Design of Processor Arrays for Real-Time Applications

Dirk Fimmel Renate Merker

Department of Electrical Engineering *
Dresden University of Technology
fimmel,merker@iee1.et.tu-dresden.de

Abstract. This paper covers the design of processor arrays for algorithms with uniform dependencies. The design constraint is a limited latency of the resulting processor array. As objective of the design the minimization of the costs for an implementation of the processor array in silicon is considered.
Our approach starts with the determination of a set of proper linear allocation functions with respect to the number of processors. It follows the computation of a uniform affine scheduling function. Thereby, a module selection and the size of partitions of a following partitioning is determined. A proposed linearization of the arising optimization problems permits the application of integer linear programming.

1 Introduction

Processor arrays represent an appropriate kind of co-processors for time consuming algorithms. Especially in heterogeneous systems a dedicated hardware for parts of algorithms, e.g. for the motion estimation of MPEG, is required. An admissible latency L_g for the selected part of the algorithm can be derived from the time constraint of the entire algorithm. The objective of the design is a processor array with minimal hardware costs that matches the admissible latency L_g.
In this paper, we consider algorithms with uniform dependencies. First, a set of linear allocation functions that lead to a small number of processors is computed. Then, for each allocation function a uniform affine scheduling function is determined. We assume that a LSGP (locally sequential and globally parallel) -partitioning can be applied to the processor array. The size of the partitions is derived with respect to the constraint that the latency of the resulting processor array is less than L_g. Furthermore, the processor functionality, i.e. kind and number of modules that have to be implemented in each processor, is determined. The objective of the design is a minimization of hardware costs. We measure hardware costs by the chip area needed to implement modules and registers in silicon.

* The research was supported by the "Deutsche Forschungsgemeinschaft", in the project A1/SFB358.

Related works handling the allocation function cover (1) a limited enumeration process to determine a linear allocation function leading to a minimal number of processors of the processor array [15], (2) the determination of a set of linear allocation functions that match a given interconnection network of the processor array [16], (3) the inclusion of a limited radius of the interconnections into the determination of the allocation function [14] and (4) the minimization of the chip area of the processor array by consideration of the processor functionality [3]. An approach to compute a variety of linear allocation and scheduling functions is proposed in [11]. Some notes to the determination of unconstrained minimal scheduling functions for algorithms with uniform dependencies can be found in [1]. Resource constraint scheduling for a given processor functionality is presented in [13]. An approach to minimize the throughput by consideration of the chip area is proposed in [9]. In [2] the approach [13] is extended to determine additionally the processor functionality in order to minimize a chip area - latency product.
The paper is organized as follows. Basics of the design of processor arrays are given in section 2. In section 3 hardware constraints considered in this paper are introduced. A linear program to determine a set of linear allocation functions is presented in section 4. Section 5 covers the determination of scheduling functions. A linear programming approach is presented in detail. Finally, a short conclusion is given in section 6.

2 Design of Processor Arrays

In this paper we restrict our attention to the class of algorithms that can be described as systems of uniform recurrence equations (SURE) [5].

Definition 1 (System of uniform recurrence equations). *A system of uniform recurrence equations is a set of equations S_i of the following form:*

$$S_i: \quad y_i[\mathbf{i}] = F_i(\cdots, y_j[f_{ij}^k(\mathbf{i})], \cdots), \quad \mathbf{i} \in \mathcal{I}, \quad 1 \leq i,j \leq m, 1 \leq k \leq m_{ij}, \quad (1)$$

where $\mathbf{i} \in \mathbb{Z}^n$ is an index vector, $f_{ij}^k(\mathbf{i}) = \mathbf{i} - \mathbf{d}_{ij}^k$ are index functions, the constant vectors $\mathbf{d}_{ij}^k \in \mathbb{Z}^n$ are called dependence vectors, y_i are indexed variables and F_i are arbitrary single valued operations. All equations are defined in the index space $\mathcal{I}$ being a polytope $\mathcal{I} = \{\mathbf{i} \mid \mathbf{H}_i\mathbf{i} \geq \mathbf{h}_{i0}\}$, $\mathbf{H}_i \in \mathbb{Q}^{m_i \times n}, \mathbf{h}_{i0} \in \mathbb{Q}^{m_i}$.

We suppose that the SURE has a single assignment form (every instance of a variable y_i is defined only once in the algorithm) and that there exists a partial order of the instances of the equations that satisfies the data dependencies.
Next, we introduce a graph representation of the data dependencies of the SURE.

Definition 2 (Reduced dependence graph (RDG)). *The equations of the SURE build the m nodes $v_i \in \mathcal{V}$ of the reduced dependence graph $\langle \mathcal{V}, \mathcal{E} \rangle$. The directed edges $e = (v_i, v_j) \in \mathcal{E}$ are the data dependencies weighted by the dependence vectors $\mathbf{d}(e) = \mathbf{d}_{ij}^k$. Source and sink of an edge $e \in \mathcal{E}$ are called $\sigma(e)$ and $\delta(e)$ respectively.*

The main task of the design of processor arrays is the determination of the time and the processor when and where each instance of the equations of the SURE has to be evaluated. In order to keep the regularity of the algorithm in the resulting processor array, we apply only uniform affine mappings [8] to the SURE.

Definition 3 (Uniform affine scheduling). *A uniform affine scheduling function $\tau_i(\mathbf{i})$ assigns an evaluation time to each instance of the equations:*

$$\tau_i : \mathbb{Z}^n \to \mathbb{Z} : \quad \tau_i(\mathbf{i}) = \boldsymbol{\tau}^T \mathbf{i} + t_i, \quad 1 \leq i \leq m, \tag{2}$$

where $\boldsymbol{\tau} \in \mathbb{Z}^n, t_i \in \mathbb{Z}$.

Definition 4 (Linear processor allocation). *A linear allocation function $\pi(\mathbf{i})$ assigns an evaluation processor to each instance of the equations:*

$$\pi : \mathbb{Z}^n \to \mathbb{Z}^{n-1} : \quad \pi(\mathbf{i}) = \mathbf{S}\mathbf{i}, \tag{3}$$

where $\mathbf{S} \in \mathbb{Z}^{n-1 \times n}$ is of full row rank. Since $\mathbf{S}$ is of full row rank, the vector $\mathbf{u} \in \mathbb{Z}^n$ which is coprime and satisfies $\mathbf{S}\mathbf{u} = \mathbf{0}$ and $\mathbf{u} \neq \mathbf{0}$ is uniquely defined and called projection vector.

Because of the lack of space we refer to [2,3] for a treatment of uniform affine allocation functions $\pi_i : \mathbb{Z}^n \to \mathbb{Z}^{n-1} : \quad \pi_i(\mathbf{i}) = \mathbf{S}\mathbf{i} + \mathbf{p}_i$, $1 \leq i \leq m$.
The importance of the projection vector is due to the fact that those and only those index points of an index space lying on a line spanned by the projection vector $\mathbf{u}$ are mapped onto the same processor. Due to the regularity of the index space and the uniform affine scheduling function, the processor executes the operations associated with that index points one after each other if $\boldsymbol{\tau}^T\mathbf{u} \neq 0$ with a constant time distance $\lambda = |\boldsymbol{\tau}^T\mathbf{u}|$ which is called iteration interval.
The application of a scheduling and an allocation function to a SURE results in a so called fullsize array.

3 Hardware Description

We consider a given set $\mathcal{M}$ of modules which are responsible to evaluate the operations of a processor. Instead of assuming given processors we want to determine modules which realize the operations of the processors. First, we introduce some measures needed to describe the modules. To each module $m_l \in \mathcal{M}$ we assign an evaluation time $d_l \in \mathbb{Z}$ in clock cycles needed to execute the operation of module m_l, a necessary chip area $c_l \in \mathbb{Z}$ needed to implement the module in silicon and the number $n_l \in \mathbb{Z}$ of instances of that module which are implemented in one processor. If a module $m_l \in \mathcal{M}$ has a pipeline architecture we assign a time offset o_l to that module which determines the time delay after that the next computation can be started on this module, otherwise $o_l = d_l$. Some modules are able to compute different operations, i.e. a multiplication unit is likewise able to compute an addition. To such modules different delays d_{li} and offsets o_{li} depending on the operations F_i are assigned.

The assignment of a module $m_l \in \mathcal{M}$ to an operation F_i is denoted as $m(i)$, and the set of modules which are able to perform the operation F_i is $\mathcal{M}_i$. The addressing of the instance of the module $m(i)$ which performs the operation F_i is given by $u_i \in \mathbb{Z}$.

4 Determination of Allocation Functions

The allocation function maps each index vector $\mathbf{i} \in \mathcal{I}$ to a processor $\mathbf{p}$ of the processor space $\mathcal{P} = \{\mathbf{p} \mid \mathbf{p} = \mathbf{Si} \wedge \mathbf{i} \in \mathcal{I}\}$. Our aim is the determination of a set of proper linear allocation functions that lead to processor spaces with a small number of processors.
In our approach we approximate the number of processors of the processor space $\mathcal{P}$ by the number of processors of the enclosing constant bounded polytop (cb-polytop) $\mathcal{Q} = \{\mathbf{p} \mid \mathbf{p}_{min} \leq \mathbf{p} \leq \mathbf{p}_{max}\}$ of $\mathcal{P}$, where $\mathcal{P} \subseteq \mathcal{Q}$, and each face of $\mathcal{Q}$ intersects $\mathcal{P}$ at least in one point. The consideration of the cb-polytop $\mathcal{Q}$ allows the formulation of the search for allocation functions as a linear optimization problem.

Program 1 (Determination of allocation functions)
for $j = 1$ **to** $n - 1$

$$\begin{array}{lll} \textit{minimize } v_j^1 - v_j^2 + 1, & v_j^1, v_j^2 \in \mathbb{Q}, & \\ \textit{subject to } v_j^2 \leq \mathbf{s}_j^T \mathbf{w}_l \leq v_j^1, & 1 \leq l \leq |\mathcal{W}|, \mathbf{s}_j \in \mathbb{Z}^n, & \\ \mathbf{s}_j^T \mathbf{b}_k + (1 - r_k)R \geq 1, & 1 \leq k \leq n - j + 1, \quad (4.1) & (4) \\ \sum_{k=1}^{n-j+1} r_k = 1, & r_k \in \{0, 1\}, & \end{array}$$

end for
where $\mathcal{W}$ is the set of vertices $\mathbf{w}_l$ of $\mathcal{I}$, R is a sufficiently large constant and the vectors $\mathbf{b}_k$, $1 \leq k \leq n - j + 1$, are spanning the right null space of the matrix $(\mathbf{s}_1, \cdots, \mathbf{s}_{j-1})^T$. Constraint (4.1) is replaced by $\mathbf{s}_1 \neq \mathbf{0}$ for $j = 1$.

Constraint (4.1) ensures that the vectors $\mathbf{s}_j$, $1 \leq j \leq n-1$, are linearly independent, i.e. that $rank(\mathbf{S}) = n-1$. Constant R has to fulfill $R \geq \max_{1 \leq k \leq n-j+1} \{|\mathbf{s}_j^T \mathbf{b}_k|\}$.

The number of processors of the cb-polytop $\mathcal{Q}$ is $\prod_{j=1}^{n-1} (v_j^1 - v_j^2 + 1)$.

A motivation of our approach is given in the following theorem.

Theorem 1 (Number of processors of an enclosing cb-polytop). *If N (N') is the number of processors of the enclosing cb-polytop of the processor space resulting after application of the allocation function defined in program 1 (of another arbitrary linear allocation function), then $N \leq N'$.*

Since we are interested in several allocation functions we replace constraint (4.1) in program 1 for $j = 1$ by $\mathbf{s}_1^T \mathbf{u}_l \neq 0$, $1 \leq l < i$, in order to determine the i-th allocation function, where $\mathbf{u}_l$ are the projection vectors of the previous determined allocation functions.

Example 1.
We consider a part of the GSM speech codec system as example. The considered algorithm consists of two equations.

$$\begin{aligned} &\text{I.} \quad y_1[i,k] = y_1[i-1,k] + r[i]y_2[i-1,k-1], \quad (i,k)^T \in \mathcal{I}, \\ &\text{II.} \quad y_2[i,k] = y_2[i-1,k-1] + r[i]y_1[i-1,k], \quad (i,k)^T \in \mathcal{I}, \end{aligned}$$

$$\mathcal{I} = \{(i,k)^T \mid 1 \leq i \leq 8, 1 \leq k \leq 120\}.$$

The index space with the data dependencies as well as the reduced dependence graph are depicted in Fig. 1.

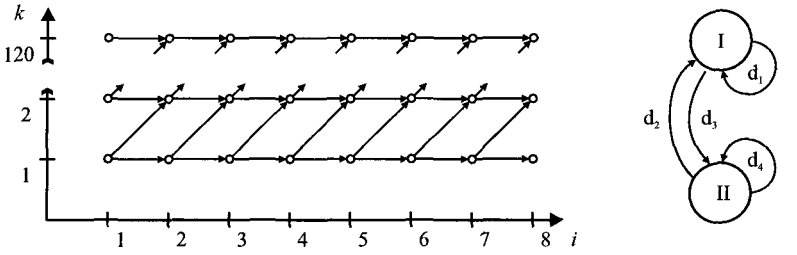

Fig. 1. *Index spaces with data dependencies and reduced dependence graph*

The dependence vectors are: $\text{I} \rightarrow \text{I} : \mathbf{d}_1 = (1,0)^T$, $\text{II} \rightarrow \text{I} : \mathbf{d}_2 = (1,1)^T$, $\text{I} \rightarrow \text{II} : \mathbf{d}_3 = (1,0)^T$, $\text{II} \rightarrow \text{II} : \mathbf{d}_4 = (1,1)^T$. The set of vertices $\mathcal{W}$ of the index space $\mathcal{I}$ is $\mathcal{W} = \{(1,1),(8,1),(1,120),(8,120)\}$.
Application of program 1 leads to the matrices $\mathbf{S}_1 = (1,0)$ and $\mathbf{S}_2 = (0,1)$, and hence to the projection vectors $\mathbf{u}_1 = (0,1)^T$ and $\mathbf{u}_2 = (1,0)^T$ respectively.

The next section covers the determination of a scheduling function and the processor functionality with respect to each projection vector $\mathbf{u}_i$ computed in program 1.

5 Determination of a Scheduling Function

5.1 General Optimization Problem

In this section we propose an approach to determine a scheduling function as well as a module selection and a conflict free assignment of the modules to the operations of the SURE. Our objective is the minimization of the costs for a hardware implementation of the processor array subject to the condition that a given latency L_g is satisfied.
A scheduling function is determined for each linear allocation function resulting after application of program 1. First, we present a general description of the optimization problem and go into more detail in the next paragraphs.

Program 2 (Determination of a scheduling function)

minimize *chip area C of the processor array*
subject to *latency L of the processor array: $L \leq L_g$*
causal scheduling function
selection of modules
conflict free access to the modules

5.2 Objective Function

The number of processors N_p is given exactly since the allocation function is fixed at this step. In each processor of the fullsize array we have to implement n_l instances of module m_l. The chip area needed to implement the modules of all processor of the fullsize array is therefore $C_f = N_p \sum_{l=1}^{|\mathcal{M}|} n_l c_l$.

Additional chip area is needed to implement (1) registers to store intermediate results, (2) combinatorial logic to control the behaviour of the processors and (3) interconnections between the processors. The costs for the control logic are assumed to be negligible. The number of registers depends on the number of processors as well as the number of data produced in each processor. We approximate the chip area C_r needed to implement the registers in silicon by $C_r = N_p\, m\, c_r$, where c_r is the chip area of one register and m is the number of equations of the SURE. Since the number of processors is independent of the scheduling function and the module selection, C_r can be treated as a constant. An assessment of the effort for the implementation of the interconnections is difficult. Instead of measuring the chip area of interconnections we propose a minimization of the length or a limitation of the radius of the interconnections respectively while determining the allocation function.

As a consequence of the above discussion we conclude that it is sufficient to consider the chip area C_f needed to implement the modules as measure for the hardware costs.

Now, we assume that the admissible latency L_g enables a slow down of the fullsize array. Hence, we can decrease the hardware costs by partitioning the fullsize array. We apply the LSGP-partitioning [4] (tiling) by a factor of K, i.e. each partition contains K processors of the fullsize array. Each partition represents a processor of the resulting processor array. We assume that the partitioning matches exactly the fullsize array and we neglect the additional hardware needed to control the partitions. Hence, we get the hardware costs of the resulting processor array $C_p = C_f/K$. Since the number of processors N_p is fixed it is sufficient to consider the chip area C'_f of only one processor of the fullsize array. Our objective is to minimize $C'_p = C'_f/K$, where $C_p = N_p C'_p$ and $C_f = N_p C'_f$.

5.3 Admissible Latency L_g

The latency L_f of the fullsize array is given by $L_f = t^{max} - t^{min}$, where

$$t^{min} \leq \boldsymbol{\tau}^T \mathbf{w} + t_i \leq t^{max} - d_{m(i),i}, \quad \forall \mathbf{w} \in \mathcal{W}, \quad 1 \leq i \leq m,$$

where $t^{min}, t^{max} \in \mathbb{Z}$ and $\mathcal{W}$ is the set of vertices $\mathbf{w}$ of the index space $\mathcal{I}$. The consideration of only the vertices of $\mathcal{I}$ is justified by the fact that linear functions have their extreme values at extreme points of a convex space [10]. The relaxation to integer values has only small effort for sufficiently large index spaces $\mathcal{I}$.

The assumption that $L_g >> L_f$ enables a partitioning of the fullsize array by a factor K, where $KL_f \leq L_g$. This inequality is a sufficient condition since we can presume that a partitioning exists where the latency L of the resulting processor array satisfies $L \leq KL_f$. An intuitive justification yields the opposite case where the distribution of a sequential program to K processors leads to a speed up less or equal to K.

Next, we linearize the constraints $C'_f = KC'_p$ and $KL_f \leq L_g$. The minimal latency L_{min} of a fullsize array is easy to determine by assumption of unlimited resources and assignment of the fastest possible module to each operation of the SURE. Using L_{min} we can limit K by $1 \leq K \leq K_{max} = \lfloor L_g / L_{min} \rfloor$. The lower bound of K can be increased by consideration of the minimal hardware costs for one processor $C'_{min} = \min\{\sum_{l=1}^{|\mathcal{M}|} n_l c_l\}$ satisfying $\exists m_l \in \mathcal{M}_i . n_l \geq 1,\ 1 \leq i \leq m$.

The application of a resource constraint scheduling [13] with the modules leading to C'_{min} yields L_{max} and $K_{min} = \max\{1, \lfloor L_g / L_{max} \rfloor\}$.

The introduction of $K_{max} - K_{min} + 1$ binary variables $\gamma_j \in \{0, 1\}$ enables an equivalent formulation of the constraints $C'_f = KC'_p$ and $KL_f \leq L_g$ as follows:

$$\begin{aligned} & C'_f \leq jC'_p + (1 - \gamma_j)C'_{max}, && K_{min} \leq j \leq K_{max}, \\ & jL_g \geq jK_{min}L_f + \gamma_j(j - K_{min})L_g, && K_{min} \leq j \leq K_{max}, \\ & \sum_{j=K_{min}}^{K_{max}} \gamma_j = 1, \end{aligned} \tag{5}$$

where $C'_{max} = \sum_{l=1}^{|\mathcal{M}|} n_l^{max} c_l$, and n_l^{max} is the maximal number of instances of module m_l which can be implemented in one processor of the fullsize array.

A reduction of the number of variables γ_i from $K_{max} - K_{min} + 1$ to $\lceil (K_{max} - K_{min} + 1)/z \rceil$, $z \in \mathbb{Z}$, is possible by solving the optimization problem iteratively. In the first iteration only such j, $K_{min} \leq j \leq K_{max}$, are considered that satisfy $j \mod z = a$, where $a = K_{min} \mod z$. The second iteration of the optimization problem is solved for $j_{max} - z \leq j \leq j_{max} + z$, where j_{max} is the solution of the first iteration.

5.4 Causality Constraint

In order to ensure a valid partial order or the equations preserving the data dependencies the scheduling function has to satisfy the causality constraint.

Definition 5 (Causality constraint). *A scheduling function $\tau_i(\mathbf{i}) = \boldsymbol{\tau}^T \mathbf{i} + t_i$ has to satisfy the following constraint:*

$$\boldsymbol{\tau}^T \mathbf{d}(e) + t_{\delta(e)} - t_{\sigma(e)} \geq d_{m(\sigma(e)),\sigma(e)}, \quad \forall e \in \mathcal{E}. \tag{6}$$

5.5 Module Selection and Prevention of Access Conflicts

The assignment of a module to the operation F_j is described by $|\mathcal{M}_j|$ binary variables $r_j^l \in \{0,1\}$, where $\sum_{m_l \in \mathcal{M}_j} r_j^l = 1$ and $r_j^k = 1 \leftrightarrow m(j) = m_k$.
The following resource constraint prevents access conflicts to the module.

Definition 6 (Resource constraint). *For a given projection vector* $\mathbf{u}$ *a uniform affine scheduling function* $\tau_i(\mathbf{i}) = \boldsymbol{\tau}^T\mathbf{i} + t_i$ *has to satisfy the following constraint:*

$$\left.\begin{array}{r} (t_j \bmod \lambda) - (t_k \bmod \lambda) \geq o_{m(j),k}, \\ \lambda - (t_j \bmod \lambda) + (t_k \bmod \lambda) \geq o_{m(j),j}, \end{array}\right\} \text{if } (t_j \bmod \lambda) > (t_k \bmod \lambda),$$
$$\left.\begin{array}{r} (t_k \bmod \lambda) - (t_j \bmod \lambda) \geq o_{m(j),j}, \\ \lambda - (t_k \bmod \lambda) + (t_j \bmod \lambda) \geq o_{m(j),k}, \end{array}\right\} \text{if } (t_j \bmod \lambda) \leq (t_k \bmod \lambda), \tag{7}$$

for all $j \neq k$, $1 \leq j,k \leq m$, *with* $m(j) = m(k)$ *and* $u_j = u_k$, *where* $\lambda = |\boldsymbol{\tau}^T\mathbf{u}|$.

We refer to [2] for an explanation and a linearization of (7).

5.6 Result of the Optimization Problem

The result of the optimization problem in program 2 is a set of modules have to be implemented in each processor and the number of processors of the fullsize array building one partition and hence one processor of the resulting processor array. Program 2 is solved for each projection vector computed in program 1. Then we select the projection vector $\mathbf{u}_i$ leading to minimal hardware costs $C = C_p + C_r = N_p(\frac{1}{K}\sum_{i=l}^{|\mathcal{M}|} n_l c_l + m\, c_r)$.

Example 2. (Continuation of example 1)
The admissible latency L_g is measured in clock cycles and supposed to be $L_g = 1200$. The considered set of modules is listed in table 1. We solve the

Table 1. Set of modules

	operation	evaluation time d_i in clock cycles	time offset o_i in clock cycles	chip area c_i normalized
m_1	add/mult	3	3	297
m_2	add/mult	4	4	169
m_3	add/mult	8	8	44
m_r	reg			1

optimization problem of program 2 separately for the projection vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ determined in program 1. In order to justify our approach we present some interesting solutions in table 2 instead of only the best one given by the solver of the optimization problem.

Table 2. Results of the optimization problem

projection vector	N_p	$C_r = N_p\, m\, c_r$	module selection	K	$C_p = \frac{N_p}{K} \sum_{i=1}^{\lvert\mathcal{M}\rvert} n_i c_i$	$C = C_r + C_p$
$\mathbf{u}_1 = (0,1)^T$	8	16	$1 \times m_3$	1	704	720
"	8	16	$1 \times m_2$	1	1352	1368
"	8	16	$2 \times m_1$	3	1584	1600
$\mathbf{u}_2 = (1,0)^T$	120	240	$1 \times m_3$	9	586	826
"	120	240	$2 \times m_2$	37	1096	1336
"	120	240	$1 \times m_1$	25	1425	1665

Minimal hardware costs occur by using projection vector $\mathbf{u}_1$ and implementing one instance of module m_3 in each processor of the resulting processor array. Finally, we want to give some short notes to the computational effort of our example. The worst case is program 2 with respect to the projection vector $\mathbf{u}_1$. The program consists of 168 constraints with 51 binary, 14 integer and one rational variable. The solution takes 11.3 seconds of CPU time on a SUN SPARC Station 10.

5.7 Selection of Projection Vectors

An alternative approach to the separate computation of a scheduling function for each projection vector consists in consideration of the projection vectors $\mathbf{u}_i$ as parameters in program 2. We assume that the iteration interval $\lambda \leq \lambda_{max}$. A linearization of the constraint $\lambda = |\boldsymbol{\tau}^T \mathbf{u}|$ is achievable using four inequalities and one binary variable v.

$$\begin{aligned} \lambda - 2v\lambda_{max} &\leq \boldsymbol{\tau}^T \mathbf{u} \leq \lambda, \quad \lambda \in \mathbb{Z}, \\ \lambda - 2(1-v)\lambda_{max} &\leq -\boldsymbol{\tau}^T \mathbf{u} \leq \lambda, \quad v \in \{0,1\}. \end{aligned} \tag{8}$$

Suppose, that we have to select one of P projection vectors $\mathbf{u}_i$, $1 \leq i \leq P$. Using P binary variables α_i, $1 \leq i \leq P$, we replace (8) by:

$$\begin{aligned} \lambda - 2v\lambda_{max} - 2(1-\alpha_i)\lambda_{max} &\leq \boldsymbol{\tau}^T \mathbf{u}_i \leq \lambda + (1-\alpha_i)\lambda_{max}, \quad 1 \leq i \leq P, \\ \lambda - 2(1-v)\lambda_{max} - 2(1-\alpha_i)\lambda_{max} &\leq -\boldsymbol{\tau}^T \mathbf{u}_i \leq \lambda + (1-\alpha_i)\lambda_{max}, \quad v \in \{0,1\}, \end{aligned}$$

$$\sum_{i=1}^{P} \alpha_i = 1, \quad \alpha_i \in \{0,1\}, \quad 1 \leq i \leq P.$$

The projection vector $\mathbf{u}_i$ is selected if $\alpha_i = 1$.
Furthermore, we have to include the hardware costs for the registers. The objective function is changed to minimize the hardware costs C of the resulting processor array. New constraints have to be introduced to take the different number of processors N_p^i of the fullsize arrays with respect to the projection vectors $\mathbf{u}_i$ into account:

$$C \geq N_p^i (C_p' + m\, c_r) - (1-\alpha_i) N_p^i (C_{max}' + m\, c_r), \quad 1 \leq i \leq P.$$

We do not recommend this approach since the condition of the integer linear program deteriorates strongly.

6 Conclusion

The presented approach is suitable to derive cost minimal processor arrays for algorithms with the requirement of an admissible latency. The arising optimization problems are given in a linearized form which permits the use of standard packages to solve the problems.
Possible extensions to our approach are the inclusion of the power consumption and the inclusion of a rough approximation of the effort needed to implement the interconnections.

References

1. A. Darte, Y. Robert: "Constructive Methods for Scheduling Uniform Loop Nests", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 8, pp. 814-822, 1994
2. D. Fimmel, R. Merker: "Determination of the Processor Functionality in the Design of Processor Arrays", *Proc. Int. Conf. on Application-Specific Systems, Architectures and Processors*, pp. 199-208, Zürich, 1997
3. D. Fimmel, R. Merker: "Determination of an Optimal Processor Allocation in the Design of Massively Parallel Processor Arrays", *Proc. Int. Conf. on Algorithms and Parallel Processing*, pp. 309-322, Melbourne, 1997
4. K. Jainandunsing: "Optimal Partitioning Scheme for Wavefront/Systolic Array Processors", *IEEE Proc. Symp. on Circuits and Systems*, 1986
5. R.M. Karp, R.E. Miller, S. Winograd: "The organization of computations for uniform recurrence equations", *J. of the ACM*, vol.14, pp. 563-590, 1967
6. D.I. Moldovan: "On the Design of Algorithms for VLSI Systolic Arrays", *Proceedings of the IEEE*, pp. 113-120, January 1983
7. P. Quinton: "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations", *IEEE 11-th Int. Symp. on Computer Architecture*, Ann Arbor, pp. 208-214, 1984
8. S.K. Rao: "Regular Iterative Algorithms and their Implementations on Processor Arrays", *PhD thesis*, Stanford University, 1985
9. J. Rossel, F. Catthoor, H. De Man: "Extension to Linear Mapping for Regular Arrays with Complex Processing Elements", *Proc. Int. Conf. on Application-Specific Systems, Architectures and Processors*, pp. 156-167, Princeton, 1990
10. A. Schrijver: *Theory of Linear and Integer Programming*, John Wiley & Sons, New York, 1986
11. A. Schubert, R. Merker: "Systolization of Recursive Algorithms with DESA", in *Proc. 5th Int. Workshop Parcella '90*, Mathematical Research, G. Wolf, T. Legendi, U. Schendel (eds.), vol. 2, Akademie-Verlag Berlin, 1990, pp. 267-276, 1994
12. J. Teich: "A Compiler for Application-Specific Processor Arrays", *PhD thesis*, Univ. of Saarland, Verlag Shaker, Aachen, 1993
13. L. Thiele: "Resource Constraint Scheduling of Uniform Algorithms", *Int. Journal on VLSI and Signal Processing*, Vol. 10, pp. 295-310, 1995
14. Y. Wong, J.M. Delosme: "Optimal Systolic Implementation of n-dimensional Recurrences", *Proc. ICCD*, pp. 618-621, 1985

15. Y. Wong, J.M. Delosme: "Optimization of Processor Count for Systolic Arrays", *Research Report* YALEU/DCS/RR-697, Yale Univ., 1989
16. X. Zhong, S. Rajopadhye, I. Wong: "Systematic Generation of Linear Allocation Functions in Systolic Array Design", *J. of VLSI Signal Processing*, Vol. 4, pp. 279-293, 1992