier is pipelined,

being able to start a new execution every two time units. The addition (variable  $c$ ) takes three time units and by use of pipelining is able to start each time unit a new operation.

An exploration for  $N_1 = 4$ ,  $N_2 = 5$  and  $N_3 = 2$  has been performed. The search space of  $|[-4, 4]| \cdot |[-5, 5]| \cdot |[-2, 2]| = 9 \cdot 11 \cdot 5 = 495$  projection vector candidates can be reduced to 83 using our reduction techniques. The results are visualized in Fig. 4. We obtain three Pareto-optimal solutions shown in Table 2.

**Table 2.** Pareto-points of the design space exploration for the matrix multiplication algorithm.

$u$	$\lambda$	$C$	$L$
$(1\ 0\ 0)^T$	$(2\ 0\ 3)$	10	16
$(0\ 1\ 0)^T$	$(0\ 2\ 3)$	8	18
$(0\ 0\ 1)^T$	$(0\ 0\ 3)$	20	10

## 5 Conclusion and Future Work

We have presented a first approach for systematically exploring Pareto-optimal space-time mappings for a class of algorithms with uniform data dependencies. The considered objective functions are cost and performance (latency). In our exploration algorithm we introduced also several new techniques for reduction of search space for Pareto-optimal space-time mappings.

Our exploration framework is part of the PARO<sup>6</sup> design system that supports also the automated synthesis of regular circuits.

In the future, we would like to extend the presented results to include energy consumption as an additional objective and to perform symbolic design space exploration for parameterized index spaces.

## References

- [CL98] Philippe Clauss and Vincent Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *Journal of VLSI Signal Processing*, 19(2):179–194, July 1998.
- [Cla96] Philippe Clauss. Counting Solutions to Linear and Nonlinear Constraints through Ehrhart polynomials: Applications to Analyse and Transform Scientific Programs. In *Tenth ACM International Conference on Supercomputing*, Philadelphia, Pennsylvania, May 1996.

<sup>6</sup> PARO is a design system project for modeling, transforming, optimization, and processor synthesis for the class of piecewise linear algorithms. For further information, check the website: <http://www-date.upb.de/research/paro/>.

- [FM98] Dirk Fimmel and Renate Merker. Determination of Processor Allocation in the Design of Processor Arrays. *Microprocessors and Microsystems*, 22(3–4):149–155, 1998.
- [KRDL00] Bart Kienhuis, Edwin Rijpkema, Ed F. Deprettere, and Paul Lieverse. High Level Modeling for Parallel Executions of Nested Loop Algorithms. In *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 79–91, Boston, Massachusetts, 2000.
- [Kuh80] 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 Lafayette, IN, April 1980.
- [Len93] 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.
- [Mol83] Dan I. Moldovan. On the Design of Algorithms for VLSI Systolic Arrays. In *Proceedings of the IEEE*, volume 71, pages 113–120, January 1983.
- [OD95] John V. Oldfield and Richard C. Dorf. *Field Programmable Gate Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems*. John Wiley & Sons, Chichester, New York, 1995.
- [Par96] Vilfredo Pareto. *Cours d'Économie Politique*, volume 1. F. Rouge & Cie., Lausanne, Switzerland, 1896.
- [Rao85] S. K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays*. PhD thesis, Stanford University, 1985.
- [SAR<sup>+</sup>00] Robert Schreiber, Shail Aditya, B. Ramakrishna Rau, Vinod Kathail, Scott Mahlke, Santosh Abraham, and Greg Snider. High-Level Synthesis of Non-programmable Hardware Accelerators. In *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 113–124, Boston, Massachusetts, 2000.
- [Tei93] Jürgen Teich. *A Compiler for Application-Specific Processor Arrays*. PhD thesis, Institut für Mikroelektronik, Universität des Saarlandes, Saarbrücken, Germany, 1993.
- [Thi95] Lothar Thiele. Resource Constrained Scheduling of Uniform Algorithms. *Journal of VLSI Signal Processing*, 10:295–310, 1995.
- [TTZ97] Jürgen Teich, Lothar Thiele, and Li Zhang. Scheduling of Partitioned Regular Algorithms on Processor Arrays with Constrained Resources. *Journal of VLSI Signal Processing*, 17(1):5–20, September 1997.
- [WD89] Yiwang Wong and Jean-Marc Delosme. Optimization of Processor Count for Systolic Arrays. Technical Report YALEEU/DCS/RR-697, Yale University, Department of Computer Science, New Haven, Connecticut, 1989.
- [Xil01] Xilinx, Inc. <http://www.xilinx.com/products/software/jbits/>