

Efficient FPGA Implementation of Equalizer for Finite Interval Constant Modulus Algorithm

Přemysl Šůcha, Zdeněk Hanzálek
Centre for Applied Cybernetics
Department of Control Engineering
Czech Technical University in Prague
{suchap,hanzalek}@fel.cvut.cz

Antonín Heřmánek, Jan Schier
Institute of Information Theory and Automation,
Academy of Sciences of the Czech Republic
{hermanek,schier}@utia.cas.cz

Abstract

This paper deals with the optimization of iterative algorithms with matrix operations or nested loops for hardware implementation in Field Programmable Gate Arrays (FPGA), using Integer Linear Programming (ILP). The method is demonstrated on an implementation of the Finite Interval Constant Modulus Algorithm, proposed for 4G communication systems. We used two pipelined arithmetic libraries based on the logarithmic number system or the floating-point number system, using the widely known IEEE format for the floating-point calculations required in the algorithm. Traditional approaches to the scheduling of nested loops lead to a relatively large code, which is unsuitable for FPGA implementation. This paper presents a new high-level synthesis methodology, which models both, iterative loops and imperfectly nested loops, by means of the system of linear inequalities. Moreover, memory access is considered as an additional resource constraint. Since the solutions of ILP formulated problems are known to be computationally intensive, important part of the article is devoted to the reduction of the problem size.

Keywords: High-level synthesis, cyclic scheduling, iterative algorithms, imperfectly nested loops, integer linear programming, FPGA, control.

1 Introduction

This paper deals with the scheduling of digital signal processing algorithms, where at least K iterations have to be executed each sampling period in order to achieve the desired algorithm convergence. Moreover, the iteration may contain rather complex data computations, usually expressed in the form of matrix operations and implemented as nested loops. The scheduling method shown in this paper models both, iterative loops and imperfectly nested loops, by means of the system of linear inequalities. The target hardware based on FPGA, a set of pipelined dedicated processors, is formalized as a system of linear inequalities with integer variables. Thanks to this mathe-

matically sound and unambiguous formalism, we solve the scheduling problem by Integer Linear Programming (ILP), while achieving the optimal schedule. Thanks to an efficient representation of imperfectly nested loops and to a polynomial reduction of decision variables we are able to handle scheduling problems of significant size as demonstrated on hardware implementation of FI-CMA. We show, how the code-size efficient synthesis on FPGAs can be solved by well reputed mathematical programming.

1.1 Motivation

The FPGA design gets complicated, due to the representation of real numbers. One solution is to use an arithmetic library implementing a 32-bit floating point number system, compliant with IEEE standards [3]. An alternative solution is to use the logarithmic number system arithmetic, where a real number is represented as the fixed-point value of base two logarithm of its absolute value [13]. In each case, rather complex arithmetic is required. Therefore, scheduling of such dedicated HW resources has to carefully consider the algorithm structure, in order to achieve the desired performance of applications. Scheduling also helps to choose the appropriate arithmetic library prior to the algorithm implementation.

To demonstrate the scheduling method, we consider the Finite Interval Constant Modulus Algorithm (FI-CMA), which will be used in 4G communication systems. Compared to other algorithms in the class of blind deconvolution algorithms (performing estimation without any training sequence), it is characterized by fast convergence at acceptable computational complexity. Nevertheless FI-CMA requires to perform 8 iterations (operating on matrices of sampled data) each $3.7\mu s$. While using our scheduling method, we reached a speedup of 46% in comparison with [9], the first implementation of FI-CMA in FPGA.

1.2 Related Work on Scheduling of Iterative Algorithms

An iterative algorithm can be seen as a computation loop performing an identical set of operations repeatedly (one repetition of the loop is called an *iteration*). When the number of iterations is large enough, the optimization can be performed by *cyclic scheduling* while minimizing the completion time of the set of K iterations. If operations belonging to different iterations interleave, the schedule is called *overlapped* [17]. Efficient exploitation of the schedule overlap and pipelining is rather difficult to achieve in manual design.

The *periodic schedule* is given by a schedule of one iteration that is repeated with a fixed time interval called a *period* (also called *initiation interval*). The aim is to find a periodic schedule with a minimum period. If the number of processors is not limited, a periodic schedule can be built in polynomial time [8]. For a fixed number of processors the problem becomes NP-hard [8]. The general solution to this problem is shown e.g. in [5]. Cyclic scheduling presented in [19] is not dependent on the period length and with respect to [5] it leads to simpler problem formulation with less integer variables. Moreover, this model allows to reduce number of interconnections by minimization of the data transfers as shown in [15]. *Modulo scheduling* and *software pipelining* [11], usually used in the compiler community, are alternative terms to *cyclic scheduling*, used in the scheduling community.

1.3 Related Work on Scheduling of Nested Loops

For practical reasons, we usually do not want to expand matrix operations into scalar operations, since we want to achieve regularity of the schedule (efficiently implemented in the form of the nested loops) and we want to prevent enormous growth in the number of the scheduled tasks (i.e. to prevent the growth of computation time required by the scheduling algorithm). Therefore, we intend to schedule matrix operations in the form of nested loops.

A great deal of work in this field has been focused on *perfectly-nested* loops (i.e. all elementary operations are contained in the innermost loop). For example, one of the often used optimization approaches called loop shifting (operations from one iteration of the loop body are moved to its previous iteration) was recently extended in [7, 4]. Another approach is the unroll-and-jam (also called unfolding or unwinding) [2], which partially unrolls one or more loops higher in the nest than the innermost loop, and fuses the resulting loops back together. Improvement by unroll-and-squash technique has been shown in [14].

In our case the loops are *imperfectly-nested* (i.e. some elementary operations are not contained in the innermost loop). Existing compilers usually use heuristics transforming them into perfectly-nested loops [20]. The tiling method, extended for imperfectly-nested loops, is dis-

cussed in [1]. Loop tiling is a transformation technique, which divides the iteration space of loop computations into tiles (or blocks) of some size and shape, so that traversing the tiles results in covering the whole iteration space [20]. However, such approaches can greatly expand the code size, which is unacceptable with respect to the limited size of FPGAs as well as to the number of interconnections in the design.

1.4 Outline and Contribution

In this paper, we propose a new method for scheduling of iterative algorithms with imperfectly-nested loops on the set of pipelined dedicated processors. The method is based on cyclic scheduling of iterative algorithms where matrix operations are modeled by “united edges” and “approximated expansion”. The method is based on the construction of an abstract model, which models the nested loops, and which is optimally scheduled using integer linear programming (ILP). Moreover, access into the memory is considered as an additional resource constraint.

A lot of research has been done in scheduling of nested loops as mentioned above. Our method differs in mathematic formulation of the scheduling problem which leads to simpler code and therefore more efficient FPGA implementation. Applications requiring matrix operations usually lead to schedules with long period. Therefore classical ILP formulations of cyclic scheduling (e.g. [17]), where the number of integer/binary variables (and also time complexity) depends on the period length, are inconvenient in this framework. Number of integer/binary variables of our ILP model is independent of period length. Primarily, we look for the shortest feasible period. For such period we find a schedule minimizing the number of interconnections (related to the data transfers).

The proposed scheduling method is demonstrated on the Finite Interval Constant Modulus algorithm (FI-CMA). To achieve an acceptable computational performance and precision, the iterative algorithm has to be implemented using floating-point arithmetics. We consider two libraries of arithmetic units (Celoxica pipelined floating-point library (FP32) [3] and High-Speed Logarithmic Arithmetics (HSLA) [13]) and we show that by using the presented method, the optimal architecture can be chosen for a given algorithm prior to its implementation.

The paper is organized as follows: in the next section the FI-CMA algorithm is briefly outlined. Section 3 surveys the scheduling of iterative algorithms (considering only scalar variables) on the set of dedicated processors by Integer Linear Programming (multiprocessor extension of [19]). Section 4 presents the main contribution of the paper, scheduling of iterative algorithms with imperfectly-nested loops, illustrated by the parallelization of the FI-CMA algorithm. Section 5 presents the HW implementation of FI-CMA algorithm on FPGA. The last section concludes the work.

2 FI-CMA Algorithm

In modern digital communication systems, an estimator of transmitted symbols represents one of the critical parts of the receiver. Since Ultra High Frequency radio waves bounce off everything (e.g. hills, buildings), many reflected signals, each with a different phase, can reach an antenna. *Equalization* is used to reconstruct the desired signal y from the mixture of received signal u also containing unwanted reflections and noise b (see Figure 1). Recent systems (such as the GSM system) use methods based on training sequences, where part of a signal is known and repeated. It works by finding out how a known transmitted signal s is modified by multipath fading, and by constructing an inverse filter g to extract the rest of the desired signal. Unfortunately, the training sequence consumes a considerable part of the overall message (approximately 25% in GSM). For this reason, much research effort has been devoted to blind deconvolution algorithms, i.e., algorithms with no training sequence. Perhaps the most popular blind algorithm is the Constant Modulus Algorithm, originally proposed by Godard [6] and improved as the Finite Interval Constant Modulus Algorithm (FI-CMA) [16].

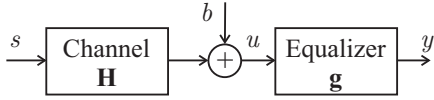


Figure 1: Simple discrete model of a digital communication system

An equalizer can be seen as a linear combiner of order M . While considering the oversampling factor equal to 1, N successive equalizer outputs can be directly rewritten in matrix form as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} u_1 & u_0 & \cdots & u_{1-M} \\ u_2 & u_1 & \cdots & u_{2-M} \\ \vdots & \vdots & \ddots & \vdots \\ u_N & u_{N-1} & \cdots & u_{N-M} \end{bmatrix} \cdot \mathbf{g} = \underbrace{\mathbf{U}}_{\mathbf{U}} \cdot \mathbf{g} = \mathbf{Q} \cdot \mathbf{R} \cdot \mathbf{g} = \mathbf{Q} \cdot \mathbf{w}, \quad (1)$$

where vector \mathbf{w} contains coefficients of the equalizer. Note that the orthonormal matrix \mathbf{Q} is obtained by QR-decomposition of matrix \mathbf{U} . It was shown in [16] that the optimal equalizer coefficients \mathbf{w} can be reached by an iterative procedure:

$$\begin{aligned} \text{for } k = 1 & \text{ to } K \text{ do} \\ \mathbf{y}(k) &= \mathbf{Q} \cdot \mathbf{w}(k-1) \\ \mathbf{v}(k) &= \mathbf{w}(k-1) - \mu \cdot \mathbf{Q}^T \cdot \mathbf{y}(k)^3 / F(k) \\ \mathbf{w}(k) &= \mathbf{v}(k) / \|\mathbf{v}(k)\| \\ \text{end} \end{aligned} \quad (2)$$

$$\text{where } k \text{ is the iteration counter and } F(k) = \frac{\sum_{i=1}^N y_i(k)^4}{(\sum_{i=1}^N y_i(k)^2)^2}.$$

The above mentioned iterative procedure with matrix operations will be used to illustrate our scheduling method for hardware implementation on FPGA with arithmetic units.

3 Formulation and Solution of the Scheduling Problem

The iterative procedure, as the one mentioned in the previous section, can be implemented as a computation loop performing an identical set of operations repeatedly. Therefore our work, dealing with an optimized implementation of such procedures, is based on cyclic scheduling.

3.1 Cyclic Scheduling Problem

Operations in a computation loop can be considered as a set of n generic tasks $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ to be performed K times where K is usually very large. One execution of \mathcal{T} labeled with integer index $k \geq 1$ is called an *iteration*. The scheduling problem is to find a start time $s_i(k)$ of every occurrence T_i [8]. Figure 2 shows an illustrative example of a simple computation loop with corresponding processing times of operations executed using HSLA arithmetic library on FPGA (see Table 1 in Section 4).

Data dependencies of this problem can be modeled by a directed graph G . Each task (node in G) is characterized by the processing time p_i . Edge e_{ij} from the node i to j is weighted by a couple of integer constants l_{ij} and h_{ij} . Length l_{ij} represents the minimal distance in clock cycles from the start time of the task T_i to the start time of T_j and is always greater than zero.

The notions of the length l_{ij} and the processing time p_i are useful when we consider the pipelined processors used in both arithmetic libraries HSLA and FP32, which is presented in Section 4. The processing time p_i represents the time to feed the processor (i.e. new data can be fed to the pipelined processor after p_i clock cycles) and length l_{ij} represents the time of computation (i.e. the input-output latency). Therefore, the result of a computation is available after l_{ij} clock cycles. On the other hand, the height h_{ij} specifies a shift of the iteration index (dependence distance) related to the data produced by T_i and consumed by T_j .

Assuming a *periodic schedule* with the *period* w (i.e. the constant repetition time of each task), each edge e_{ij} in graph G represents one precedence relation constraint

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij}, \quad (3)$$

where s_i denotes the start time of task T_i in the first iteration. Figure 2(b) shows the data dependence graph of the computation loop shown in Figure 2(a).

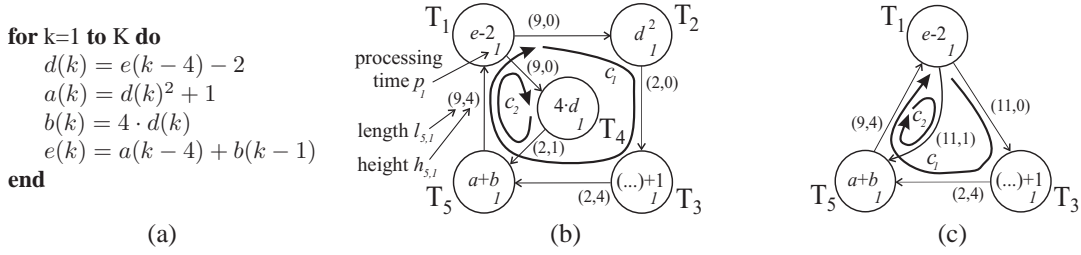


Figure 2: (a) Illustrative example of the iterative algorithm. (b) The corresponding data dependency graph G . G contains two cycles c_1 and c_2 with average cycle times $\{22/8, 20/5\}$. The critical circuit is c_2 . (c) The corresponding reduced graph, which is explained in Section 3.2.

The aim of cyclic scheduling [8] is to find a periodic schedule while minimizing the period length w . The scheduling problem is simply solved when the number of processors is not limited, i.e. is sufficiently large. Thereafter, the minimal feasible period w is given by the *critical circuit* c in graph G , maximizing the ratio

$$w = \max_{c \in C(G)} \frac{\sum_{e_{ij} \in c} l_{ij}}{\sum_{e_{ij} \in c} h_{ij}}, \quad (4)$$

where $C(G)$ denotes a set of cycles in G . Critical circuit can be found in polynomial time, $O(n^3 \cdot \log(n))$ [8], and we use this value to determine lower bound of the period length w in our scheduling problem.

3.2 Problem with Dedicated Processors

When the number of processors m is restricted, the cyclic scheduling problem becomes NP-hard [8]. Unfortunately, in our case the number of processors is restricted and the processors are dedicated to execute specific operations (see Table 1). Due to the NP-hardness it is meaningful to formulate the scheduling problem as a problem of Integer Linear Programming (ILP), since various ILP algorithms solve instances of reasonable size in reasonable time. Fortunately, in FPGA applications some tasks (further called *reduced tasks*, e.g. T_2 and T_4 in Figure 2(b)) need not be restricted by resource constraints, since there is a sufficient number of processors (further called *reduced processors*) able to execute these tasks (e.g. cheaply implemented MUL operations in logarithmic arithmetics). Then only the tasks on dedicated processors, denoted by the set T' , must be considered in the scheduling algorithm. The input-output latency of the reduced tasks is reflected in the lengths of arcs in the reduced graph G' . When the tasks on dedicated processors are scheduled, the reduced tasks are simply executed as soon as possible, while satisfying precedence relations (given by the original graph G).

Therefore, we perform the following reduction of graph G to the *reduced graph* G' while using the calculation of the longest paths (solved e.g., by the Floyd's algorithm). All nodes (tasks), except the ones running on the dedicated processors, are eliminated (see Figure 2(c)). Therefore, tasks T' running on the dedicated processors consti-

tute the nodes of G' . There is e'_{ij} (the edge from T_i to T_j in G') of height h'_{ij} , if and only if, there is a path from T_i to T_j in G of height h'_{ij} such that this path goes only through the reduced tasks. The value of length l'_{ij} is the longest path from T_i to T_j in G of height h'_{ij} . Therefore, the input-output latency of the reduced tasks is merged into l'_{ij} and their iteration shift into h'_{ij} .

Such reduction allows to formulate our scheduling problem as a problem with set of m' dedicated processors, where the set of tasks T' is the conjunction of disjoint subsets $T'_1, \dots, T'_d, \dots, T'_{m'}$. Both tasks T_i and T_j are assigned to the d -th dedicated processor if and only if $T_i \in T'_d$ and $T_j \in T'_d$. The reduction, considering one dedicated processor (ADD unit) performed on graph G , in Figure 2(b), is shown in Figure 2(c).

3.3 Solution of Cyclic Scheduling on Dedicated Processors by ILP

The approach presented in this subsection is a multiprocessor extension of work shown in [19]. The size of our ILP model (and also time complexity) is independent of the period length w , which is particularly useful in our application since matrix operations and nested loops lead to schedules with a long period.

Let \hat{s}_i be the remainder after division of s_i by w and let \hat{q}_i be the integer part of this division. Then s_i can be expressed as follows

$$s_i = \hat{s}_i + \hat{q}_i \cdot w, \quad \hat{s}_i \in \langle 0, w-1 \rangle, \quad \hat{q}_i \geq 0. \quad (5)$$

This notation divides s_i into \hat{q}_i , the index of the *execution period*, and \hat{s}_i , the index within the execution period. The schedule has to obey two kinds of restrictions. The first kind of restriction is given by the *precedence constraint* corresponding to Inequality (3). It can be formulated using \hat{s} and \hat{q} as

$$(\hat{s}_j + \hat{q}_j \cdot w) - (\hat{s}_i + \hat{q}_i \cdot w) \geq l'_{ij} - w \cdot h'_{ij}. \quad (6)$$

We have n'_e inequalities (n'_e is the number of edges in the reduced graph G'), since each edge represents one precedence constraint.

The second kind of restriction are *processor constraints*. They are related to the processor restriction, i.e. at maximum one task is executed at a given time on the dedicated processor P_d , therefore

$$p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i, \quad (7)$$

where \hat{x}_{ij} is a binary decision variable ($\hat{x}_{ij} = 1$ when T_i is followed by T_j and $\hat{x}_{ij} = 0$ when T_j is followed by T_i).

To derive a feasible schedule, Double-Inequality (7) must hold for each unordered couple of two distinct tasks $T_i, T_j \in \mathcal{T}'_d$. Therefore, there are $(n'_d{}^2 - n'_d)/2$ Double-Inequalities related to processor P_d (where n'_d is the number of tasks on one dedicated processor P_d in the reduced graph G'), i.e. there are $n'_d{}^2 - n'_d$ inequalities specifying the processor constraints.

Using ILP formulation we are able to test the schedule feasibility for a given w . In addition, we can minimize the data transfers among the tasks (i.e. minimize the number of registers used to store intermediate results) as an objective function. Minimization of data transfers considerably simplifies interconnections on FPGA. To achieve this we add one slack variable Δ_{ij} to each precedence constraint (6) resulting at

$$(\hat{s}_j + \hat{q}_j \cdot w) - (\hat{s}_i + \hat{q}_i \cdot w) + \Delta_{ij} = l_{ij} - w \cdot h_{ij}. \quad (8)$$

When $\Delta_{ij} = 0$, the intermediate result is passed to the next task without storing in registers or memory. On the other hand when $\Delta_{ij} > 0$, the memory or register is required. The aim is to minimize the number of $\Delta_{ij} > 0$. Therefore we introduce new binary variable Δ_{ij}^b which is equal to 1 when $\Delta_{ij} > 0$ and Δ_{ij}^b is equal to 0 otherwise. This relation is formulated as

$$(w \cdot (\hat{q}_{max} + 1)) \cdot \Delta_{ij}^b \geq \Delta_{ij}, \quad \forall e_{ij} \in G, \quad (9)$$

where $(w \cdot (\hat{q}_{max} + 1))$ represents an upper bound on Δ_{ij} and the objective is to minimize $\sum \Delta_{ij}^b$. Such a reformulated problem not only decides the feasibility of the schedule for the given period w , but if such a schedule exists, it also finds the one with minimal data transfers among the tasks.

The summarized ILP model, using variables $\hat{s}_i, \hat{q}_i, \hat{x}_{ij}, \Delta_{ij}^b, \Delta_{ij}$, is shown in Figure 3. It contains $2n' + \sum_{d=1}^{m'} (n'_d{}^2 - n'_d)/2 + n'_e$ integer variables and $2n'_e + \sum_{d=1}^{m'} (n'_d{}^2 - n'_d)$ constraints.

Please notice that w is assumed to be a constant in the ILP model (in order to avoid multiplication of two variables). Optimal period w^* , the shortest period resulting in a feasible schedule, can be found by formulating one ILP model for each w between the lower and upper bound (explained in Subsection 5.2). Moreover, the interval bisection method can be used, since w^* is not preceded by any feasible solution (i.e. no $w \leq w^* - 1$ re-

$$\begin{aligned} & \min \sum \Delta_{ij}^b \\ & \text{subject to} \\ & \hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w + \Delta_{ij} = l'_{ij} - w \cdot h'_{ij}, \\ & \hspace{15em} \forall e'_{ij} \in G' \\ & p_j \leq \hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} \leq w - p_i, \\ & \hspace{15em} \forall i, j : i < j \text{ and } T_i, T_j \in \mathcal{T}'_d \\ & (w \cdot (\hat{q}_{max} + 1)) \cdot \Delta_{ij}^b \geq \Delta_{ij}, \\ & \hspace{15em} \forall e'_{ij} \in G' \\ & \text{where} \\ & \hat{s}_i \in \langle 0, w - 1 \rangle; \hat{q}_i, \Delta_{ij} \geq 0; \hat{x}_i, \Delta_{ij}^b \in \langle 0, 1 \rangle, \\ & \hspace{15em} \hat{s}_i, \hat{q}_i, \hat{x}_{ij}, \Delta_{ij}^b \text{ are integers.} \end{aligned}$$

Figure 3: ILP model.

sults in a feasible solution). Therefore, there are, at maximum, $\log_2(w_{upper} - w_{lower})$ iterative calls of ILP, where w_{lower} and w_{upper} are the lower and upper bound of w , respectively.

4 Parallelization of Algorithms with Matrix Operations

As already mentioned, the dynamic range of data in an FI-CMA algorithm requires calculations in the floating-point. That is, however, rather costly for an FPGA implementation. In this work, we consider two arithmetic libraries HSLA and FP32.

The High-Speed Logarithmic Arithmetic (HSLA) library, implementing the logarithmic arithmetics, consists of fully pipelined blocks, implementing basic arithmetic operations. Multiplication, division and square-root are implemented simply as fixed-point addition, subtraction and right shift, but addition and subtraction require more complicated evaluation using look-up tables. In order to utilize the look-up tables efficiently, the addition and subtraction operations have been implemented as a twin-adder. For more details see [13].

The second one, the 32-bit Celoxica pipelined floating-point library (FP32) [3] uses the widely known IEEE format to store the data. Both libraries have comparable precision, but different timing parameters (see Table 1). Namely HSLA has efficient implementation of MUL, DIV and SQRT units, however ADD unit is the only one, which is expensive in clock cycles, SLICES and Block RAMs (BRAMs). Therefore, HSLA can be modeled by one dedicated processor, since the number of MUL, DIV and SQRT units can be almost arbitrary. Consequently these units are not critical resource and they are not subject of processor constraints (see Subsection 3.2). On the other hand, FP32 has all units of non negligible size, but it can be run on a higher clock frequency than HSLA. Therefore, it has to be modeled by several dedi-

cated processors.

Unit	HSLA / FP32 (19(32)-bit, XCV2000E-6)			
	Max clock frequency: 50 / 93 MHz			
	Processing time [clk]	Input-output Latency [clk]	SLICES [%]	BRAMs [%]
ADD	1 / 1	9 / 11	8(13) / 8	3(70) / 0
MUL	1 / 1	2 / 8	1 / 4	0 / 0
DIV	1 / 1	2 / 28	1 / 10	0 / 0
SQRT	1 / 1	2 / 27	1 / 6	0 / 0

Table 1: Arithmetic units parameters of the 19/32-bit HSLA and 32-bit FP32.

To demonstrate the scheduling method, we use FI-CMA algorithm. But this method is universal for all iterative algorithms with matrix operations or imperfectly nested loops.

According to the results obtained by the scheduling method presented in this paper we decided to implement FI-CMA algorithm with HSLA library, as explained in Section 5.1. Therefore, the parameters of HSLA are used in all examples even though we have modeled and calculated both cases.

The FI-CMA algorithm consists of two successive parts: the input data matrix is processed by the *QR-decomposition algorithm* to obtain matrix \mathbf{Q} (see equation (1)), which is further used by the *equalizer algorithm* (based on equations (2)). Matrix \mathbf{Q} of N rows and M columns is known after the last iteration of the QR-decomposition, therefore, both parts are scheduled separately.

Both parts are iterative algorithms, and in this section we show how the cyclic scheduling (given in Section 3) can be used for their parallelization. Since the equalizer algorithm is more complex and well suited for demonstration of the scheduling method, the parallelization of the QR-decomposition is not discussed here.

4.1 Equalizer Algorithm

The equations (2) are implemented as an equalizer algorithm, shown in Figure 4(a), while denoting $\|v_{i+1}\|$ by α and $(\mathbf{Q}^T \cdot \mathbf{y}^3)$ by \mathbf{v}_Δ . The parallelism in the loop is increased while substituting \mathbf{y} with $(\mathbf{y}' \cdot \alpha)$. This substitution increases the number of multiplications, but it significantly increases the parallelism among additions. The corresponding data dependencies are shown in Figure 4(b).

4.2 Expansion of Imperfectly Nested Loops

A single node of the *condensed graph* G^c (see Figure 4(b)) represents the set of tasks performing e.g. vector addition, vector multiplication, scalar addition, With respect to further efficient implementation, we do not want to simply expand these nodes into scalar operations but we want to keep them in a vector form intending to implement them as computation loops. The reduced condensed graph G'^c , shown in Figure 4(c), is obtained from Figure 4(b) while

reducing the nodes not executed on the twin-adder. The presented approach is based on the expansion of G'^c to graph G' (see Figure 6), where the first level of nesting is modeled by using, so called, *united edges* and the second level of nesting (for matrix operations) is modeled by the *processing time fusion* while fully utilizing the twin-adder. Further G' can be scheduled by ILP in Figure 3 while partially fixing the precedence constraints.

4.2.1 Processing Time Fusion

For example, task T_1 in G'^c computes matrix-vector multiplication $\mathbf{y}' = \mathbf{Q} \cdot \mathbf{v}$. When the multiplication is evaluated in a common form, i.e. row-wise with respect to matrix \mathbf{Q} , the efficiency of the resource utilization is relatively small. This is caused by the relatively big input-output latency of the twin-adder with respect to the row length. We propose to compute the multiplication in column-wise form, where all the elements of the j^{th} column are multiplied by a j^{th} element of vector \mathbf{v} . The partial sums are stored in the memory. For example, the computations of all rows in the 1^{st} column (second level of nesting) are represented by task $T_{1,1}$ in G' in Figure 5. Task $T_{1,1}$ represents the fused ADD operations of the 1^{st} column, and therefore, its processing time $p_{1,1}$ is equal to N . The outer loop over the columns, further called *column loop* (first level of nesting), is modeled using united edges, explained below.

4.2.2 United Edges

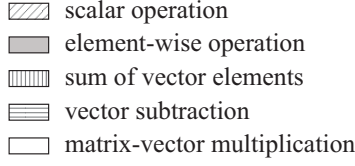
The first level of nesting is modeled by loop expansion. In order to keep regularity of the loop implementation, the time delay between consequent iterations of the column loop must be the same. Therefore, we introduce a new kind of edge - *united edges*. For example, when the united edge from $T_{1,1}$ to $T_{1,2}$ belongs to the same group g as the united edge from $T_{1,2}$ to $T_{1,3}$ then $s_{1,2} - s_{1,1} = s_{1,3} - s_{1,2}$. The equivalent precedence constraints are expressed by equalities:

$$\begin{aligned} s_{1,2} - s_{1,1} - z_g &= l_g, \\ s_{1,3} - s_{1,2} - z_g &= l_g, \end{aligned}$$

where z_g is one variable of ILP, such that $z_g \geq 0$, and constant l_g is the length of the united edge. Both z_g and l_g are common for all united edges belonging to the group g . Then $w_g = z_g + l_g$ is a period of the column loop belonging to group g . Since one half of the columns are processed on the second twin-adder, the model consists of $M/2$ iterations of the column loop.

4.2.3 Approximated Expansion

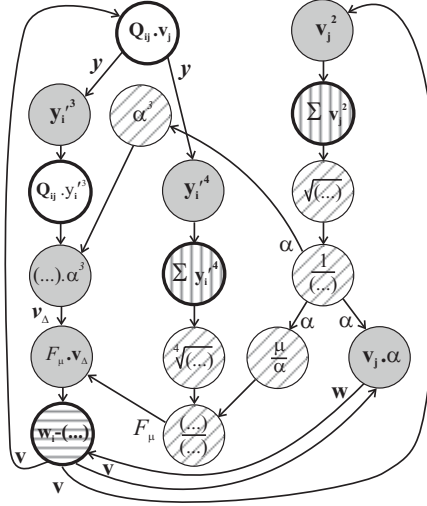
The scheduling algorithm (in Figure 3) for model with processing time fusion and united edges can find such an inner period w_g so that the global schedule is optimal. To



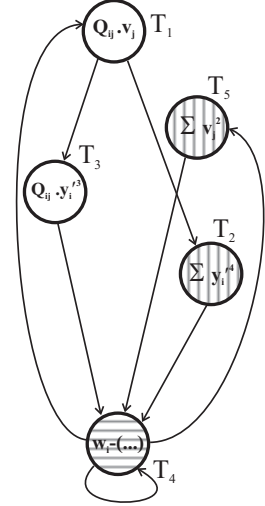
for k=1 to K do

$$\begin{aligned}
 \mathbf{y}'(k) &= \mathbf{Q} \cdot \mathbf{v}(k-1) \\
 \alpha(k-1) &= \frac{1}{\sqrt{\sum v^{(k-1)^2}}} \\
 \mathbf{w}(k-1) &= \mathbf{v}(k-1) \cdot \alpha(k-1) \\
 F_\mu(k) &= \frac{\mu}{\sqrt{\sum y'^{(k)^4 \cdot \alpha(k-1)}}} \\
 \mathbf{v}_\Delta(k) &= \mathbf{Q}^T \cdot \mathbf{y}'(k)^3 \cdot \alpha(k-1)^3 \\
 \mathbf{v}(k) &= \mathbf{w}(k-1) - F_\mu(k) \cdot \mathbf{v}_\Delta(k)
 \end{aligned}$$

end



G^c
(b)



G'^c
(c)

(a)

Figure 4: (a) Equalizer algorithm. (b) Data dependencies of the algorithm represented by the condensed graph G^c . (c) Corresponding reduced condensed graph G'^c .

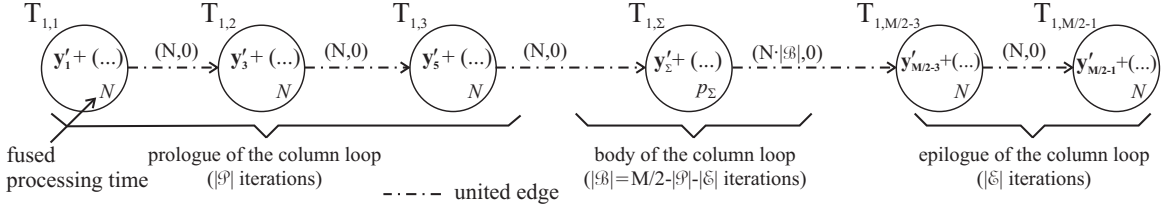


Figure 5: Approximated expansion of the column loop (first level of nesting) of matrix-vector multiplication $\mathbf{y}' = \mathbf{Q} \cdot \mathbf{v}$.

reduce the number of tasks ($T_{1,1}, T_{1,3}, \dots, T_{1,M/2-1}$) representing expanded iterations of the nested loop processed on the first twin-adder, we propose the following *approximated expansion*. We divide the iterations of the nested loop into a *prologue* \mathcal{P} , a *body* \mathcal{B} and an *epilogue* \mathcal{E} . The prologue and epilogue contain $|\mathcal{P}|$ (respectively $|\mathcal{E}|$) iterations, modeled as explained above.

The body contains one task fully exploiting the dedicated processor and therefore, preventing any other task using it. In fact it represents $|\mathcal{B}|$ iterations, i.e. $|\mathcal{B}| = |\mathcal{S}| - |\mathcal{P}| - |\mathcal{E}|$, where $|\mathcal{S}|$ is number of modeled iterations (in case of task T_1 $|\mathcal{S}| = M/2$). Processing time of the body task p_Σ is variable and it is equal to $p_\Sigma = |\mathcal{B}| \cdot N + (|\mathcal{B}| - 1) \cdot z_g$. United edge from the body to the epilogue is given by the following precedence constraint:

$$s_j - s_i - |\mathcal{B}| \cdot z_g = |\mathcal{B}| \cdot l_g. \quad (10)$$

The utility of the approximation depends on the estimation of the prologue and the epilogue length permitting sufficient overlap of single operations. For example, when there are two loops that can be processed in parallel while

sharing one dedicated unit, then their interleaving has to be carefully implemented since, in the model, the body of one loop blocks the dedicated unit.

4.3 Access to a Dual Ported BRAM

The previous section described modeling of the first and the second level of nesting by expansion of the condensed graph G'^c to G' . The model considers restriction to the one ADD unit only. In fact there are two ADD units realized as twin-adder, both accessing a dual ported BRAM. Using the dual ported BRAMs of the device, we are able to address two columns of \mathbf{Q} at the same time. Since iterations of the first level nested loop are identical, odd iterations are processed on the first half of the twin-adder and even iterations on the second one. Thereafter, both halves of the twin-adder process the same tasks in the same order, but they are shifted by 9 clocks, i.e. input-output latency of ADD unit in HSLA library. It is due to the proposed HW architectures where the first half reads the partial sum value of \mathbf{y}' from the dual ported BRAM, the second half directly reads the value from the first half without stor-

age in the memory and finally the second half writes the partial sum value of \mathbf{y}' to the second port of dual ported BRAM.

Note: Since variables \mathbf{v} , \mathbf{w} are located in a distributed memory where the number of accesses is not restricted, the memory access of \mathbf{v} , \mathbf{w} is not considered as an operation on the dedicated processor (the dual ported BRAM). In the same way the memory access of \mathbf{Q} is not a subject of the precedence constraint, even though it is located in another dual ported BRAM, since the reading of entries of \mathbf{Q} (\mathbf{Q} is constant during equalizer algorithm) is done synchronously with long addition operation.

Therefore, we assume two distinct dedicated processors. The first one is the twin-adder and the second one is the dual ported BRAM containing vector \mathbf{y}' . The accesses into the memory containing \mathbf{y}' are modeled as new tasks labeled "READ" in Figure 6. Data read from the dual ported BRAM is directly processed in the twin-adder in order to avoid temporal storage of data in the register. Therefore, precedence constraints related to this data are no longer given by the inequality (3), but they are given by the equality $s_j - s_i = l_{ij} - w \cdot h_{ij}$. Such an edge, represented by a dashed line, is called a *fixed edge* in the rest of this text.

4.4 Equalizer Algorithm Schedule

The resulting equalizer algorithm model by the reduced graph G' including accesses into the memory (fixed edges) and expansion of nested loops (united edges) is shown in Figure 6.

The particular constants M and N are chosen to be greater than the input-output latency of the pipelined twin-adder (i.e. $M = 20$ and $N = 24$). The schedule of this instance of cyclic scheduling is shown in Figure 7. While setting greater values of constants M and N , we obtain different parameters of graph G' and an unclear Gantt chart, therefore, we have shown this example.

The model in Figure 6 considers two processors, i.e. the first half of twin-adder and the dual ported BRAM containing vector \mathbf{y}' . But it corresponds to the HW design with both parts of the twin-adder, since odd iterations are processed on the first half of the twin-adder and even iterations on the second one. This mapping scheme known from parallel computing can be easily used to extend this model to a different number of ADD units. The advantage of such a representation lies in smaller corresponding ILP model and therefore, lower computing time requirements. Moreover, it leads to a simpler code structure and therefore a smaller FPGA design.

Figure 7 shows the interleaving of two iterations. It contains only two dedicated processors, but in fact we obtain a more complex schedule, also including reduced processors and processing in pipelines (the length of rectangular blocks in Figure 7 corresponds only to the processing time, not to the input-output latency). From this complex schedule, we automatically generate the code in

	XCV2000E-6		XC2V1000-5	
Block RAM	16	10 %	16	40%
SLICES	4349	22%	4222	82%
MULT18x18	-	-	9	22 %
TBUFs	192	1%	192	7%
Clock Rate	35 MHz		50 MHz	
Performance	210 MFlops		300 MFlops	

Table 2: Resource utilization of the FI-CMA implementation with 19-bit HSLA library.

Handel-C language.

5 Experimental Results in FIC-CMA

5.1 Implementation Results

The FICMA algorithm has been implemented in two FPGA devices: Xilinx Virtex XCV2000E-6 and Xilinx Virtex-II XC2V1000-5. The results of the implementation are shown in Table 2. The design was implemented in the Celoxica DK3 suite using the Handel-C language. The Celoxica compiler for this language was used to get the standard VHDL description, which was compiled using the Simplicity's Simplify Pro to the EDIF format for electronic data interchange. Finally, the Xilinx ISE 6.1.03 tool was used to generate the bitstream for FPGA.

The choice of architecture to be used was made by construction of two models, one for HSLA (Figure 7) resulting in the period $w^* = 565$ and one for the FP32 resulting in the period $w^* = 1180$. Considering the clock cycle length (measured in the case of HSLA and estimated in the case of FP32) we were able to determine which architecture leads to faster implementation. Therefore, we have chosen HSLA architecture with 19-bit precision, which also saves memory.

The implementation was tested on a data matrix of the size 24×20 . The FI-CMA algorithm uses one twin-adder and four MUL units, which results in 300 MFlops for Virtex-II (210 MFlops for Virtex-E). One iteration of equalizer update procedure takes 565 cycles which represents the execution time $11\mu\text{s}$ for Virtex-II ($16\mu\text{s}$ for Virtex-E). This performance is sufficient for the symbol period $3.7\mu\text{s}$ (corresponding to 24 symbols processed in one batch in GSM systems), since the implementation of equalizer update is fast enough to perform 8 iterations, needed for the algorithm convergence.

5.2 Results of Scheduling

The presented scheduling technique was tested on an Intel Pentium 4 at 2.4 GHz using the non-commercial ILP solver GLPK [12] and the commercial ILP solver CPLEX [10]. The abstract model of the equalizer algorithm model in Figure 6 consists of 47 tasks on two distinct dedicated processors (29 on the first half of twin-

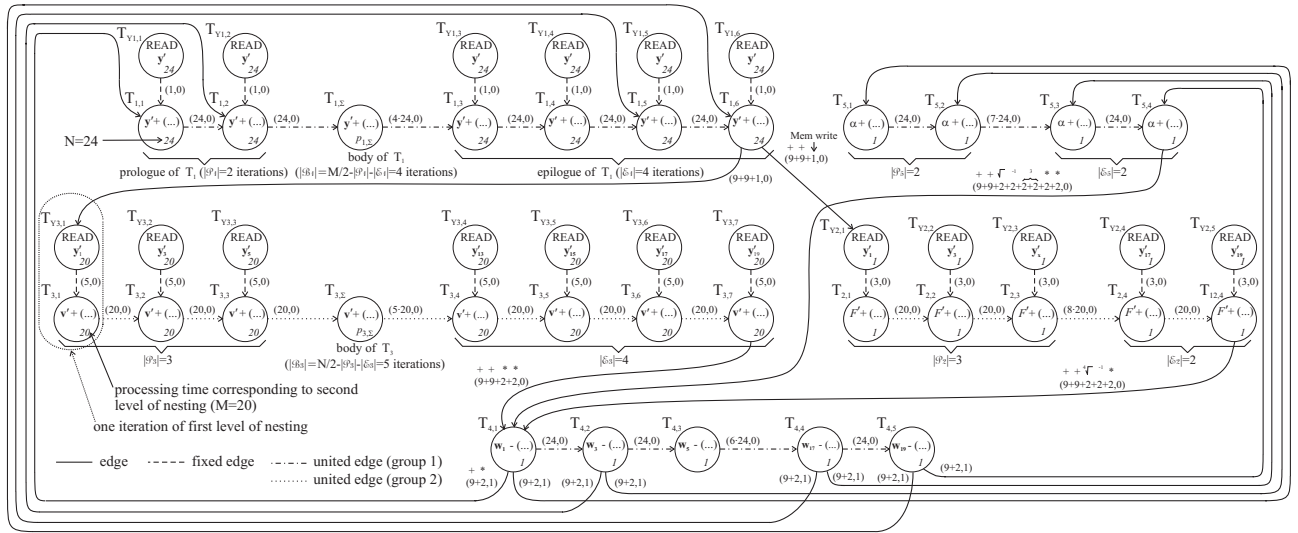


Figure 6: Model of the equalizer algorithm by the reduced graph G' with fixed edges and united edges on HSLA library.

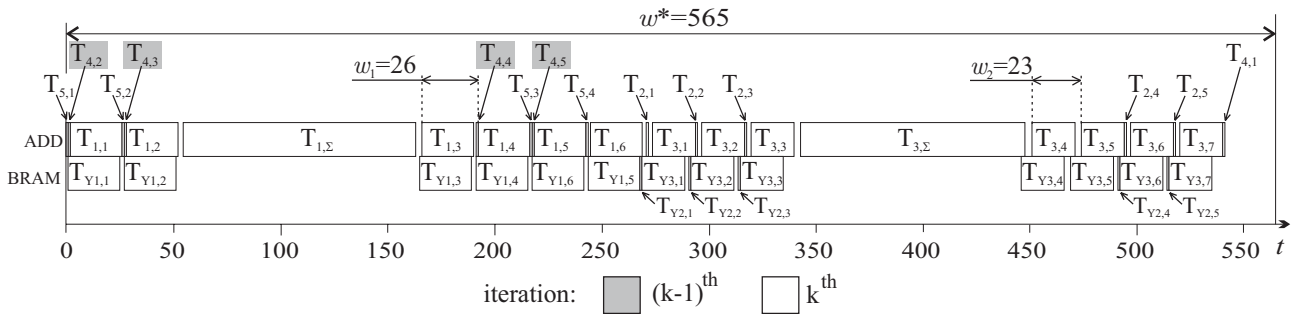


Figure 7: Resulting schedule of the equalizer algorithm ($w^* = 565$; inner periods $w_1 = 26$, $w_2 = 23$) on two dedicated processors (one half of HSLA twin-adder, and BRAM containing y').

adder and 18 on the dual ported Block RAM containing vector y' . The upper bound of \hat{q}_i was given *a priori* for tasks $T_{1,i}$ and $T_{5,i}$ equal to 0 and for $T_{3,i}$, $T_{2,i}$ and $T_{4,i}$ equal to 1. The lower bound of period w , $w_{lower} = 493$ is given by Equation 4. The upper bound $w_{upper} = 774$ was calculated using the ILP model (Figure 3) for the reduced condensed graph G'^c (in Figure 4(c)), where all tasks in the column loop are supposed to be executed simply, in sequence.

The resulting schedule with optimal period $w^* = 565$ was obtained by the interval bisection method in 9 iterative calls of the ILP. The corresponding ILP model contains 621 variables. The time required to compute the optimal solution without elimination of redundant processor constraints (see [18]), given as a sum of iterative calls of the ILP solver was 1634s with GLPK and 11s with CPLEX. This time does not include the construction of the ILP model, since it is negligible from the complexity point of view. The time required to compute the optimal solution with elimination of redundant processor constraints was 57.8s with GLPK and only 3.4s with CPLEX. In this

case, as many as 436 variables were eliminated (the number of eliminated redundant inequalities is twice as big), which is mainly due to the ILP model keeping a regular structure of matrix operations.

6 Conclusions

This paper presented a new scheduling method based on mathematical programming used to optimize the computation speed of real-time iterative algorithms with matrix operations running on architectures including distinct dedicated processors with restricted access into memory. The contribution lies in the representation of imperfectly nested loops and memory access in the form of linear inequalities that are incorporated in the cyclic scheduling framework based on ILP. In order to compare two different arithmetic libraries, we have extended our previous work in this paper, cyclic scheduling on one dedicated processor, to its multiprocessor version.

The method was used to find a schedule for FPGA implementation of the Finite Interval Constant Modulus AI-

gorithm (FI-CMA). The first implementation architecture for this algorithm, has been described in [9]. Using our ILP based scheduling, the number of cycles needed for one iteration of the equalizer update has been improved by approximately $24 \cdot M$ cycles. Therefore, the speedup was 46%.

7 Acknowledgement

This work been partially supported by the Grant Agency of the Academy of Sciences of the Czech Republic under Project 1ET300750402 and project 1ET400750406 and by the Ministry of Education of the Czech Republic under Project 1M0567.

References

- [1] N. Ahmed, N. Mateev, and K. Pingali. Tiling imperfectly-nested loop nests. In *Proceedings of the IEEE/ACM SC2000 Conference, Dallas, Texas*, November 2000.
- [2] S. Carr, C. Ding, and P. Sweany. Improving software pipelining with unroll-and-jam. In *Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS'96)*, January 1996.
- [3] Celoxica Ltd. *Platform Developers Kit: Pipelined Floating-point Library Manual*, 2004. <http://www.celoxica.com>.
- [4] A. Darte and Guillaume Huard. Loop shifting for loop compaction. *International Journal of Parallel Programming*, 28(5):499–534, 2000.
- [5] Dirk Fimmel and Jan Müller. Optimal software pipelining under resource constraints. *International Journal of Foundations of Computer Science*, 12(6):697–718, 2001.
- [6] D. N. Godard. Self-recovering equalization and carrier tracking in two-dimensional data communication systems. *IEEE Trans. Communications*, 28:1867–1875, November 1980.
- [7] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. Loop shifting and compaction for the high-level synthesis of designs with complex control flow. In *Design, Automation and Test in Europe Conference and Exhibition (DATE'04), Paris, France*, February 2004.
- [8] C. Hanen and A. Munier. A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, 57:167–192, February 1995.
- [9] A. Heřmánek, J. Schier, and P. A. Regalia. Architecture design for FPGA implementation of Finite Interval CMA. In *Proc. European Signal Processing Conference*, pages 2039–2042, Wiena, Austria, September 2004.
- [10] ILOG, Inc. *CPLEX Version 8.0*, 2002. <http://www.ilog.com/products/cplex/>.
- [11] M. Lam. Software pipelining: an effective scheduling technique for VLIW machines. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 318–328, 1988.
- [12] A. Makhorin. *GLPK (GNU Linear Programming Kit) Version 4.6*, 2004. <http://www.gnu.org/software/glpk/>.
- [13] R. Matoušek, M. Tichý, Z. Pohl, J. Kadlec, and C. Softley. Logarithmic number system and floating-point arithmetics on FPGA. In M. Glesner, P. Zipf, and M. Renovell, editors, *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, volume 2438 of *Lecture Notes in Computer Science*, pages 627–636, Berlin, 2002. Springer.
- [14] D. Petkov, R. Harr, and S. Amarasinghe. Efficient pipelining of nested loops:unroll-and-squash. In *16th International Parallel and Distributed Processing Symposium (IPDPS'02), Fort Lauderdale, California*, April 2002.
- [15] Z. Pohl, P. Šůcha, J. Kadlec, and Z. Hanzálek. Performance tuning of iterative algorithms in signal processing. In *The International Conference on Field-Programmable Logic and Applications (FPL'05), Tampere, Finland*, August 2005.
- [16] P. A. Regalia. A finite interval constant modulus algorithm. In *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP-2002)*, volume III, pages 2285–2288, Orlando, FL, May 13-17 2002.
- [17] S. L. Sindorf and S. H. Gerez. An integer linear programming approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays. In *PROGRESS 2000 Workshop on Embedded Systems, Utrecht, The Netherlands*, 2000.
- [18] P. Šůcha and Z. Hanzálek. Optimization of iterative algorithms with matrix operations:case studies. Technical report, CTU FEL DCE, Prague, 2005.
- [19] P. Šůcha, Z. Pohl, and Z. Hanzálek. Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004), Toronto, Canada*, 2004.
- [20] M. J. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.