Journal of VLSI Signal Processing Systems, 15, 5–21 (1997)

Partitioning Processor Arrays under Resource Constraints

JÜRGEN TEICH AND LOTHAR THIELE
Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), CH-8092 Zurich, Switzerland
{thiele,teich}@tik.ee.ethz.ch

LEE Z. ZHANG
Institute on Microelectronics, University of Saarland, D-66041 Saarbrücken, Germany

Received 1996; Revised 1997

Editor: S. Y. Kung

Abstract. A single *integer linear programming* model for optimally scheduling partitioned regular algorithms is presented. The herein presented methodology differs from existing methods in the following capabilities: 1) Not only constraints on the number of available processors and communication capabilities are taken into account, but also *local memories* and constraints on the *size of available memories.* 2) Different types of processors can be handled. 3) The size of the optimization model (number of integer variables) is independent of the size of the tiles to be executed. Hence, 4) the number of integer variables in the optimization model is greatly reduced such that problems of relevant size can be solved in practical execution time.

Keywords: systolic array, partitioning, integer linear programming, scheduling

1. Introduction

This paper deals with techniques for mapping partitioned regular algorithms onto massively parallel processor arrays that are realized in VLSI. These architectures are distinguished by the properties of distributed memory and computing power, simple and regular communication, locality in time and space, and intensive pipelining and parallelism. They are considered the most promising architectures for computationally intensive applications.

The most important problems to be solved in the synthesis process include *scheduling* where the operations in the algorithmic description are mapped onto discrete time steps, and *binding* where the operations and variables in the algorithmic description are mapped onto shared hardware resources.

In the realm of regular processor arrays, this is achieved by definition of a linear space-time mapping. Later, *partitioning* methods have been proposed for mapping an arbitrary size regular algorithm onto a processor array with a restricted number of processing elements and dimension. This problem is called *partitioning.* The aim is to establish a balance between local memory, I/O-rate and computation rate, see e.g., [20, 15].

However, while the emphasis of *partitioning* has generally been on restricted resources on the *array level* (i.e., number of processing elements) (*regu-*

lar resource constraints), typical problems like a constrained number of functional resources, communication resources and memory resources inside the processing elements (so called *irregular resource constraints*) have been ignored. For example, the number of available function units in a processing element is not sufficient to execute the operations assigned to them concurrently.

As a result, a new research direction on extending array mapping methodologies can be recognized in the *fusion* of mapping techniques for regular processor arrays and high-level synthesis tasks, see, e.g., [20, 4, 2, 18, 6, 5]. Here is a short overview of recent work in this direction:

Except of special cases, resource constrained scheduling problems are known to be *NP-complete*, see e.g., [3, 7]. Hence, numerous heuristic methods have been proposed in order to find a compromise between the runtime of the scheduling algorithms and the quality of the solution [11]. Moreover, it is possible to model resource constrained scheduling problems by an *Integer Linear Program* (ILP) [9, 1, 8].

Now, the work described in [18] was the first to establish a connection between partitioning and high-level synthesis techniques by providing a single ILP-based optimization model for partitioning with a constrained number of irregular functional resources.

In [10] and [2], communication constraints between processing elements have been included in ILP models. However, as mentioned in [6], the ILP formulation as described in [2] can only be used in a very restrictive instance of the scheduling problem. In [6], the ILP-model proposed in [18] has been extended to include communication delays and communication constraints. Unfortunately, the size of this model (measured in the number of integer variables) is proportional to the number of points inside a tile which is executed sequentially by one processing element. This is a severe restriction, because ILP solving techniques, such as *branch and bound* techniques tend to be applicable only to small size problems [12] because they suffer from long CPU-times.

Furthermore, constraints on local and global memory have so far not been considered in any existing approach.

In order to overcome these major drawbacks, we will present several extensions to the approaches described in [18], [2], and [6] including:

- *Memory design*:
  Based on the results presented in [1], we will also incorporate *memory* constraints into our model. The model will be extended to include the effects of local and global memory, the size and the optimization of memory through reuse. This is achieved by lifetime analysis of variables and reformulating the objective function of our ILP model to include memory optimization. Hence, not only the number of processing elements, but also the available function units, the total amount of local memory, the restricted communication capabilities between processing elements, as well as the limited communication between function units and local memory can be included in the ILP formulation.
- *Multiprojection by Loop Scanning*:
  For specifying the order of execution of operations at index points inside a tile, we define a *loop scanning matrix* P that describes a *multiprojection*. From this matrix, a number of inequalities to be added to the ILP can be easily generated.
- *Reduction of the problem size*:
  Contrary to [6], the number of integer variables in the final ILP model is not dependent on the number of index points inside a tile, but only on its dimension. This enables us to greatly reduce the size of our optimization model, and to solve the optimization problem for instances with large index spaces while including memory optimization.

In Section 2, we will introduce the class of algorithms we are dealing with. For reasons of clarity, we will also briefly discuss the partitioning schemes represented in [15]. In Section 3, a new ILP formulation for scheduling of partitioned regular algorithms with limited resources will be presented. Some experimental results of a *Finite Impulse Response* (FIR) digital filter and a *Wave*

Digital Filter (WDF) are reported in Section 4, followed by concluding remarks.

2. Regular Algorithms and Partitioning

In order to explain the background of our optimization methodology, we review the definitions of regular algorithms and of partitioning.

2.1. Regular Algorithms

The class of algorithms we are using in our scheduling problem is the same as that dealt with in [18]: Extensions of *Regular Iterative Algorithms* [13] which are similar to the class of *piecewise regular algorithms*, (see [16, 17]), however, with a more complex index domain, namely a *linearly bounded lattice* [16, 14] which will be defined next. In particular, this generalization is necessary if the regular algorithm is obtained as a result of the partitioning method as introduced in [15].

Definition 1. [Regular Algorithm] A regular algorithm consists of a set of N quantified indexed equations $S_i[I]$

$$S_1[I] \ \cdots \ S_i[I] \ \cdots \ S_N[I] \qquad \forall \ I \in \mathbf{I} \qquad (1)$$

where $I \in \mathbf{Z}^s$ is the index vector and the index domain $\mathbf{I} \subseteq \mathbf{Z}^s$ is a linearly bounded lattice that is defined below. Each equation $S_i[I]$ is of the form

$$x_i[I] = \mathcal{F}_i(\cdots x_j[I - d_{ji}] \cdots) \qquad (2)$$

where $x_i[I]$ are indexed variables, $\mathcal{F}_i$ are arbitrary functions and $d_{ji} \in \mathbf{Z}^s$ are constant dependence vectors.

Linearly bounded lattices, see [16, 14], are defined as follows [1]:

Definition 2. [Linearly Bounded Lattice] A linearly bounded lattice is an index space of the form

$$\mathbf{I} = \{I : \ I = L\kappa + m \ \wedge \ A\kappa \geq b \ \wedge \ \kappa \in \mathbf{Z}^l\} \quad (3)$$

where $L \in \mathbf{Z}^{s \times l}$, $m \in \mathbf{Z}^s$, $A \in \mathbf{Z}^{m \times l}$ and $b \in \mathbf{Z}^m$.

Obviously, $\{\kappa : \ A\kappa \geq b \ \wedge \ \kappa \in \mathbf{Z}^l\}$ defines the set of all integer vectors within a polytope. The integral points in this polytope are mapped to index vectors I using an affine function, i.e., $I = L\kappa + m$. For reasons of simplicity, we assume L and m to be the unity matrix and the zero vector respectively,[2] i.e., the index space can be written as

$$\mathbf{I} = \{I \in \mathbf{Z}^s \ : \ AI \geq b\} \qquad (4)$$

2.2. Partitioning Schemes for Regular Algorithms

The partitioning procedure consists of three major parts, see [15].

- *Tiling*:
  The given iteration space $\mathbf{I}$ is decomposed into the direct sum of two index sets $\mathbf{J}$ and $\mathbf{K}$ such that $\mathbf{I} \subseteq \mathbf{J} \oplus \mathbf{K}$ [3]. The index set $\mathbf{J} \subseteq \mathbf{Z}^s$ represents one of the tiles and the set $\mathbf{K} \subseteq \mathbf{Z}^s$ accounts for their regular repetition. In order not to destroy the regularity of the algorithm, we only consider regular tiles $\mathbf{J}$, i.e., they are spanned by s linearly independent tiling directions $p_i \in \mathbf{Z}^s$, $i = 1, \cdots, s$, combined in a tiling matrix $P = (p_1, \cdots, p_s)$. An example of a tiling of an iteration space is shown in Fig. 1. The optimal choice of the tiling matrix is not considered in the present paper.
- *Embedding*:
  The second step is to *embed* all variables in a higher dimensional index space according to the chosen tiling. In particular, some new dependence vectors have to be introduced if dependencies cross different tiles. Then the embedding transformation leads to a set of equivalent equations of (2)

$$x_i[J, K] = \mathcal{F}_i(\cdots x'_j[J, K] \cdots)$$

for all $J + K \in \mathbf{I}$ and

$$x'_j[J, K] = x_j[J - d_{ji} - P\theta, K + P\theta] \quad \text{if } J - d_{ji} - P\theta \in \mathbf{J} \qquad (5)$$

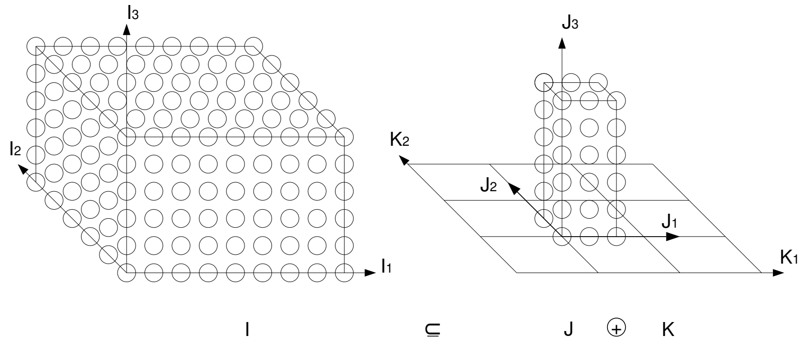

Fig. 1. Example of tiling an iteration space $\mathbf{I}$ according to $\mathbf{I} \subseteq \mathbf{J} \oplus \mathbf{K}$

for all $J+K \in \mathbf{I}$ and for all vectors $\theta \in \Theta$ and [4]

$$\Theta = \{\theta \in \mathbf{Z}^s \ : \ -e < \theta + P^{-1} d_{ji} < e\} \quad (6)$$

- *Multiprojection*: Affine transformations of the index space after embedding can be used to permute indices, normalize lattices, etc. Using these, all known partitioning schemes can be realized, see e.g., [15]. Therefore, we assume without loss of generality that the index points inside a tile $\mathbf{J}$ are executed sequentially by a processing element in a linear order and that $\mathbf{K}$ is associated with the processor space of the realization, i.e., to each index point in $\mathbf{K}$ there corresponds one processing element. This scheme is often called Locally Sequential Globally Parallel (LSGP). The multiprojection can be described as follows:

$$I' = \begin{bmatrix} t \\ p \end{bmatrix} = \begin{bmatrix} \lambda_J J + \lambda_K K \\ K \end{bmatrix} \quad (7)$$

where $t \in \mathbf{Z}$, $\lambda_J \in \mathbf{Z}^s$, $\lambda_K \in \mathbf{Z}^s$, such that $\lambda_J J + \lambda_K K$ is unique for all $J \in \mathbf{J}$, $K \in \mathbf{K}$.

The above described partitioning scheme will be explained in the following example.

Example 1. The equation

$$Y_{i_2} = \sum_{i_1=0}^{N-1} A_{i_1} \cdot U_{i_2-i_1} \quad (8)$$

with $i_2 \in \{0, \cdots, (T-1)\}$ is the difference equation of a finite impulse response (FIR)-digital filter. The functionality of this equation can be captured by the following regular algorithm after carrying out a series of affine transformations [15].

$$\begin{aligned} y[i_1,i_2] &= y[i_1-1,i_2] + x[i_1,i_2] && (9)\\ x[i_1,i_2] &= a[i_1,i_2] * u[i_1,i_2] && (10)\\ a[i_1,i_2] &= a[i_1,i_2-1] && (11)\\ u[i_1,i_2] &= u[i_1-1,i_2-1] && (12) \end{aligned}$$

where the index domain, $\mathbf{I} = \{(i_1,i_2) : \ 0 \le i_1 \le N-1 \ \wedge \ 0 \le i_2 \le T-1\}$, is a linearly bounded lattice with

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ 0 \\ -N+1 \\ -T+1 \end{bmatrix}$$

For $N = 4$ and $T = 6$ (see Fig. 3(a)) the tiling matrix

$$P = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$$

is chosen. The subsequent embedding of the equations (9)...(12), leads to the following regular algorithm (15)...(23).

These equations hold for all $J \in \mathbf{J}$ and $K \in \mathbf{K}$ with

$$\begin{aligned} \mathbf{J} &= \{(j_1,j_2) \ : \ 0 \le j_1 < 2 \ \wedge \ 0 \le j_2 \le 1\} && (13)\\ \mathbf{K} &= \{(k_1,k_2) \ : \ 0 \le k_1 < 2 \ \wedge \ 0 \le k_2 \le 2\} && (14) \end{aligned}$$

$$
\begin{aligned}
y[j_1,j_2,k_1,k_2] &= y[i_1-1,i_2,k_1,k_2]+x[i_1,i_2,k_1,k_2] \quad \text{if } j_1\geq 1 && (15)\\
y[j_1,j_2,k_1,k_2] &= y[i_1+1,i_2,k_1-1,k_2]+x[i_1,i_2,k_1,k_2] \quad \text{if } j_1=0 \ \wedge \ k_1\geq 1 && (16)\\
x[j_1,j_2,k_1,k_2] &= a[j_1,j_2,k_1,k_2]*u[j_1,j_2,k_1,k_2] && (17)\\
a[j_1,j_2,k_1,k_2] &= a[j_1,j_2-1,k_1,k_2] \quad \text{if } j_2\geq 1 && (18)\\
a[j_1,j_2,k_1,k_2] &= a[j_1,j_2+1,k_1,k_2-1] \quad \text{if } j_2=0 \ \wedge \ k_2\geq 1 && (19)\\
u[j_1,j_2,k_1,k_2] &= u[j_1-1,j_2-1,k_1,k_2] \quad \text{if } j_1\geq 1 \ \wedge \ j_2\geq 1 && (20)\\
u[j_1,j_2,k_1,k_2] &= u[j_1+1,j_2+1,k_1-1,k_2-1] \quad \text{if } j_1=0 \ \wedge \ j_2=0 && (21)\\
u[j_1,j_2,k_1,k_2] &= u[j_1+1,j_2-1,k_1-1,k_2] \quad \text{if } j_1=0 \ \wedge \ j_2\geq 1 && (22)\\
u[j_1,j_2,k_1,k_2] &= u[j_1-1,j_2+1,k_1,k_2-1] \quad \text{if } j_1\geq 1 \ \wedge \ j_2=0 && (23)
\end{aligned}
$$

In order to describe the scheduling and allocation procedures, it is useful to define the well known *Reduced Dependence Graph* (RDG).

Definition 3. [RDG] The reduced dependence graph $RDG=(V_D,E_D,D)$, associated to a partitioned regular algorithm in the sets $\mathbf{J}$ and $\mathbf{K}$ as defined above is defined as follows: To each variable $x_i[J,K]$, a node $v_i\in V_D$ is associated. An edge $(v_i,v_j)\in E_D$ with distance vector $d_{ij}=(d_{ij}^J\ d_{ij}^K)^T$ and $d_{ij}\in\mathbf{Z}^{2s\times 1}$ appears if the variable $x_j[J,K]$ directly depends on $x_i[J-d_{ij}^J,K-d_{ij}^K]$ via the dependence vector d_{ij}.

The RDG can also be represented algebraically. To this end we will use the node-arc *incidence* matrix $C\in\{0,1,-1\}^{|V_D|\times|E_D|}$ [5] and the *distance* matrix $D\in\mathbf{Z}^{2s\times|E_D|}$ where each column vector is associated with an edge (i,j) and corresponding entries being d_{ij}. The following example demonstrates the notations of RDG as discussed above.

Example 2. Again we use the algorithm and the partitioning described in the previous example (see Fig. 3). In the corresponding RDG shown in Fig. 2, the data dependencies are numbered in the order of appearance in the algorithm (15), (16), (18)...(23). They are represented by edges $(y,y)^1$, $(y,y)^2$, $(a,a)^1$, $(a,a)^2$, and $(u,u)^1...(u,u)^4$,

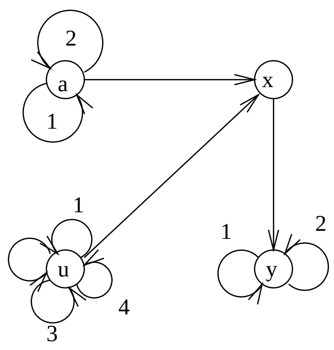

Fig. 2. Reduced dependence graph of the FIR-filter

respectively. Corresponding incidence and distance matrices are

$$
C = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
$$
$$
D = \begin{bmatrix} 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & -1 & 1\\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}
$$

where the columns are ordered according to (a,x), (u,x), (x,y), $(y,y)^1$, $(y,y)^2$, $(a,a)^1$, $(a,a)^2$, and $(u,u)^1...(u,u)^4$, respectively.

3. Scheduling and Binding

Now, the construction of an integer linear program is described. Its solution yields a time and/or re-

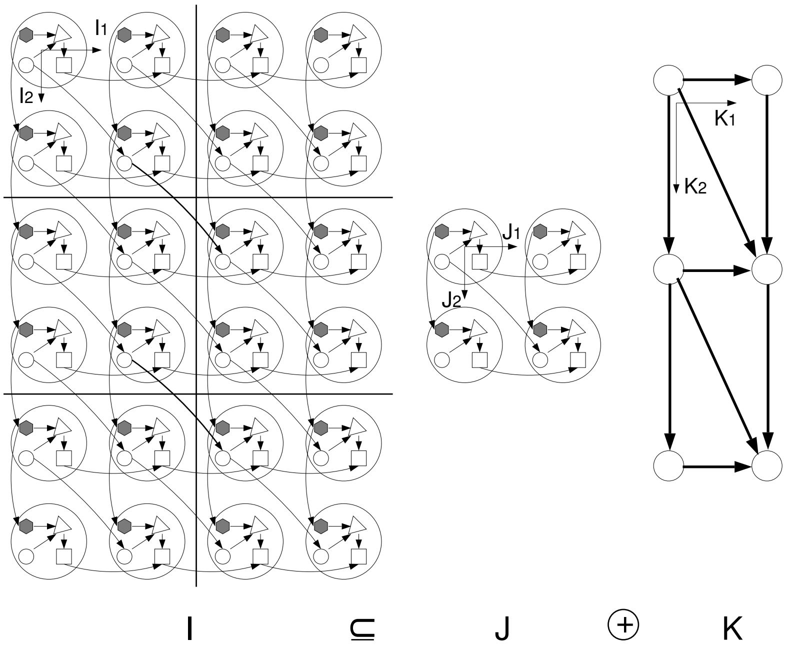

Fig. 3. Tiled dependence graph of the FIR-filter algorithm

source optimal scheduling and binding of operations.

As is typical for VLSI processor array design, a *linear scheduling* is applied to the partitioned regular algorithm with the index space $\mathbf{I} \subseteq \mathbf{J} \oplus \mathbf{K}$. The calculation of a variable (or an operation) $x_j[J,K]$ for $J \in \mathbf{J}$ and $K \in \mathbf{K}$ is finished at time step

$$t_j(J,K) = \lambda_K \cdot K + \lambda_J \cdot J + \Gamma_j \qquad (24)$$

where $t_j(J,K) \in \mathbf{Z}$, $\Gamma_j \in \mathbf{Z}$, and $\lambda_K \in \mathbf{Z}^s$, $\lambda_J \in \mathbf{Z}^s$ are linear scheduling vectors.

3.1. Objective function

The goal of the optimization is to minimize the total evaluation time which is approximated by

$$\begin{array}{lll} max & \lambda_K(K_2 - K_1) + \lambda_J(J_2 - J_1) & \\ subject\ to & A_K K_1 \geq b_K & A_J J_1 \geq b_J \\ & A_K K_2 \geq b_K & A_J J_2 \geq b_J \end{array}$$

where the inequalities $A_K K \geq b_K$ and $A_J J \geq b_J$ specify the index space of $\mathbf{K}$ and $\mathbf{J}$, respectively. The above problem can be transformed into following form using the duality theorem of linear programming:

$$\begin{array}{lll} min_{\lambda_j,\lambda_K} & -(y_1+y_2)b_K - (y_3+y_4)b_J & \\ subject\ to & y_1 A_K = \lambda_K & y_1 \geq 0 \\ & y_2 A_K = -\lambda_K & y_2 \geq 0 \\ & y_3 A_J = \lambda_J & y_3 \geq 0 \\ & y_4 A_J = -\lambda_J & y_4 \geq 0 \end{array} \qquad (25)$$

Clearly, the duality theorem holds for linear programs only. Consequently, the total evaluation time is approximated only. For bounding the error which is introduced in the case of additional integral constraints, see e.g. [4, 5] and the references herein.

3.2. Dependence constraints

Let us suppose that an operation j directly depends on an operation i via the distance d_{ij}. In this case there is an edge $(v_i, v_j) \in E_D$ in the reduced dependence graph with distance vector d_{ij}. The result of operation j is available not earlier than the corresponding data sending operation i

has finished plus the *evaluation time* w_j associated to the node $v_j \in V_D$.

The corresponding constraint

$$t_j(J_j, K_j) - t_i(J_i, K_i) \geq w_j \qquad \forall (v_i, v_j) \in E_D$$

directly leads to

$$(\lambda_J \ \ \lambda_K) d_{ij} + \Gamma_j - \Gamma_i \geq w_j \qquad \forall (v_i, v_j) \in E_D \tag{26}$$

3.3. Sequentialization constraints

The next set of constraints guarantees that the operations within the tile are executed sequentially. To this end, it is useful to define the iteration interval as follows:

Definition 4. The iteration interval δ of a scheduled regular algorithm is the distance in time between two successive executions of a node of the reduced dependence graph on one processing element.

With $t_j(J, K) = \lambda_K \cdot K + \lambda_J \cdot J + \Gamma_j$, the distance in time between two operations corresponding to node v_j and with index points J_1 and J_2 is $t_j(J_1, K) - t_j(J_2, K) = \lambda_J(J_1 - J_2)$. As we require a periodic execution within one tile with corresponding period δ we have

$$\lambda_J = \delta \tilde{\lambda}_J \qquad \tilde{\lambda}_J \text{ integral} \tag{27}$$

In addition, we must require that the operations corresponding to a node v_i are executed sequentially within a tile $\mathbf{J}$:

$$\lambda_J(J_1 - J_2) \neq 0 \qquad \forall J_1, J_2 \in \mathbf{J} \quad J_1 \neq J_2 \tag{28}$$

In order to incorporate this condition into the system of linear inequalities, a loop matrix $P = \begin{pmatrix} p_1 & \cdots & p_s \end{pmatrix} \in \mathbf{Z}^{s \times s}$ will be defined which determines the sequential ordering of index points $J \in \mathbf{J}$ as follows: operations corresponding to index points in direction of p_1 are executed one after the other, operations in direction of p_2 are separated in time by blocks of operations in direction p_1 and so on. The ordering is similar to a sequential nested loop program where a loop index i_k corresponds to iterations in the direction of p_k. The inner loop index is i_1, the outer index is i_s. Moreover, P will be chosen such that the space spanned by the column vectors contains $\mathbf{J}$:

$$\mathbf{J} \subseteq \{J \in \mathbf{Z}^s \ : \ J = P\kappa \wedge 0 \ \leq \ \kappa < e\}$$

where e denotes the unit vector $\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^t$. Note, that the matrix P not only defines the loop directions but also the loop bounds. For example, $P = 5$ leads to $J = \{0, 1, 2, 3, 4\}$.

The following theorem contains the relation between a sequential schedule and the loop matrix. Independently, a similar result has been obtained in [21].

Theorem 1. *The condition (28) is satisfied if the following system of linear inequalities hold:*

$$\lambda_J P \begin{pmatrix} 1/r_{11} & -1 & \cdots & -1 \\ 0 & 1/r_{22} & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/r_{ss} \end{pmatrix} \geq \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \tag{29}$$

The parameters r_{ii} are determined by performing a left hand unimodular transformation of P to the upper triangular matrix

$$R = \begin{pmatrix} r_{11} & * & \cdots & * \\ 0 & r_{22} & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{ss} \end{pmatrix}$$

with $P = CR$, $C \in \mathbf{Z}^{s \times s}$, $|det(C)| = 1$ and $r_{ii} > 0$ for all $1 \leq i \leq s$.

Proof: *It will be shown that (29) leads to (28) or equivalently, to*

$$\lambda_J J \neq 0 \quad \forall J \in \mathbf{Z}^s \ \wedge \ J = P\kappa \ \wedge \ -e < \kappa < e$$

To this end we make the decomposition

$$\begin{aligned} \lambda_J P \kappa &= \lambda_J P \begin{pmatrix} \kappa_1 & \cdots & \kappa_{l-1} & \kappa_l & 0 & \cdots & 0 \end{pmatrix}^T \\ &= \lambda_J P \begin{pmatrix} -\alpha_l & \cdots & -\alpha_l & \alpha_l / r_{ll} & 0 & \cdots & 0 \end{pmatrix} + \\ &\quad \lambda_J P \begin{pmatrix} \kappa_1 + \alpha_l & \cdots & \kappa_{l-1} + \alpha_l & 0 & \cdots & 0 \end{pmatrix} \end{aligned}$$

where we assume that $\kappa_l > 0$ (the case where $\kappa_l < 0$ can be handled in a similar way). Now we will show that the first and second terms are *both greater than zero which than concludes the proof.*

Let us start with the first term. We show that $\alpha_l \in \mathbf{Z}$ and $\alpha \geq 1$. Using the decomposition $P = CR$ where C is unimodular and R has positive diagonal elements r_{ii} we can see that κ_l is an integral multiple of r_{ll} because $J = P\kappa = CR\kappa$ is required to be integral. Hence $\alpha_l \in \mathbf{Z}$ and $\alpha \geq 1$. As the column vector in the first term is now a positive multiple of the lth column in (29), the first term is non-negative.

Now the second term. As $\alpha_l \in \mathbf{Z}$, $\alpha \geq 1$ and $-e < \kappa < e$, the first $l-1$ components of the column vector in the second term are strictly positive. It remains to be shown that $\lambda_J P \begin{pmatrix} u_1 & \cdots & u_{l-1} & 0 & \cdots & 0 \end{pmatrix}^t > 0$ if $u_i > 0$ for all $1 \leq i \leq l-1$. To this end, the vector u is expressed as

$$\begin{pmatrix} u_1 \\ \vdots \\ u_{l-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 1/r_{11} & -1 & \cdots & -1 \\ 0 & 1/r_{22} & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/r_{ss} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_{l-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

It can easily be seen that if $u_i > 0$ for all $1 \leq i \leq l-1$ then $v_i > 0$ for all $1 \leq i \leq l-1$. Now we can write

$$\lambda_J P \begin{pmatrix} u_1 & \cdots & u_{l-1} & 0 & \cdots \end{pmatrix}^t \geq \begin{pmatrix} 1 & 0 & \cdots \end{pmatrix} \begin{pmatrix} v_1 & \cdots & v_{l-1} & 0 & \cdots \end{pmatrix} > 0$$

which concludes the proof. □

Example 3. Fig.4 shows the relation between a chosen loop matrix P, a feasible transformation vector λ_J and the resulting execution ordering for a given tiling space $\mathbf{J}$. In particular, we have $P = \begin{pmatrix} -3 & 3 \\ 3 & 6 \end{pmatrix}$ which leads to $R = \begin{pmatrix} 3 & 6 \\ 0 & 9 \end{pmatrix}$ and $7\lambda_2/10 \leq \lambda_1 \leq \lambda_2 - 1$. In Fig.4, the labels correspond to $\begin{pmatrix} \lambda_1 & \lambda_2 \end{pmatrix} \begin{pmatrix} j_1 \\ j_2 \end{pmatrix}$ with $\lambda = \begin{pmatrix} 3 & 4 \end{pmatrix}$.

So far, the objective function (25) together with the constraints (26), (27) and (29) provide an integer linear program whose solution gives a time optimal schedule with the following property: The operations within a tile $\mathbf{J}$ are executed index point by index point with the iteration interval δ. The execution ordering is given by the loop matrix P. Consequently, it is an *optimal solution for a fixed loop matrix.* Until now, resource constraints *within* each processing element and *between* the processing elements are not taken into account. Each processing element is responsible for executing all the operations corresponding to $v_i \in V_D$ within a tile $\mathbf{J}$. These resource constraints will be the subject of the following sections.

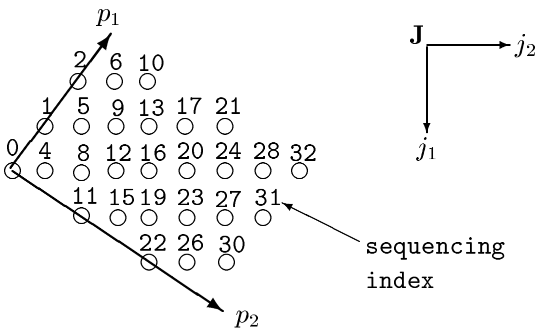

Fig. 4. Relation between a loop matrix and the execution ordering

3.4. Resource constraints

In order to take into account different kinds of resource constraints within a processing element, two kinds of representations are used:

Node-Resource Graph: The node-resource graph represents the fact that a resource is used during the computation time of an operation, i.e. a node in the dependence graph. Resource types may be functional resources such as ALU, multiplier, processor but also a bus or the access to a memory. The graph can also be used to describe the use of memory during a computation: *work memory.*

Edge-Resource Graph: The edge-resource graph specifies the use of resources during the lifetime of an edge in the dependence graph, i.e. the use of a resource between the end of one operation and the end of another operation. For example, this type of graph may describe the amount of memory necessary to transfer data from one operation to the next one: *transfer memory.*

3.4.1. Node-Resource Graph A node-resource graph describes which nodes in the reduced dependence graph can be mapped onto which resources.

Definition 5. A node-resource graph is a bipartite graph $G_N(V_N, E_N)$ where $V_N = V_D \cup V_{NR}$ and

- V_{NR} denotes node-resource types and V_D is the set of nodes of the reduced dependence graph.
- $E_N \subseteq V_D \times V_{NR}$ describes a relation where $e = (v_i, v_k) \in E_N$ means that a resource request from node $v_i \in V_D$ can be satisfied on resources of type $v_k \in V_{NR}$.

Note that if there is more than one edge with head $v_i \in V_D$, the request can be satisfied by one of the tail nodes v_k. This models module selection, for example the situation, that an operation can be implemented either on an application specific or on a general purpose module.

Moreover, to the edges and nodes of the node-resource graph attributes can be associated which describe additional parameters such as computation time of a node, necessary work memory of a node or the available number of resources of a specific type. In particular, for the present paper, the following attributes are used:

- $delay(v_i, v_k)$ describes the execution time in time units of node $v_i \in V_D$ if executed on resource type $v_k \in V_{NR}$.
- $alloc(v_k)$ denotes the number of available resources of type $v_k \in V_{NR}$, for example the number of available multipliers in a processing element.
- $work(v_i, v_k)$ describes the amount of memory needed on $v_k \in V_{NR}$ during a computation $v_i \in V_D$ (work memory).
- $size(v_k)$ denotes the size of the memory $v_k \in V_{NR}$, i.e. the amount of data which can be stored.

Now it remains to be shown how the constraints corresponding to a node-resource graph can be formulated as additional linear inequalities. The main idea has been described in [19] but some generalizations are necessary in order to take into account module selection and memory/bus resources.

Functional Resources: Let us suppose that we have a resource graph whose resource types are functional units such as multipliers, adders or even busses. Then the main concept can be summarized as follows:

- The relative start time Γ_j of each operation in the *RDG* has a lower and upper bound. For example, for operation j corresponding to node $v_j \in V_D$ we have $asap(j) \leq \Gamma_j \leq alap(j)$. These bounds can be obtained from schedules following the *As Soon As Possible* (ASAP) and *As Late As Possible* (ALAP) principles, respectively. Resource constraints within the processing elements are not taken into account. These schedules can be obtained by formulating linear programs which determine the earliest and latest scheduling times of the operations without taking resource constraints into account. This linear program contains the sequentialization and dependency constraints only.
- The relative starting times Γ_j in (24) are represented as the weighted sum of binary variables $\gamma_{j,k,t}$, where $\gamma_{j,k,t} = 1$ denotes that at the relative time instance $t \in \mathbf{Z}$ the operation j has finished its operation on resource type k. Obviously, $t \in T(j) = \{asap(j), asap(j) + 1, \cdots, alap(j)\}$, or $asap(j) \leq t \leq alap(j)$ and we have

$$\Gamma_j = \sum_{v_k \in V_{NR}} \sum_{t \in T(j)} t\gamma_{j,k,t} \quad \forall\, v_j \in V_D \qquad (30)$$

- As each operation can be executed on one resource type only, we need to add the following equalities

$$\sum_{v_k \in V_R : (v_j, v_k) \in E_N} \sum_{t \in T(j)} \gamma_{j,k,t} = 1 \quad \forall\, v_j \in V_D \qquad (31)$$

- The evaluation time corresponding to a node $v_j \in V_D$ can now be determined as

$$w_j = \sum_{v_k \in V_R : (v_j, v_k) \in E_N} \sum_{t \in T(j)} delay(v_j, v_k) \gamma_{j,k,t} \qquad \forall\, v_j \in V_D \tag{32}$$

- The following constraints are necessary and sufficient to prevent more than $alloc(v_k)$ operations being executed by the same type of resources $v_k \in V_{NR}$ at the same control step:

$$\sum_{v_j \in V_D : (v_j, v_k) \in E_N} \sum_{t \in T'(j,k,t')} \gamma_{j,k,t} \leq alloc(v_k) \qquad \forall\, v_k \in V_{NR} \quad \forall\, 0 \leq t' \leq \delta - 1 \tag{33}$$

  with the set of time instances $T'(j,k,t') = \{t \in T(j) : \ t = t' + p + q\delta,\ 0 \leq p \leq delay(v_j, v_k) - 1,\ q \in \mathbf{Z}\}$. The set $T'(j,k,t')$ is introduced for convenience. Note, that δ is the iteration interval, therefore we need a constraint for every time step $0 \leq t' \leq \delta - 1$. For a fixed t' we need to consider the time interval where an operation uses a resource, i.e. $0 \leq p \leq delay(v_j, v_k)$. The variable q takes into account that operations from different iterations overlap.

After adding to the objective function (25) and the constraints (26), (27) and (29) the new restrictions (30, 31, 32, 33), resource constraints for functional units within a processing element have been considered. This method can also be used to take into account bus constraints between functional units as will be seen in the examples in later sections.

The following example serves to explain the notation of a resource graph.

Example 4. Once more, we use the algorithm described by in Example 1. We will suppose that there are two different types of function units available, one the multiplier, and the other the ALU. Therefore, we have $alloc(ALU) = alloc(MUL) = 1$. In the resource graph, four nodes in V_D correspond to a, u, x and y, and two nodes in V_{NR} correspond to the function units multiplier and ALU. There are edges from all nodes in V_D to the node ALU as all operations can be run on the general purpose ALU. Node x can run on either MUL or ALU, see Fig. 5.

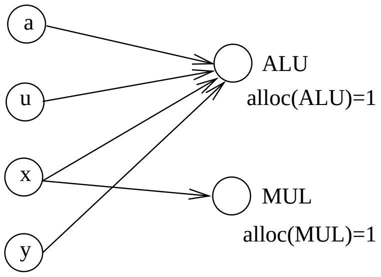

Fig. 5. Resource Graph

Memory Resources: Let us now suppose that we have an additional node-resource graph $G'_N(V'_N, E'_N)$, $V'_N = V_D \cup V'_{NR}$ which models work memory. In particular we suppose that a node $v_i \in V_D$ needs during its evaluation time w_j the amount of $work(v_i, v_l)$ memory on the storage module $v_l \in V'_{NR}$ of size $size(v_l)$. Then there is an edge $(v_i, v_l) \in E'_N$. (For the sake of simplicity we do not assume the selection between different memory element). This constraint can be considered by adding the following inequalities to the integer linear program:

$$\sum_{v_j \in V_D : (v_j, v_l) \in E'_N} work(v_j, v_l) \sum_{v_k \in V_{NR} : (v_j, v_k) \in E_N} \sum_{t \in T'(j,k,t')} \gamma_{j,k,t} \leq size(v_l) \tag{34}$$

$$\forall\, v_l \in V'_{NR} \quad \forall\, 0 \leq t' \leq \delta - 1$$

with the set of time instances $T'(j,k,t') = \{t \in T(j) : \ t = t' + p + q\delta,\ 0 \leq p \leq delay(v_j, v_k) - 1,\ q \in \mathbf{Z}\}$.

3.4.2. Edge-Resource Graph The edge-resource graph models the use of resources during the lifetime of an edge.

Definition 6. An edge-resource graph is a bipartite graph $G_E(V_E, E_E)$ where $V_E = E_D \cup V_{ER}$ and

- V_{ER} denotes edge-resource types and E_D is the set of edges of the reduced dependence graph.
- $E_E \subseteq E_D \times V_{ER}$ describes a relation where $(e_l, v_k) \in E_E$ means that a resource request from edge $e_l \in E_D$ can be realized on resources of type $v_k \in V_{ER}$.

Again, attributes can be associated to the edge-resource graph which describe additional parameters such as necessary transfer memory during data communication related to an edge or the available amount of memory in a memory module. The following attributes will be used:

- $transfer(e_l, v_k)$ describes the amount of memory needed on memory $v_k \in V_{ER}$ for edge $e_l \in E_D$ (transfer memory).
- $size(v_k)$ denotes the size of the memory $v_k \in V_{ER}$, i.e. the amount of data which can be stored.

Now it remains to be shown how the constraints corresponding to the transfer of data can be modeled using an edge-resource graph. To this end the auxiliary variables $z_{l,t}$ are defined with the following interpretation: $z_{l,t} = 1$ for an edge $e_l = (v_i, v_j) \in E_D$ for all relative time instances between the end of operation i and the end of operation j. $z_{l,t} = 0$ holds otherwise.

$$z_{l,t} = \sum_{v_k:(v_i,v_k)\in E_N} \sum_{p=0}^{asap(i)-t} \gamma_{i,k,t-p} - \sum_{v_k:(v_j,v_k)\in E_N} \sum_{p=0}^{asap(j)-t} \gamma_{j,k,t-p} \tag{35}$$

$$\forall\, e_l = (v_i, v_j) \in E_D \quad t \leq \max\{alap(i), alap(j)\} \quad t \geq \min\{asap(i), asap(j)\}$$

The calculation of $z_{l,t}$ contains two terms. The first and second term have a value of 1 for all time instances greater than the end of operation i and j, respectively. To this end, the first sum of a term considers all potential resources for operation v_i or v_j, the second sum considers all potential time instances where an operation can start, i.e. $asap(i), ..., t$ and $asap(j), ..., t$, respectively.

The following set of inequalities represent that the sum of necessary transfer memory does not exceed the available memory resources:

$$\sum_{e_l\in E_D:(e_l,v_k)\in E_E} transfer(e_l, v_k) \cdot \left(\sum_q z_{l,t+q\delta} + \tilde{\lambda} d_{e_l}\right) \leq size(v_k) \tag{36}$$

$$\forall\, v_k \in V_{ER} \quad \forall\, 0 \leq t \leq \delta - 1$$

where d_e denotes the distance associated to edge e and $\lambda = \begin{pmatrix} \lambda_J & \lambda_K \end{pmatrix} = \delta\tilde{\lambda}$, $\tilde{\lambda} \in \mathbf{Z}^{2s}$, see also (25, 26, 28). Note, that the first sum takes into account all edges which contribute to the memory v_k. The next sum considers that the execution of one iteration is mapped onto one interval $0 \leq t \leq \delta - 1$. The term $\tilde{\lambda} d_{ij}$ with $(v_i, v_j) = e_l$ takes into account that the result of an operation v_i is needed at operation v_j d_{ij} iterations later. Therefore, $\tilde{\lambda} d_{ij}$ sets of data from different iterations are alive simultaneously.

The above consideration of transfer memory does not take into account the situation shown in Fig.9. The operation i generates data which are used by two (or more) operations k and l. The data should be kept in memory until the last of both operations has finished. The simple trick shown is to add an additional node v_t with computation time 0 to the graph and to associate transfer memory to edge (v_i, v_t) only.

3.4.3. Modeling As an example for the use of the above specifications the simple model of a communication between two operations is used. The two operations are executed in two different processing elements. Consequently, a bus is responsible for the data transfer. Moreover, in the two processing elements there is a memory available to store the data before and after the transmission. Only parts of the dependence graph, architecture and resource graphs are shown. A more complete example is given in the next section.

Fig.10 shows the architecture which consists of a bus B and two processing elements $P1$ and $P2$ which contain function modules $F1$, $F2$ and memory modules $M1$, $M2$, respectively. In Fig.11, the

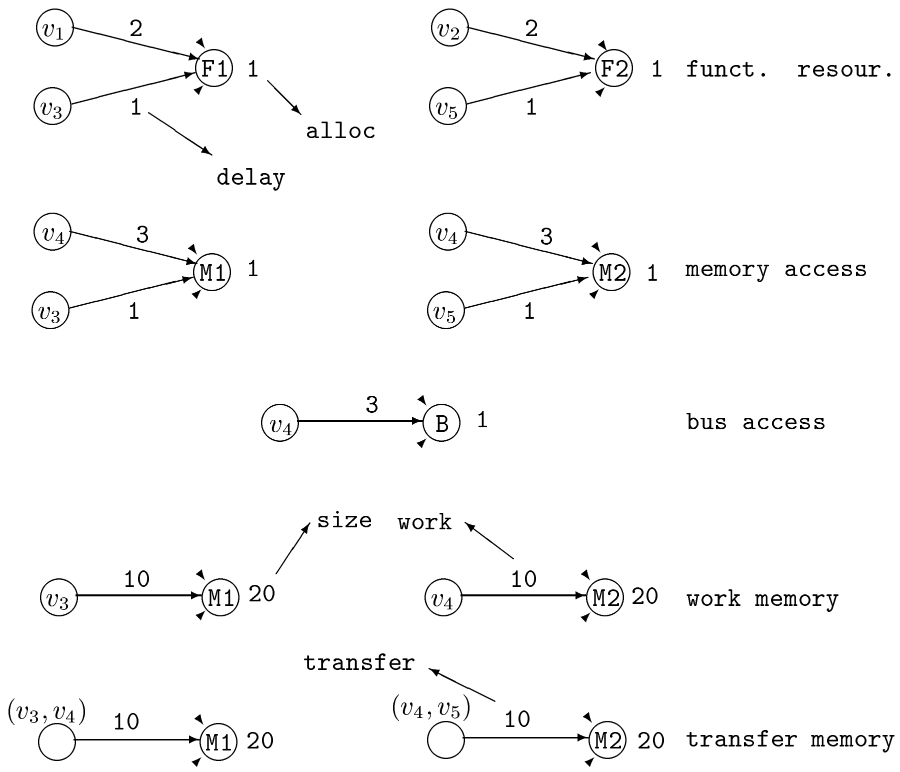

Fig. 6. Resource graphs

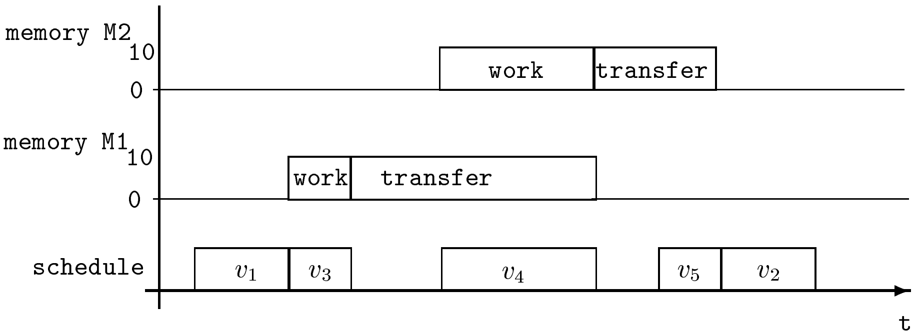

Fig. 7. Scheduling and memory usage

given dependence graph is shown and an extended one which accounts for additional operations for the communication:

- Node v_3 accounts for the time to write to the memory $M1$. In order to ensure that operation of v_3 directly starts after v_1 has finished, $t_3 - t_1 =$ "write time" must be used in the system of linear inequalities. It is supposed that the data which are generated by an operation must be stored immediately, see Figure 7.

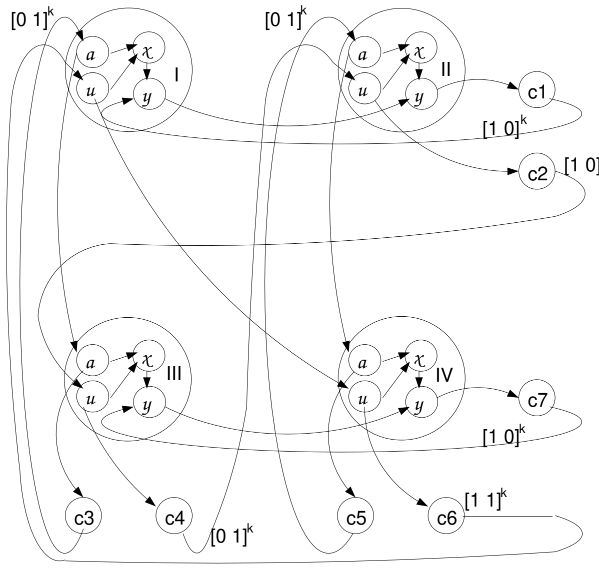

Fig. 8. RDG of the FIR operations inside a tile including the inter-processor communications.

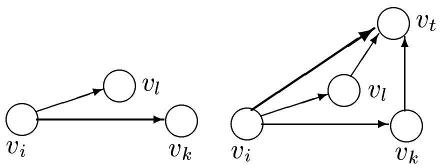

Fig. 9. Consideration of lifetimes for the calculation of transfer memory

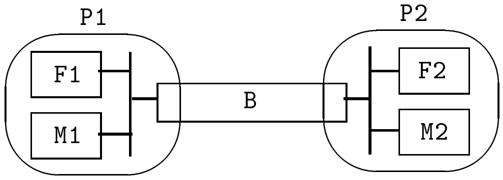

Fig. 10. Architecture model

- Node v_5 models the time to read from memory $M2$. The edge (v_5, v_2) leads to $t_2 - t_5 =$ "read time".
- Node v_4 models the reading from memory $M1$, transfer of data through the bus B and writing to memory $M2$. The computation time is $w_4 =$ "write time" $+ \frac{\text{"size of data"}}{\text{"transfer rate"}} +$ "read time".

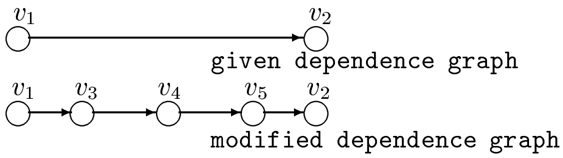

Fig. 11. Dependence graph

Fig.6 shows parts of the node- and edge-resource graphs used here. Fig.7 contains sample diagrams which show a possible schedule and the associated memory usage.

4. Experimental Results

The proposed ILP formulation that can be used to schedule partitioned regular algorithms and allocate necessary resources simultaneously has been implemented in C++, running on a SUN Sparc Station. In this section we continue with Example 1 of Section 2 to illustrate a solution calculated from our ILP. Then, we will present a second example borrowed from high-level synthesis.

By applying the ILP formulation, it is possible to take into account the number of available function units, restricted communication capabilities between function units or between function unit

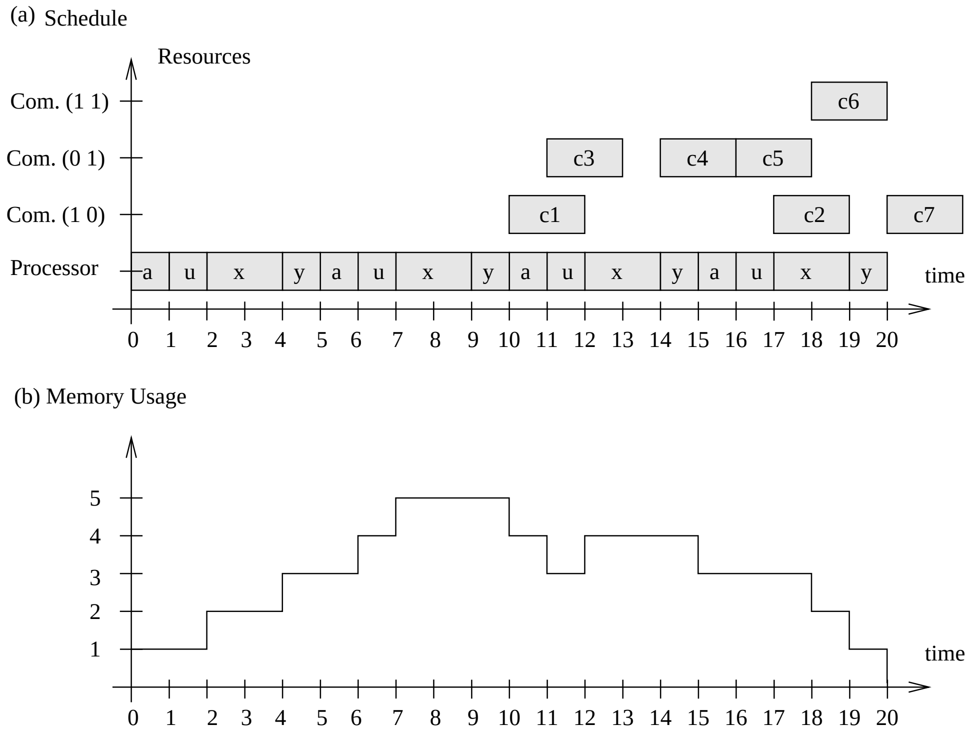

Fig. 12. (a) Bar chart of operations including communication nodes and (b) memory requirements.

and local memory, and the total amount of local memory. Meanwhile, the problem of chaining, pipelined arithmetic units, multi-cycle operations and conditional code can also be dealt with. For clarity, we will thus focus on the problems of the limitation of available function units, local memory (registers) in a processing element and finite interprocessor communication capabilities.

The implementation of a FIR filter consists of processing elements, buses and local memory. By applying the partitioning method introduced in Section 2, a tile can be obtained as shown in Fig.8. Based on the multiprojection, all operations of this tile are to be run on the same PE. Thus the following hardware resource restrictions exist:

- Within each element of a processor array, there is only 1 non-pipelined ALU and one multiplier available.
- The data transfer operation (from the registers to the processor in a PE) a and u need 1 time instance to run, the add-operation y also lasts 1 control step and the delay of the multiply-operation x takes 2 units time.
- Each PE has the inter-processor communication channels in the directions $(K_1, K_2) = (1,0)$, $(0,1)$, and $(1,1)$. The communication bandwidth is $1/2$ for each channel. That means that only one datum can be transferred through a channel during the 2 control steps.
- If the inter-processor communications can be executed immediately after the data are produced and can finished just before the data are used as inputs by the receiving PE, we need not any memory (registers) to save these data. Otherwise, the registers have to be allocated if the inter-processor channels are not available at the time the data are produced and/or the transfer cannot be completed just before the data are consumed.

Required communications are illustrated in Fig. 8: $c_1 ... c_7$ simulate the inter-processor communication. The edges with non-zero dependence vectors in the k_1 and k_2 components result in interprocessor communications which are executed on corresponding links. Correspondingly, communication nodes $c_1 ... c_7$ are added.

Now, the resource constrained scheduling and register allocation is carried out. The solution of the corresponding ILP model took about 4 seconds of CPU time. The results are $\lambda_K = (8, 13)$, $\lambda_J = (5, 10)$, $\delta = 5$, and $\Gamma = (0, 1, 2, 10, 17, 11, 14, 16, 18, 20, 20)$ where the elements of Γ are ordered according to a, u, x, y and $c_1 ... c_7$, respectively. Figs.12(a) and 12(b) show the bar chart of the schedule and the usage of local memory. It has also been evaluated that the execution time of whole FIR algorithm for $N = 60$ and $T = 40$ is 999 units of time.

The second example is a wave digital filter (WDF) that was borrowed from [18]. The WDF used for the solution of certain partial differential equations (PDE) contains 21 additions and 6 multiplications. Its index vector is of the form:

$$I \in \mathbf{I} = \{0 \leq i \leq N-1 \ \wedge \ 0 \leq j \leq M-1 \ \wedge \ 0 \leq t \leq T-1\}$$

By applying the partitioning method introduced in Section 2 and choosing the tiling matrix

$$P = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & T \end{bmatrix}$$

the tiling $\mathbf{I} = \mathbf{J} \oplus \mathbf{K}$ and the corresponding RDF can be obtained as shown in Eq. 36 and Fig.13, respectively.

$$\begin{aligned} \mathbf{J} &= \{j_1, j_2, t : 0 \leq j_1 \leq 1 \ \wedge \ 0 \leq j_2 \leq 1 \\ &\quad \wedge \ 0 \leq t \leq T-1\} \\ \mathbf{K} &= \{k_1, k_2 : 0 \leq k_1 \leq N/2-1 \\ &\quad \wedge \ 0 \leq k_2 \leq M/2-1\} \end{aligned} \tag{37}$$

All distance vectors are 0 but those whose edges contain the boxes with d_{ij}^t inside. Based on the multi-projection, all operations of tile $\mathbf{J}$ are to be run on the same PE. The implementation of the WDF consists of PEs, buses and local memory. Thus the following hardware resource restrictions exist:

- Within each element of a processor array, there are only 2 adders and one non-pipelined multiplier available.
- The delays of add-operations and multiply-operations take 1 and 2 time instances, respectively.
- There is only one inter-processor communication channel in each direction $(K_1, K_2) = (1, 0)$ and $(0, 1)$. The communication bandwidth is 1/2 for each channel. That means that only one datum can be transferred through a channel during the 2 control steps.

Again, the communications are shown in Fig.13 where $c_1 ... c_4$ represent the inter-processor communication. Similar to the previous example, the edges with non-zero dependence vectors in the k_1 and k_2 components result in interprocessor communications which are executed on corresponding links.

Scheduling and allocation took about 46 seconds of CPU time with the following results: $\lambda_K = (0, 0)$, $\lambda_J = (16, 32, 96)$, $\delta = 16$, and $\Gamma = (14, 10, 10, 7, 15, 12, 8, 0, 0, 11, 11, 2, 2, 14, 12, 1, 6, 5, 7, 4, 2, 3, 8, 12, 13, 6, 10, 15, 13, 15, 13)$ where the elements of Γ are ordered according to $a11$, $a12$, $a13$, $a14$, $a15$, $m11$, $m12$, $a21$, $a22$, $a23$, $a24$, $a31$, $a32$, $a33$, $a34$, $a41$, $a42$, $a43$, $a44$, $m41$, $m42$, $a51$, $a52$, $a53$, $a54$, $m51$, $m52$ and $c_1 ... c_4$, respectively. Figs.14(a) and 14(b) show the bar chart of the schedule and the usage of local memory. The execution time of the whole WDF algorithm for $N = M = T = 1000$ is 96096 units of time.

5. Concluding Remarks

It has been shown that the mapping of regular algorithms onto processor arrays under resource constraints is feasible. A two layer procedure has been used: At first, the algorithm is tiled and a suitable partitioning scheme (such as Locally Sequential Globally Parallel or Locally Parallel Globally Sequential) is chosen such that constraints on the number and dimensionality of the processor array are satisfied. This optimization is not part of the optimization procedure described in the present paper. After that, resource constraints on the individual processing elements are taken into account.

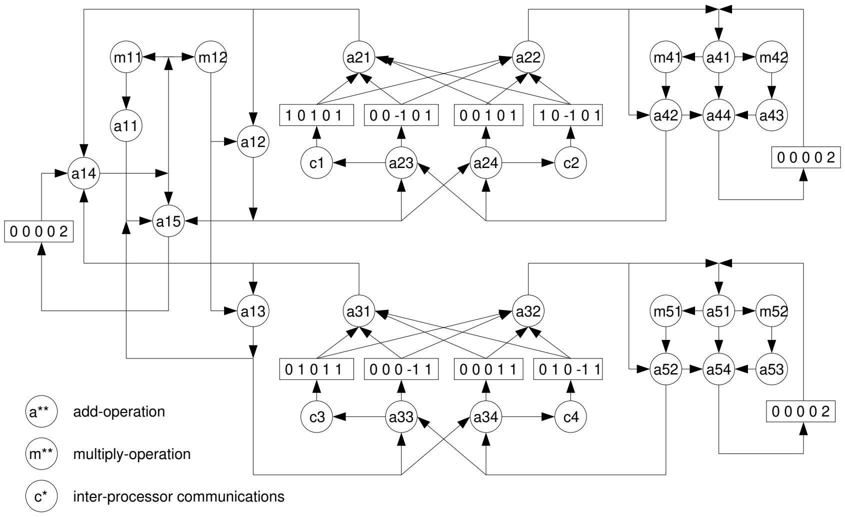

Fig. 13. RDG of WDF including the inter-processor communications

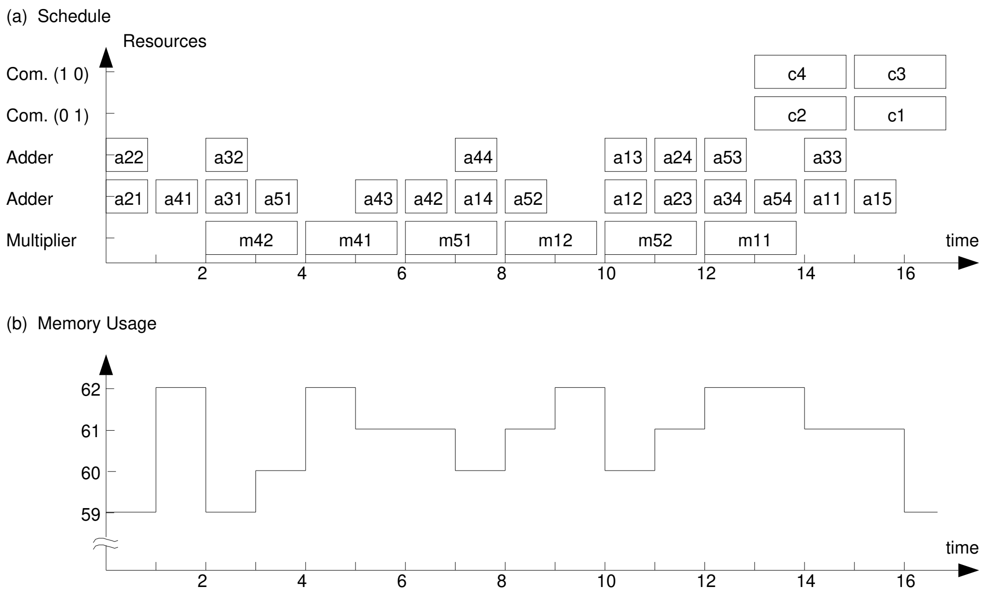

Fig. 14. (a) Bar chart of operations including communication nodes and (b) memory requirements.

In contrary to previous approaches, constraints on local memory, inter- and intra-bus bandwidth, number and types of functional resources and on access conflicts are taken into account in the second layer. Simultaneously, the computational complexity of the procedure is independent of the tile size.

On the other hand, there are some open questions which will be subject of further investigations. In particular, the ordering of operations within a tile should be part of the optimization in the second layer, i.e. the ordering of vectors in the loop matrix P. In addition, one should try to include the different partitioning possibilities into the optimization procedure such that both layers can be optimized simultaneously.

Notes

1. Given two vectors a and b with elements a_i and b_i, $a < b$ denotes $a_i < b_i$ for all i.
2. Using the same arguments as described in [18], we can also deal with regular algorithms which have the general index space form in (3) if L is regular.
3. $\mathbf{J} \oplus \mathbf{K} = \{i = j + k \ : \ j \in \mathbf{J} \wedge k \in \mathbf{K}\}$
4. $e = (1, \cdots, 1)^t$
5. The column vector c_{ij} of C associated to an edge (i, j) satisfies $c_{ij} = e_j - e_i$ where e_i and e_j denote the ith and jth unit vector, respectively.

References

1. A. Bachmann, M. Schöbinger, and L. Thiele. Synthesis of domain specific multiprocessor systems including memory design. In *VLSI Signal Processing VI*, pages 417–425, IEEE Press, New York, 1993.
2. W.H. Chou and S.Y. Kung. Scheduling partitioned algorithms on processor arrays with limited communication supports. In *Proc. of Conf. on Application Specific Processor Arrays, ASAP 93*, pages 53–64, Venice, Italy, October 1993.
3. E. Coffman. *Computer and Job-Scheduling Teory.* John Wiley and Sons, New York, 1976.
4. A. Darte and Y. Robert. Scheduling uniform loop nests. *Tech. Rep. 92-10, ENSL Lyon, France*, 1992.
5. Alain Darte and Yves Robert. Constructive methods for scheduling uniform loop nests. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 5:814–822, August 1994.
6. Michèle Dion, Tanguy Risset, and Yves Robert. Resource-constrained scheduling of partitioned algorithms on processor arrays. *Integration, the VLSI Journal*, 20:139–159, 1996.
7. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1976.
8. C.H. Gebotys and M.I. Elmasry. Global optimization approach for architectural synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-12, No. 9:1266–1278, September 1993.
9. C. Hwang, J. Lee, and Y. Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-10, No. 4:464–475, 1991.
10. Y.T. Hwang and Y.H. Hu. Novel scheduling scheme for systolic arrays. In *Proceedings of IEEE Workshop on VLSI Signal Processing*, pages 355–364, 1992.
11. M. McFarland, A. Parker, and R. Camposano. The high-level synthesis of digital system. *Proceedings of the IEEE*, 78, No. 2:301–318, 1990.
12. G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization.* John Wiley and Sons, New York, 1988.
13. S. K. Rao. *Regular iterative algorithms and their implementations on processor arrays.* PhD thesis, Stanford University, 1985.
14. J. Teich. *A Compiler for Application-Specific Processor Arrays.* Shaker (Reihe Elektrotechnik). Zugl. Saarbrücken, Univ. Diss, ISBN 3-86111-701-0, Aachen, Germany, 1993.
15. J. Teich and L. Thiele. Partitioning of processor arrays: A piecewise regular approach. *INTEGRATION: The VLSI Journal*, 14(3):297–332, 1993.
16. L. Thiele. On the design of piecewise regular processor arrays. *In Proc. IEEE Symp. on Circuits and Systems*, pages 2239–2242, 1989.
17. L. Thiele. CAD for signal processing architectures. In *in The State of the Art in Computer Systems and Software Engineering (P. Dewilde, ed.)*, Boston: Kluwer Academic Publishers, 1992.
18. L. Thiele. Resource constraint scheduling of uniform algorithms. In *Proc. of Conf. on Application Specific Processor Arrays, ASAP 93*, pages 29–40, Venice, Italy, October 1993.
19. L. Thiele. Resource constraint scheduling of uniform algorithms. *Int. Journal VLSI Signal Processing*, 10:295–310, 1995.
20. Y. Wong and J. M. Delosme. Optimization of computation time for systolic arrays. In *Proc. Int. Conf. Parallel Processing*, September 1988.
21. K.H. Zimmermann and W. Achtziger. Finding space-time transformations for uniform recurrences via branching parametric linear programming. *Int. Journal VLSI Signal Processing, to appear*, 11, 1996.