

# Multi-Dimensional Interleaving for Synchronous Circuit Design Optimization

*Nelson Luiz Passos*

Dept. of Computer Science  
Midwestern State University  
Wichita Falls, TX 76308

*Edwin Hsing-Mean Sha*

Dept. of Computer Science & Engineering  
University of Notre Dame  
Notre Dame, IN 46556

*Liang-Fang Chao*

Dept. of Electrical and Computer Engineering  
Iowa State University  
Ames, Iowa 50011

## ABSTRACT

This paper presents a novel optimization technique for the design of application specific integrated circuits dedicated to perform iterative or recursive time-critical sections of multi-dimensional problems, such as image processing applications. These sections are modeled as cyclic multi-dimensional data flow graphs (MDFGs). This new optimization technique, called *multi-dimensional interleaving* consists of a multi-dimensional *expansion* and *compression* of the iteration space, followed by a multi-dimensional retiming, while considering memory requirements. It guarantees that all functional elements of a circuit can be executed simultaneously, and no additional memory queues proportional to the problem size are required. The algorithm runs optimally in  $O(|E|)$  time, where  $E$  is the set of edges of the MDFG representing the circuit. Our experiments show that the additional memory requirement is significantly less than the results obtained in other methods.

# 1 Introduction

The design of Application Specific Integrated Circuits (ASICs) is usually required in order to improve the execution performance of computation-intensive applications. A large group of such applications are multi-dimensional problems, i.e., problems involving more than one dimension, such as computer vision, high-definition television, medical imaging, and remote sensing. An important characteristic of these problems is their computational time-critical sections, which consist of iterative or recursive execution of sets of operations also known as loops. It is well known that a parallel implementation of such operations would improve the performance of the ASIC design.

Most of the previous research on loop parallelization has focused on one-dimensional problems [3, 4, 8, 11, 13, 15, 19, 27]. However, the performance improvement achievable by those one-dimensional methods is constrained by the number of delays (registers) in a cyclic data path [15]. In this paper, we model the loops, or iterations, as multi-dimensional data flow graphs (MDFGs). A novel multi-dimensional transformation, applicable to such MDFGs, is able to obtain the desired high level performance, while special consideration is given to the effects of this transformation on the memory requirements. The multi-dimensional characteristic of the problems is the foundation for the high parallelism achievable, usually superior to results obtained through traditional methods based on one-dimensional techniques.

Recent studies have considered the optimization of nested loops, a software point of view of the multi-dimensional problems [1, 2, 6, 18, 29, 30]. In general, these methods transform the loops in such a way to obtain a new sequence of execution characterized by a higher parallelism. This sequence of execution is commonly associated with a schedule vector  $s$ , also called an ordering vector, which affects the order in which the iterations are performed. The iterations are executed along hyperplanes defined by  $s$ . When the execution of a hyperplane reaches the boundaries of the iteration space, it advances to the next hyperplane according to the direction of  $s$ . In the area of high-level synthesis, researchers have focused on the optimization of multi-dimensional problems through the selection of an appropriate schedule vector [16, 22, 24, 25]. The new schedule vector usually differs from the one used in the original design, introducing new memory requirements that may end up in complex storage control and substantial increment on the memory size.

In a previous study, it has been shown that full parallelism can be obtained by the application of multi-dimensional retiming techniques [23]. However, that method required the use of additional memory queues<sup>1</sup> proportional to the size of the problem. In this study, we introduce an algorithm that optimizes the circuit design through the application of a multi-dimensional transformation technique which incorporates the multi-dimensional retiming method. This technique improves the parallelism by restructuring the iteration space and the loop body without changing the original schedule vector, and consequently, not requiring additional queues. The dimensions of the problem are associated with axes of a cartesian space, where integral

---

<sup>1</sup>The term *queue* will be used throughout this paper for queues with size dependent on the problem.

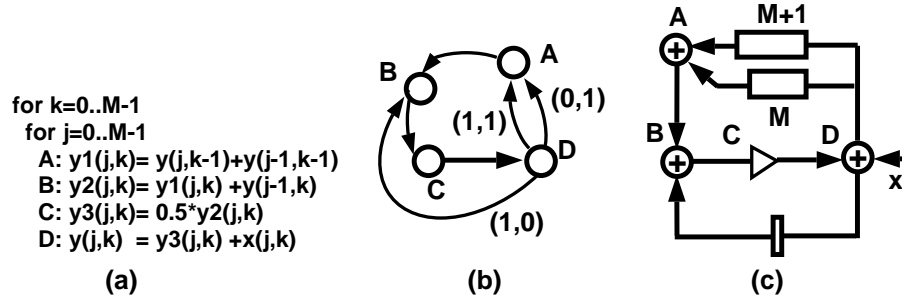


Figure 1: (a) code segment for a two-dimensional filter (b) MDFG representing the filter (c) equivalent circuit for a row-wise execution

points represent each iteration of the loop. In a two-dimensional (2-D) problem, such axes are also known as rows and columns. If the system is originally scheduled to be computed row by row, operations executed sequentially within one iteration due to data dependences require an unfolding or retiming technique to be parallelized [5, 15, 20]. The retiming limitations are well-known for one-dimensional problems. In a circuit design, data dependences between neighbor points in the row direction are translated into one register in the equivalent data path. This one register may restrict the parallelism of operations which belong to a cyclic path through that register. This constraint can jeopardize the chances of improving the parallelism of sequentially executed operations.

Leiserson and Saxe proposed a slow-down technique [15] in order to increase the number of delays in the circuit, and to achieve the desired retiming, so called systolic design. However, this technique also reduces the circuit performance. In this paper, an expansion of the iteration space is applied to improve the potential of parallelism, while a compression avoids any loss of performance. The combination of the expansion and compression of the iteration space is the basis for our optimization technique called *multi-dimensional interleaving*. A maximum throughput of one result per cycle time<sup>2</sup> is achievable after the multi-dimensional interleaving is applied.

For simplicity we use a two-dimensional problem aimed to a single processor system as an example. It consists of a two-dimensional filter [9], represented by the transfer function:

$$H(z_1, z_2) = \frac{1}{\left(1 - \sum_{n_1=0}^w \sum_{n_2=0}^w c(n_1, n_2) * z_1^{-n_1} * z_2^{-n_2}\right)}$$

where  $w = 1$ ,  $c(n_1, n_2) = .5, \forall n_1, n_2$ , which can be translated into

$$y(n_1, n_2) = x(n_1, n_2) + \sum_{k_1=0}^1 \sum_{k_2=0}^1 .5 * y(n_1 - k_1, n_2 - k_2), \text{ for } k_1, k_2 \neq 0, 1 \text{ or}$$

$$y(n_1, n_2) = x(n_1, n_2) + .5 * (y(n_1 - 1, n_2) + y(n_1, n_2 - 1) + y(n_1 - 1, n_2 - 1))$$

<sup>2</sup>The cycle time is assumed to be the longest execution time among all operations in the loop body

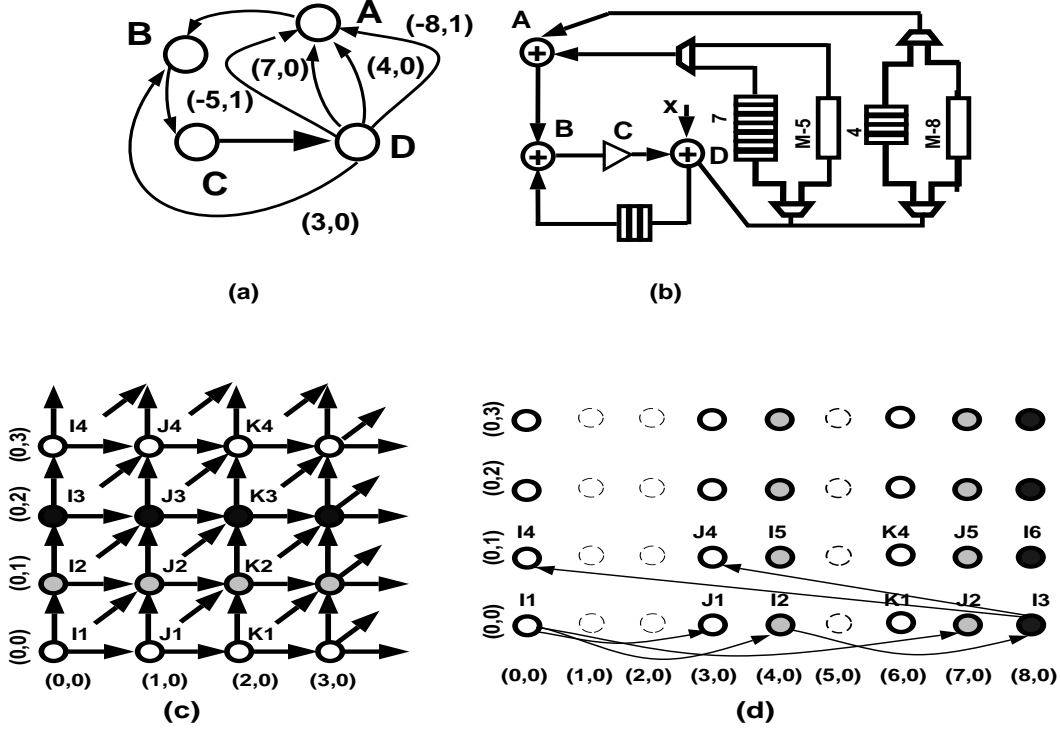


Figure 2: (a) transformed MDFG (b) sketch of the transformed digital circuit (c) original iteration space (d) iteration space after transformation

Figure 1(a) shows a segment of a program where four statements labeled  $A$ ,  $B$ ,  $C$  and  $D$  are computed inside of a doubly nested loop. This code is equivalent to the simulation of the filter described above. A multi-dimensional data flow graph representing this problem is shown in figure 1(b). Nodes represent operations and edges represent data dependences. The labels on the edges indicate the distance between iterations. Figure 1(c) shows a circuit design implementing the solution for this problem. In this example, we can observe the one-dimensional retiming constraint. For a row-wise execution, as established by the loop control variables, a register is placed between  $D$  and  $B$  to store data for the dependence  $(1, 0)$ . If we assume the problem size to be  $M \times M$  points, a queue of size  $M$  is used for the dependence  $(0, 1)$ , and one of size  $M + 1$  is used for the dependence  $(1, 1)$ . It is easy to notice that  $B$ ,  $C$  and  $D$  can not be executed in parallel by applying one-dimensional retiming techniques, due to the number of delays in the lower cycle. Using multi-dimensional retiming techniques, it is possible to overcome this problem by selecting a new schedule vector such as  $(1, 1)$ . However, this solution would require three queues of variable size with a maximum length  $M$ .

By applying our new technique, a combined expansion of the row-wise recursion and the compression of every three consecutive rows, we obtain the design shown in figure 2. In (a), we see the new MDFG with some additional edges. The new pairs of edges are translated into data paths containing multiplexers and demultiplexers as seen in figure 2(b). The reason for such

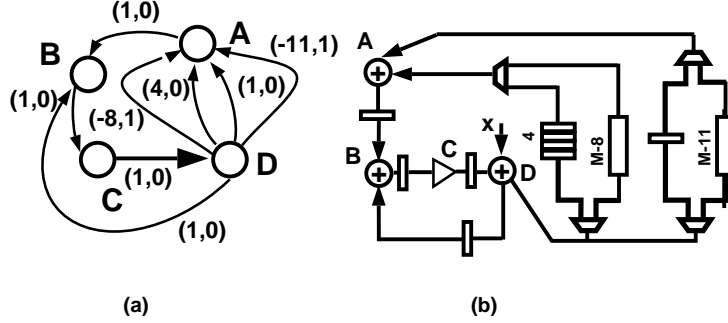


Figure 3: (a) transformed and retimed MDFG (b) sketch of the digital circuit

extra edges is easily explained by examining the original iteration space in figure 2(c), compared to figure 2(d) where the iteration space was expanded in the row-direction and compressed in the column direction. The movement of iteration  $I2$  to position  $(4,0)$  implies that the original dependence  $(0,1)$  from  $I1$  to  $I2$  is now  $(4,0)$  as well as the edge between  $I2$  and  $I3$ , while the dependence between  $I3$  and  $I4$  becomes  $(-8,1)$ . Intuitively, we notice that if  $I2$  had been moved to iteration  $(1,0)$ , the retiming of the upper cycle in order to obtain full parallelism would not be possible. Proceeding with the optimization process, a multi-dimensional retiming applied to this new MDFG results the desired fully parallel solution shown in figure 3. In this solution, we can observe that  $A$ ,  $B$ ,  $C$  and  $D$  can be executed in parallel. Multiplexors and demultiplexors are added to the design to obtain the fully parallel solution. The memory elements still have the same characteristics, i.e., only two queues. All other elements comprise fixed size queues not dependent on the problem. To evaluate the optimization achieved in this simple example, let us assume that this filter would be applied to a two-dimensional image of  $512 \times 512$  pixels. Considering the design implemented through the usage of standard CMOS cells where a multiplier has an execution time of 40 ns and an adder 20 ns, the total computation time for the original design would be approximately 26.2 ms. After the optimization, the total computation time is reduced to approximately 10.4 ms, which implies a gain of 60 % in the circuit performance. The total number of queues required by a fully parallel solution, obtained by using previously developed multi-dimensional retiming techniques, would be at least three, while as seen in the example, our optimized design requires only two queues. In a more general filter, covering a window of size  $(w + 1) \times (w + 1)$ , the number of additional queues required by the multi-dimensional retiming methods would be  $O(w^2)$ , while in our proposed method is zero.

The remaining of this paper presents the multi-dimensional interleaving technique beginning by showing basic concepts and terminology. In Section 3, we discuss the properties necessary to have an expansion of the iteration space in order to improve the potential for a fully parallel solution. Section 4 shows how to combine multiple rows of the iteration space to avoid any performance loss due to the expansion. Section 5 presents the main algorithm, showing how it affects the memory elements in the circuit. There follow examples of application of our technique. A final conclusion summarizes the concepts introduced in this paper.

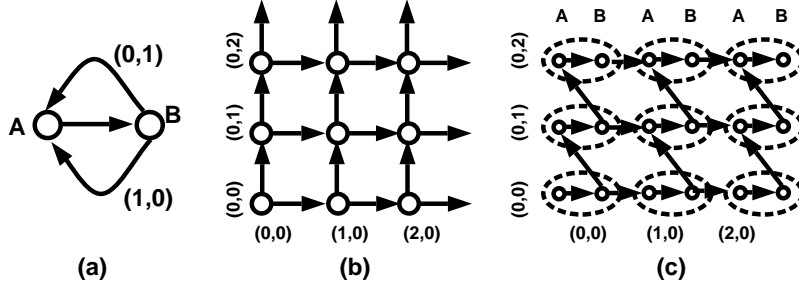


Figure 4: (a) a simple two nodes MDFG (b) the equivalent iteration space (c) a close look at the cells of the iteration space

## 2 Background

### 2.1 Modeling Multi-Dimensional Problems

A *multi-dimensional data flow graph (MDFG)*  $G = (V, E, d, t)$  is a node-weighted and edge-weighted directed graph, where  $V$  is the set of computation nodes, i.e., the functional elements in the circuit design,  $E$  is the set of dependence edges, equivalent to the circuit data paths,  $d$  is a function representing the multi-dimensional delays between two nodes, also known as dependence vectors, implicitly indicating the storage elements required in the circuit design, and  $t$  is a function representing the computation time of each node. An *iteration* is the execution of each node in  $V$  exactly once. We say that a design is *fully parallel* if all nodes in one iteration can be executed simultaneously. Iterations are represented as integral points in a cartesian space, called *iteration space*, where the coordinates are defined by the loop control indices. Such points are identified by a vector  $\hat{i}$ , equivalent to a multi-dimensional index. The components of  $\hat{i}$  are arranged from the innermost loop control index to the outermost one, always implying a row-wise execution. If a node  $v$  at iteration  $\hat{j}$ , depends on a node  $u$  at iteration  $\hat{i}$ , then there is an edge  $e$  from  $u$  to  $v$ , such that  $d(e) = \hat{j} - \hat{i}$ . Figure 4(b) shows a representation of the iteration space for the MDFG presented in figure 4(a). For simplicity, we will always show a small section of the iteration space with respect to our examples. Figure 4(c) is a magnification of the nodes in the iteration space in such a way that we can see the internal operations of each iteration. Notice that these figures of the iteration space will be used in the paper for illustration only.

We use the notation  $u \xrightarrow{e} v$  to indicate that  $e$  is an edge from node  $u$  to node  $v$ . The notation  $u \xrightarrow{p} v$  means that  $p$  is a path from  $u$  to  $v$ . The delay vector of a path  $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} v_2 \dots \xrightarrow{e_{k-1}} v_k$  is  $d(p) = \sum_{i=0}^{k-1} d(e_i)$  and the total computation time of a path  $p$  is  $t(p) = \sum_{i=0}^k t(v_i)$ . To manipulate MDFG characteristics represented in vector notation, such as the delay vectors, we make use of component-wise vector operations. Considering the

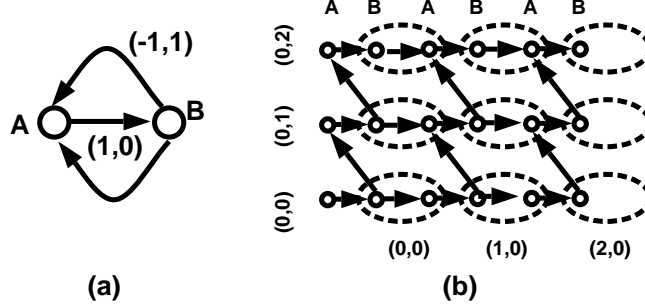


Figure 5: (a) MDFG after retimed by  $r(A)=(1,0)$  (b) equivalent iteration space

two-dimensional vectors  $P$  and  $Q$ , represented by their coordinates  $(P.x, P.y)$  and  $(Q.x, Q.y)$ , examples of arithmetic operations are  $P + Q = (P.x + Q.x, P.y + Q.y)$  and  $P \times Q = (P.x * Q.x, P.y * Q.y)$ . The notation  $P \cdot Q$  indicates the inner product between  $P$  and  $Q$ , i.e.,  $P \cdot Q = P.x * Q.x + P.y * Q.y$ . Vectors are ordered in a right-to-left lexicographic order, i.e., for two  $n$ -dimensional vectors  $P = (P_1, P_2, P_3, \dots, P_n)$  and  $Q = (Q_1, Q_2, Q_3, \dots, Q_n)$ ,  $P < Q$ , if for some  $1 \leq i \leq n$ ,  $P_i < Q_i$ , and  $\forall j > i, P_j = Q_j$ . For example,  $(1, 0, 0) < (0, -2, 1) < (0, 1, 1)$ . Vectors are also used to indicate the sequence of computation. In this paper, the sequence of execution is defined as a row-wise computation, i.e, iteration  $\hat{i}$  is executed before iteration  $\hat{j}$  if  $\hat{i} < \hat{j}$ .

## 2.2 The Multi-Dimensional Retiming

A *multi-dimensional retiming*  $r$  is a function from  $V$  to  $Z^n$  that redistributes the nodes in the iteration space created by the replication of an MDFG  $G = (V, E, d, t)$ . A new MDFG  $G_r = (V, E, d_r, t)$  is created, such that each iteration still has one execution of each node in  $G$ . The retiming vector  $r(u)$  of a node  $u \in G$  represents the offset between the original iteration containing  $u$ , and the one after retiming. The delay vectors change accordingly to preserve dependences, i.e.,  $r(u)$  represents delay components pushed into the edges  $u \rightarrow v$ , and subtracted from the edges  $w \rightarrow u$ , where  $u, v, w \in G$ . Therefore, we have  $d_r(e) = d(e) + r(u) - r(v)$  for every edge  $u \xrightarrow{e} v$ , and  $d_r(l) = d(l)$  for every cycle  $l \in G$ . After retiming, the execution of node  $u$  in iteration  $i$  is moved to the iteration  $i - r(u)$ . An example of a multi-dimensional retiming is presented in figure 5, where node  $A$  from figure 4(a) has been retimed by the vector  $(1, 0)$ . The retimed iteration space appears in figure 5(b).

An illegal multi-dimensional retiming occurs if the resulting iteration space presents cycles. Previous studies have proposed different methods on how to find a legal multi-dimensional retiming [21, 22, 23]. In this paper, by using multi-dimensional interleaving, we will show that the multi-dimensional retiming  $(1, 0, \dots, 0)$  is suffice as the basic retiming function to achieve full parallelism. The next section will show how the MDFG will be transformed to accommodate this retiming function without producing a cycle.

### 3 Expansion of the Iteration Space

Let us examine again the example in figure 4. We see that the data produced by  $A$  is immediately consumed by  $B$ . This condition implies a mandatory serial execution of these two operations. In order to improve the circuit performance, both operations should be executed in parallel. The simultaneous execution of those two functions requires a register or latch device between them. A well-known solution for this problem is to retime node  $A$  by one, i.e., to remove one register of each incoming edge (data path) of  $A$  and push it to the outgoing edges. This is called by us the *traditional one-dimensional retiming*. It is easy to verify that such a transformation does not solve the problem, since it produces a direct path between  $B$  and  $A$  in the lower cycle. In [23], it has been proven that a solution for this problem can be found by using a multi-dimensional retiming, associated with a new schedule vector. However, this new schedule vector may require additional memory queues. Therefore, we imposed as objective of our method, that the original schedule vector,  $(0, 1)$  in our example, should remain the same. To solve this new problem, we begin by defining a *expander function* that will provide the potential for the desired retiming.

**Definition 3.1** Given an MDFG  $G = (V, E, d, t)$  representing an  $n$ -dimensional problem, and an  $n$ -dimensional vector  $f_e$ , an *expander function* transforms  $G$  into a new MDFG  $G_e = (V, E, d_e, t)$  such that  $d_e = f_e \times d$ . The vector  $f_e$  is called the *expander coefficient*.

The expansion of the iteration space by the application of the expander function results in a new set of iterations larger than the existing number of points in the original problem. Therefore, some of the new iterations are not necessary, and we will consider them as if no computation was taking place at all. We say that these new iterations are *empty* or *inactive*, while the original ones are considered *valid* or *active*. For instance, if we apply the expander function with coefficient  $f_e = (2, 1)$  to the example in figure 4, we will obtain the MDFG presented in figure 6(a) with an equivalent iteration space shown in figure 6(b). In this new iteration space,  $(0, 0)$  and  $(0, 1)$  are examples of valid iterations, while  $(1, 0)$  and  $(1, 1)$  are empty. Intuitively, one can notice that the size of the memory elements in the corresponding design has doubled.

To compute the size of each memory element associated with the dependence vectors, we define a vector that represents the number of points in each of the dimensions of the problem, given by  $S = (S_1, S_2, S_3, \dots, S_n)$ . We also define an auxiliary function called *linearization*, which computes the vector  $L(S) = (1, S_1, S_1 S_2, S_1 S_2 S_3, \dots, \prod_{i=1}^{i=n-1} S_i)$ .

With those values, given an MDFG  $G = (V, E, d, t)$  being computed according to a row-wise sequence, with a size vector  $S$ , the size of the memory element with respect to a dependence  $d$  can be computed by  $M_d = d \cdot L(S)$ . After the expansion of the iteration space, the memory elements have an increased size. For an expander function with coefficient  $f_e$ , the memory elements of the transformed MDFG  $G_e = (V, E, d_e, t)$  will have size  $M_{d_e} = (f_e \times d) \cdot L(f_e \times S)$ .

With the increased size of the memory elements, the required retiming is no more constrained by the number of delays in a cycle since we can increase such number of delays as much as

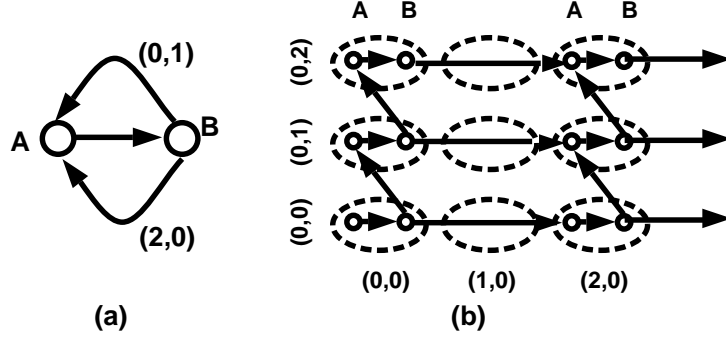


Figure 6: (a) MDFG after expansion (b) equivalent iteration space

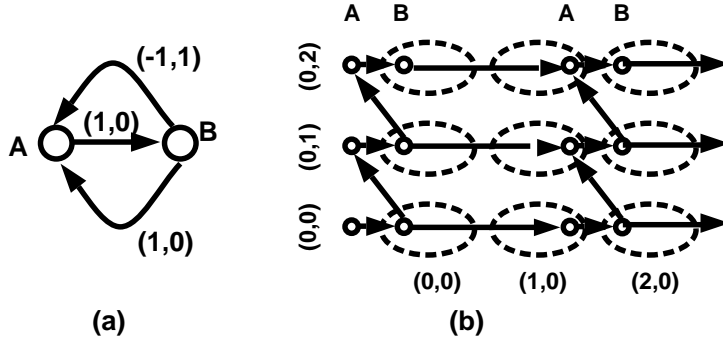


Figure 7: (a) MDFG after expansion and retiming (b) equivalent retimed iteration space

necessary. For instance, in our example in figure 6, each cycle has now two or more registers. Since the longest cycle has only two edges, a retiming function  $r(A) = (1,0)$  applied to the graph would produce the desired latch between the two operations  $A$  and  $B$ . Figure 7 shows the retimed MDFG and iteration space.

Considering that our final objective is to allow the usage of a multi-dimensional retiming, specifically  $(1, 0, 0, \dots, 0)$ , to obtain the parallelism among all operations in the circuit, and that there is the possibility of multiple cycles, we generalize the condition for the correct selection of the expander coefficient by the theorem below.

**Theorem 3.1** *Given an MDFG  $G = (V, E, d, t)$ , and an expander function with coefficient  $f_e$  transforming  $G$  in  $G_e = (V, E, d_e, t)$ , there exists a multi-dimensional retiming function such that  $d_e(e) \neq (0, 0, \dots, 0), \forall e \in E$ , if for each cycle  $l = v_1 \xrightarrow{1} v_2 \xrightarrow{2} \dots \xrightarrow{k-1} v_k \xrightarrow{k} v_1$ ,  $v_i \in V, 1 \leq i \leq k$ ,  $d_e(l) \geq (k, 0, 0, \dots, 0)$  (according to Section 2.1).*

*Proof:* Immediate from multi-dimensional retiming concepts and considering  $M_{d_e} = d_e(l) \cdot L(S_e)$ .  $\square$

However, the expander function has the side effect of lowering the efficiency of the circuit by introducing empty iterations. The real fully parallel solution, i.e., all operations executed in parallel per valid iteration, is not achieved, besides the latches required in each edge for a legal multi-dimensional retiming had been provided. In the next section, we show how to compensate such a loss by compressing the iteration space into a new form, while keeping the same global schedule vector.

## 4 The Compression Factor

As already mentioned, the expanded iteration space presents a loss of performance due to inactive iterations. In this section, we show how to recover such a loss by compressing several rows in one. In order to apply such a compression stage, we introduce the idea of a row-wise expansion, i.e., the use of an expander coefficient of the form  $f_e = (f, 1, 1, \dots, 1)$ . The row-wise expansion was chosen because it does not increase the number of queues while maintaining the row-wise sequence of execution, and it suffices to achieve the target fully parallel solution. We begin by the definition of an *expanded row cycle*, related to the row-wise expansion.

**Definition 4.1** Given an MDFG  $G$ , and an expander coefficient  $f_e = (f, 1, 1, \dots, 1)$  transforming  $G$  in  $G_e$ , an *expanded row cycle* (ERC) is the set of  $f$  consecutive iterations in the row direction, in  $G_e$ , such that the first iteration of the set is equivalent to one of the original iterations in  $G$  and the next  $f - 1$  are empty iterations. This first iteration is called *head* of the expanded row cycle.

Looking at figure 6(b), the first expanded row cycle consists of iterations  $(0, 0)$  and  $(1, 0)$ , with  $(0, 0)$  being the head of the ERC. After a row-wise expansion of the iteration space, we will apply the compression operation. This operation consists of two stages: an *iteration migration*, where the valid iterations are moved to empty positions in such a way to reduce the loss of performance, and a *column compression*, when rows containing only empty iterations are eliminated. The migration operation is then defined as below.

**Definition 4.2** Given an MDFG  $G_e$ , resulting from a row-wise expansion by the expander coefficient  $f_e = (f, 1, 1, \dots, 1)$ , applying the *migration operation* to row  $h$  moves valid iterations from rows  $h + 1$  through  $h + f - 1$  to the inactive iterations found in the ERCs of row  $h$ . We say that row  $h$  is a *target row*.

The iteration space will contain several target rows. One is particularly important for our method, the one that is used as the first target. Without loss of generality, we assume that such a row is the row zero, and that the lowest column index is also zero.

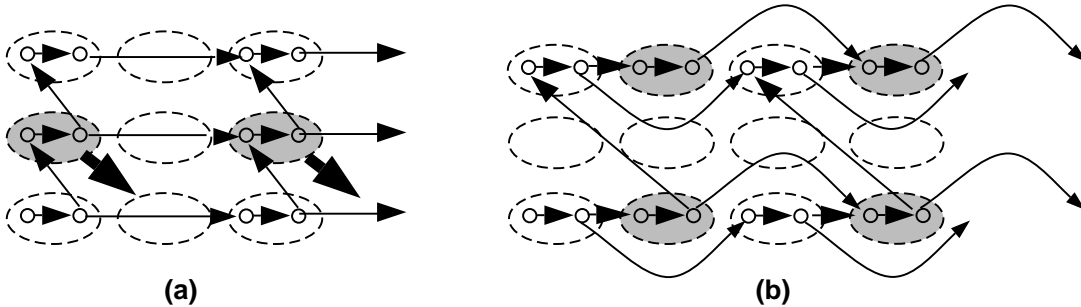


Figure 8: (a) expanded iteration space and iterations being moved down (b) iteration space after migration

Figure 8 presents a migration operation applied to the expanded iteration space shown in figure 6. In this example, the original iteration  $(0, 1)$  is being moved to the new iteration  $(1, 0)$ ; however, the result is not useful for our further optimization because some of the original dependences in the vertical direction became dependences in the row-direction in the transformed space and will restrict the application of the multi-dimensional retiming, i.e., the dependence  $(0, 1)$  became  $(1, 0)$  and  $(-1, 2)$ , and the new  $(1, 0)$  is a retiming restriction to the upper cycle of the MDFG. Another possible problem would happen if there was a dependence  $(-1, 1)$ , in the original unexpanded MDFG, that after compression would become  $(-1, 0)$ , creating a cycle in the iteration space, and conflicting with the row-wise execution sequence.

Intuitively, we know that after the migration there will be no inactive iterations in the target rows, except for an eventual start-up sequence, i.e., the loss of performance in those rows has been recovered. In order to avoid the problem of introducing a cycle and/or violating the execution sequence in the iteration space, we will migrate iterations in such a way that no dependences that conflict with the row-wise execution will be produced, as we did in the example of figure 8. For an expander coefficient  $f_e = (f, 1, 1, \dots, 1)$ , there will be  $f - 1$  rows to be mapped to a target row. To impose an uniform solution, we will map iterations from row  $h + 1$  to the first iteration after the head of an ERC at a target row  $h$ , and in a more general form, iterations from row  $h + \delta$  will be mapped to the  $\delta^{th}$  position after the head, for  $0 < \delta < f$ . To identify such ideal iterations, and where they will be mapped to, we define a *migration vector*.

**Definition 4.3** Given an expanded MDFG  $G_e = (V, E, d_e, t)$ , a *migration vector*  $c$  indicates the direction that valid iterations should follow to move to the target row.

Since our objective is to migrate iterations in the column direction, no changes in any other dimension should occur during the migration. Therefore, a migration vector with the form  $c = (g, -1, 0, \dots, 0)$  suffices to perform such mapping. The next lemma shows how to find the final position of some iteration after the expansion and migration stages.

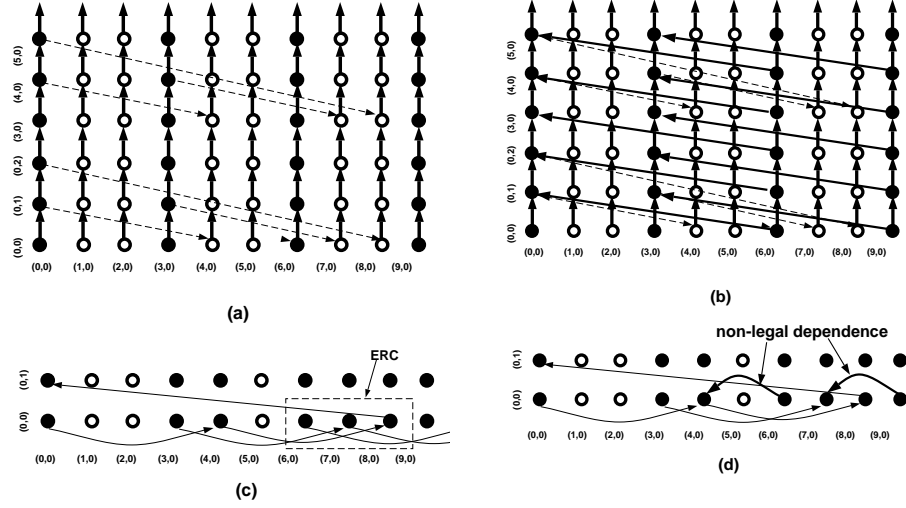


Figure 9: (a) An iteration space with dependences  $(0,1)$  expanded by coefficient  $(3,1)$  and iterations being moved (b) iteration space after migration with  $c = (4,-1)$  and some of the new dependences shown (c) additional dependence  $(-2,1)$  (d) after expansion and migration of  $(-2,1)$ , a cycle exists

**Lemma 4.1** *Given an MDFG  $G = (V, E, d, t)$ , submitted to an expander function with coefficient  $f_e = (f, 1, 1, \dots, 1)$ , and a migration vector  $c = (g, -1, 0, 0, \dots, 0)$ , an iteration  $P = (p_1, p_2, \dots, p_n)$ , is expanded and moved to a target row in position  $f_e \times P + \text{mod}(p_2, f) * c$ .*

Intuitively, we know that for a target row  $h$ , we have the heads of ERCs at positions  $(\gamma * f, h, x_3, \dots, x_n), \forall x_k, 3 \leq k \leq n$  and  $\gamma \geq 0$ . After migrating the first iteration from row  $h + 1$  to the first position after the head of an ERC at row  $h$ , we can determine the origin of the remaining  $f - 2$  iterations through the next lemma.

**Lemma 4.2** *Given an MDFG  $G = (V, E, d, t)$ , submitted to an expander function with coefficient  $f_e = (f, 1, 1, \dots, 1)$ , and a migration vector  $c = (g, -1, 0, 0, \dots, 0)$ , if a point  $P = (p_1, h+1, p_3, \dots, p_n)$  is expanded and moved to a target row  $h$  at position  $(\gamma * f + 1, h, p_3, \dots, p_n)$ , then  $(\gamma * f + \delta, h, p_3, \dots, p_n)$  is the mapping of a point  $Q = ((1 - \delta) * \gamma + \delta * p_1, h + \delta, p_3, \dots, p_n)$  for  $0 < \delta < f$ .*

In the example of figure 8, since we want to bring down iteration  $(0, 1)$  to position  $(1, 0)$ , the migration vector is  $c = (1, -1)$ . Figure 9(a) shows a different example, where a two-dimensional iteration space with the sole dependence  $(0, 1)$  is expanded by  $f_e = (3, 1)$  and has its rows migrated according to  $c = (4, -1)$ . Figure 9(b) shows the resulting iteration space after the migration operation. At this point, we will ignore any changes in the coordinates of

the second row of the resulting iteration space, which will be explained later in the column compression stage. In this example, the point  $(1, 1)$  has been mapped to position  $(7, 0)$ , i.e.,  $(3, 1) \times (1, 1) + (4, -1) = (7, 0)$ . From lemma 4.2, we can identify the third iteration  $Q'$  in that same ERC ( $\gamma = 2$ , since the ERC begins at iteration  $(6, 0)$ ) as expanded and migrated from  $Q = ((1 - 2) * 2 + 2 * 1, 2) = (0, 2)$ , which can be verified in the figure. When iterations are moved to new positions, their incoming dependences change accordingly. The next lemma shows what happens to dependences between the iterations in a target row  $h$  and those that migrated to  $h$ .

**Lemma 4.3** *Given an MDFG  $G = (V, E, d, t)$ , submitted to an expander function with coefficient  $f_e = (f, 1, 1, \dots, 1)$ , and a migration vector  $c = (g, -1, 0, 0, \dots, 0)$ , if an iteration  $P = (p_1, h + \delta, p_3, \dots, p_n)$ ,  $0 < \delta < f$  is expanded and migrates to a target row  $h$ , the incoming dependence  $d = (d_1, \delta, d_3, \dots, d_n)$  of  $P$ , from a node in row  $h$ , becomes  $d_m = (f * d_1 + \delta * g, 0, d_3, \dots, d_n)$*

*Proof:* Let us assume that iteration  $Q$  in row  $h$  produces the data required by  $P$ . Then,  $Q + d = P$ . This allows us to determine the coordinates of  $Q$  as  $Q = (p_1 - d_1, h, p_3 - d_3, \dots, p_n - d_n)$ . From lemma 4.1,  $P$  is expanded and mapped to  $P' = f_e \times P + \delta * c = (f * p_1 + \delta * g, h, p_3, \dots, p_n)$ . Iteration  $Q$  after expansion becomes  $Q' = f_e \times Q = (f * (p_1 - d_1), h, p_3 - d_3, \dots, p_n - d_n)$ . Therefore, the new dependence  $d_m$  can be computed by  $d_m = P' - Q' = (f * p_1 + \delta * g, h, p_3, \dots, p_n) - (f * (p_1 - d_1), h, p_3 - d_3, \dots, p_n - d_n) = (f * d_1 + \delta * g, 0, d_3, \dots, d_n)$ .  $\square$

In figures 9(a) and (b), when iteration  $(1, 1)$  ( $(3, 1)$  after expansion) is moved to position  $(7, 0)$ , the dependence  $d = (0, 1)$  from node  $(1, 0)$  ( $(3, 0)$  after expansion) becomes  $d_m = (3 * 0 + 1 * 4, 0) = (4, 0)$ . It was already mentioned that two valid iterations can not be expanded and migrated to the same position. We also know that the final iteration space can not have dependences contradicting the execution sequence, i.e., iterations depending on data to be produced in future iterations. For example, let us consider the dependence  $d = (-2, 1)$  in figure 9(c). Using the same expansion and migration applied to figure 9(a), this dependence would become  $d_m = (-2, 0)$  as seen in figure 9(d) which conflicts with the sequence of execution, and also creates a cycle with the transformed dependence  $(4, 0)$ . This implies that we need to define a *legal* migration vector, which is done as follows:

**Definition 4.4** *Given an MDFG  $G = (V, E, d, t)$ , expanded by an expander coefficient  $f_e = (f, 1, 1, \dots, 1)$ , a *legal migration vector* produces a new iteration space for  $G$ , transforming the dependences  $d$  in  $d_m$ , such that no two valid iterations are assigned to the same position, no cycles are created in the row direction, and the execution sequence is not violated, i.e.,  $d_m \geq (1, 0, \dots, 0)$  according to the right-to-left lexicographic order explained in Section 2.*

The next theorem shows how to select a legal migration vector.

**Theorem 4.4** Given an MDFG  $G = (V, E, d, t)$ , submitted to an expander function with coefficient  $f_e = (f, 1, 1, \dots, 1)$ , a compression vector  $c = (g, -1, 0, 0, \dots, 0)$  is legal if  $g$  is computed by:

- (a)  $g = 1$  if there is no  $d = (d_1, d_2, 0, 0, \dots, 0)$ , such that  $0 < d_2 < f$ .
- (b)  $g = \gamma * f - f * p + 1$ , for  $\gamma \geq 0$  and  $p = \min \left\{ 0, \left\lfloor \frac{d_1}{d_2} \right\rfloor, \forall d(e) = (d_1, d_2, 0, 0, \dots, 0), 0 < d_2 < f \right\}$ .

*Proof:* For part (a), there are two cases:

Case 1:  $d_2 = 0$  or  $d_2 \geq f$  for all  $d(e) = (d_1, d_2, 0, 0, \dots, 0)$ . This implies that the iterations migrated according to  $c = (1, -1, 0, 0, \dots, 0)$  do not depend on data produced in the target row. Therefore, there will be no cycle.

Case 2: Assume  $P$  is migrating to a target row  $h$  and  $d(e) = (d_1, d_2, \dots, d_n)$  with  $d_k > 0$  for some  $3 \leq k \leq n$ . We must make sure that for any incoming dependence  $d$  to  $P$ , originated in  $h$ , the transformed dependence  $d_m$  will be greater than or equal to  $(1, 0, \dots, 0)$ . According to lemma 4.3, the transformed  $d_m$  does not change  $d_k$ . Therefore  $d_m > (1, 0, \dots, 0)$ , and the condition is satisfied.

Choosing  $g = 1$ , the iteration  $P$  is mapped to a target row  $h$  at position  $P' = f_e \times P + \delta * c$ . We know that  $f * p_1 < f * p_1 + \delta < f * p_1 + f$ , or  $f * p_1 < f * p_1 + \delta < f * (p_1 + 1)$ . Therefore  $P'$  is located between two consecutive valid iterations where  $p_1 = i_1$ , which is the position of an empty iteration. For part(b), from lemma 4.3,  $d_m = (f * d_1 + \delta * g, 0, \dots, 0)$ . Replacing  $g$ , and assuming that  $\frac{d_1}{d_2} < 0$ ,  $d_m \geq (f * d_1 - \delta * f * \frac{d_1}{d_2} + 1, 0, \dots, 0)$ . Replacing  $\delta$  by  $d_2$ ,  $d_m \geq (f * d_1 - f * d_1 + 1, 0, \dots, 0) \geq (1, 0, \dots, 0)$ , and no cycles exist. From lemma 4.1,  $P$  is mapped to  $P' = f_e \times P + \delta * c = (f * p_1 + \delta * g, h, p_3, \dots, p_n)$ . Replacing  $g$ ,  $(f * (p_1 - \delta * p + \gamma * \delta), h, p_3, \dots, p_n) < P' < (f * (p_1 - \delta * p + \gamma * \delta + 1), h, p_3, \dots, p_n)$ . Using (1) we conclude that  $P'$  is inside an ERC, and according to lemma 4.2, it does not conflict with any other valid iteration.  $\square$

We notice that for an expander coefficient  $f_e = (f, 1, 1, \dots, 1)$ , after migration, there will be  $f - 1$  rows without any valid iteration in between the target rows. Since these rows are not useful for the desired computation, we remove them by a *column compression*. As a result of the column compression, an iteration  $P = (p_1, h, p_3, \dots, p_n)$  in a target row  $h = \alpha * f$ ,  $\alpha \geq 0$ , in  $G'$ , will have a new index  $P' = (p_1, \alpha, p_3, \dots, p_n)$ .

Examining again the examples in figure 9, we notice that after migration and column compression, the target rows  $0, 3, \dots$  became  $0, 1, \dots$ , and also that there is a non-regularity on the dependence vectors at every 3 iterations in the row direction. In figure 9(b), we have the new dependence  $(4, 0)$  for the the first two iterations in the ERC (i.e.,  $(6, 0)$  and  $(7, 0)$ ), and  $(-8, 1)$  for the last iteration (i.e.,  $(8, 0)$ ). To model this non-regularity, we introduce the concept of an MDFG associated with an activation mechanism that will provide the necessary information on when to use the dependence vectors. We call this new model a *Multiplexed Multi-Dimensional Flow Graph (MMDFG)*. The two definitions below are used to characterize the MMDFG.

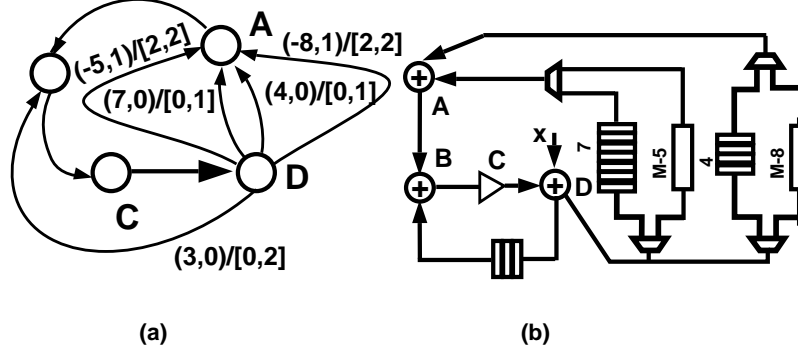


Figure 10: (a) The MMDFG representing the 2D filter. Notice the pair of labels in each edge, indicating the actual dependence, followed by the activation function (b) the equivalent circuit

**Definition 4.5** An edge  $u \xrightarrow{e} v$  is said *active* at iteration  $\hat{j}$  when the data produced by  $u$  during the execution of  $\hat{j}$  is transmitted to  $v$  by the edge  $e$ .

**Definition 4.6** A *multiplexed multi-dimensional flow graph (MMDFG)*  $G_m = (V, E, d, t, a)$  is an MDFG with an additional function  $a$  from  $E$  to  $Z^2$ , representing the range of iterations during the execution of an ERC when the edge is active, i.e, if the ERC executes  $f$  iterations,  $a(e) = [t_0, t_1]$  implies that the edge  $e$  is active from iteration  $t_0$  until iteration  $t_1$  such that  $0 \leq t_0 \leq t_1 \leq f$ .

Figure 10 shows the MMDFG and the circuit equivalent to the example of figure 1 after expansion by  $f = (3, 1)$ , migration according to  $c = (4, -1)$ , and column compression. The ERC executes in the time equivalent to three iterations, which implies that the range of the activation function will begin at time 0 and end at time 2. Therefore, the activation functions are: for  $d(e1) = (3, 0)$ ,  $a(e1) = [0, 2]$ ,  $d(e2) = (4, 0)$ ,  $a(e2) = [0, 1]$ ,  $d(e3) = (-8, 1)$ ,  $a(e3) = [2, 2]$ ,  $d(e4) = (7, 0)$ ,  $a(e4) = [0, 1]$ , and  $d(e5) = (-5, 1)$ ,  $a(e5) = [2, 2]$ . The activation function is represented by a second label on each edge of the graph, as shown in figure 10(a). When implementing the design equivalent to an MMDFG in a single processor, the activation function is translated in multiplexers and demultiplexors as shown in figure 10(b). The lemma below shows how to compute the new dependence vectors and activation functions for each edge after the column compression operation.

**Theorem 4.5** Given an MDFG  $G = (V, E, d, t)$ , submitted to an expander function with coefficient  $f_e = (f, 1, 1, \dots, 1)$ , a migration vector  $c$  with the form  $c = (g, -1, 0, 0, \dots, 0)$ , and a column compression, producing the MMDFG  $G_m = (V, E, d_m, t, a)$ , the dependence vectors and activation functions obtained from an original dependence  $d = (d_1, d_2, d_3, \dots, d_n)$  are given by: (a) if  $\gcd(d_2, f) = f$  then  $d_m = (f \times d_1, \frac{d_2}{f}, d_3, \dots, d_n)$  and  $a = [0, f - 1]$ .

(b) if  $\gcd(d_2, f) \neq f$  two new edges replace the original one s.t.  $d_m$  has the form

$(d_{m_1}, d_{m_2}, d_3, \dots, d_n)$  and:

1)  $d_{m_1} = f * d_1 + \text{mod}(d_2, f) * g$  and  $d_{m_2} = \lfloor \frac{d_2}{f} \rfloor$  with  $a = [0, f - \text{mod}(d_2, f) - 1]$  and

2)  $d_{m_1} = f * d_1 + (\text{mod}(d_2 - 1, f) - (f - 1)) * g$  and  $d_{m_2} = \lceil \frac{d_2}{f} \rceil$  with  $a = [f - \text{mod}(d_2, f), f - 1]$

*Proof:* Consider an iteration  $P = (p_1, \delta, p_3, \dots, p_n)$  in some row  $\delta$ , such that  $0 \leq \delta < f$ , being expanded and migrating to a target row. From lemma 4.1,  $P$  becomes  $P' = (f * p_1 + \delta * g, 0, p_3, \dots, p_n)$ . An iteration  $Q$  depending on  $P$  by the dependence  $d$  can be computed by  $Q = P + d = (p_1 + d_1, \delta + d_2, p_3 + d_3, \dots, p_n + d_n)$ . After transforming  $P$  and  $Q$  we compute  $d_m = P' - Q' = (f * d_1 + \text{mod}(\delta + d_2, f) * g - \delta * g, \frac{\delta + d_2 - \text{mod}(\delta + d_2, f)}{f}, d_3, \dots, d_n)$  (1). For part (a),  $\gcd(d_2, f) = f$  implies  $d_2 = \alpha * f$  for some integer  $\alpha$ . By transformations on (1),  $d_m = (f * d_1, \frac{d_2}{f}, d_3, \dots, d_n)$ . Since there is a unique transformed dependence, the activation function spans all iterations in the ERC, i.e.,  $a = [0, f - 1]$ . For part (b),  $\gcd(d_2, f) \neq f$  implies  $d_2 = \alpha * f + \beta$  for  $\alpha, \beta$  integers and  $1 \leq \beta < f$ . From (1),  $d_m = (f * d_1 + \text{mod}(\delta + \beta, f) * g - \delta * g, \frac{\delta + \alpha * f + \beta - \text{mod}(\delta + \beta, f)}{f}, d_3, \dots, d_n)$ . For  $1 \leq \delta + \beta < f$ , then  $d_m = (f * d_1 + \beta * g, \frac{\alpha * f}{f}, d_3, \dots, d_n)$ . This solution is valid only for rows  $\delta$  such that  $0 \leq \delta < f - \text{mod}(d_2, f)$ , which provides the activation function  $a = [0, f - \text{mod}(d_2, f) - 1]$ . For  $f \leq \delta + \beta < 2 * f$ , then  $d_m = (f * d_1 + \delta * g + \beta * g - f * g - \delta * g, \frac{\delta + \alpha * f + \beta - \delta - \beta + f}{f}, d_3, \dots, d_n)$ , and this solution is valid only for rows  $\delta$  such that  $f - \text{mod}(d_2, f) \leq \delta < f$ , which provides the activation function  $a = [f - \text{mod}(d_2, f), f - 1]$ .  $\square$

From theorem 4.5, we notice that the new set of dependences in the row-direction, created by the transformation of vertical dependences, may affect the final multi-dimensional retiming by introducing new restrictions on the number of delays in a cycle. In figure 10, the original dependence (0, 1) produced the new dependence (4, 0) in a cycle containing exactly 4 edges, which implies that the new dependence satisfies the desired retiming. However, this is not always true. Therefore, in order to adjust our solution to verify this new problem, we need to rewrite theorem 3.1 to accommodate this new model.

**Theorem 4.6** *Given an MDFG  $G$ , submitted to an expander function with coefficient  $f_e$ , a migration vector  $c$ , and a column compression, transforming  $G$  in the MMDFG  $G_m = (V, E, d_m, t, a)$ , there exists a multi-dimensional retiming function such that*

$d_m(e) \neq (0, 0, \dots, 0), \forall e \in E$ , if for any cycle  $l = v_1 \xrightarrow{1} v_2 \xrightarrow{2} \dots \xrightarrow{k-1} v_k \xrightarrow{k} v_1$ ,  $v_i \in V, 1 \leq i \leq k$ ,  $d_m(l) \geq (k, 0, 0, \dots, 0)$ .

Considering these new concepts, in the next section we introduce the algorithm that will produce the final parallel solution using, as input data, the MDFG describing the problem.

## 5 The Multi-Dimensional Interleaving Algorithm

As denoted by theorems 3.1 and 4.6, the correct choice of the expander coefficient and the migration vector is the key for obtaining the multi-dimensional retiming in the row direction such that all operations can be executed in parallel. We also know that a modified single-source shortest path algorithm measuring the one-dimensional distance of all nodes in the graph to an arbitrary inserted node is one approach to verify the non-existence of negative cycles while computing the retiming function [7, 15, 26]. In our study, we reduce the problem of finding the expander coefficient, the migration vector and the retiming function to an algorithm based on a modified topological sort of the MDFG combined with the formulation developed in the previous sections. In order to obtain such an algorithm, we define the *Multi-Dimensional Interleaving* transformation as the changes in the structure of the iteration space represented by an MDFG  $G$  producing an MMDFG  $G_m$  which has its row-wise dimension expanded by a coefficient  $f$  and its column-wise dimension compressed by  $f$ .

The combination of multi-dimensional interleaving and multi-dimensional retiming allows us to obtain fully parallel solutions for the multi-dimensional problems, without changing the initial global execution schedule. The algorithm applying such a combination is called *MdIntRet* for *Multi-Dimensional Interleaving and Retiming*, and it is shown below:

---

**MdIntRet( $G$ )**

```

 $G' \leftarrow G$  /* save G */
/* execute a topological sort on G */
 $LEV(u \in V) \leftarrow TOPSORT(G)$ 
/* find f such that dependences in the row direction do not affect the retiming */
 $\forall u \xrightarrow{e} v \in E$  s.t.  $d(e) = (d_1, 0, 0, \dots, 0) \neq (0, 0, \dots, 0)$ 
   $f \leftarrow \max \left\{ \left\lceil \frac{LEV(v) - LEV(u) + 1}{d_1} \right\rceil \right\}$  /* according to theorem 5.1 */
/* find p according to theorem 4.4 */
 $\forall d(e) = (d_1, d_2, 0, 0, \dots, 0), 0 < d_2 < f, e \in E$ 
   $p \leftarrow \min \left\{ 0, \left\lfloor \frac{d_1}{d_2} \right\rfloor \right\}$ 
/* find g such that the compressed iterations do not affect the retiming */
 $\forall u \xrightarrow{e} v \in E$  s.t.  $d(e) = (d_1, d_2, 0, \dots, 0), 0 < d_2 < f$ 
   $\gamma \leftarrow \max \left\{ 0, \left\lceil \frac{LEV(v) - LEV(u) + 1 - f \cdot d_1 - d_2 + d_2 \cdot f \cdot p}{d_2 \cdot f} \right\rceil \right\}$ 
 $g \leftarrow \gamma \cdot f - f \cdot p + 1$ 
/* compute  $G_m$  according to theorem 4.5 */
 $G_m \leftarrow TransformedG$ 
/* compute the retiming function for each node */
 $\forall v \in V$ , compute  $r(v) = LEV(v) \times (1, 0, \dots, 0)$ 

```

---

In the first step, a modified topological sort is used to order the nodes in levels, stored in the vector  $LEV$ , in  $O(|E|)$  time. Every node is assigned to a unique negative level number

equivalent to the results obtained by the single-source shortest path algorithm if each non-zero delay edge had infinite delays. Each node is labeled according to its level number, which produces a monotonically decreasing characteristic of the node indices in any path in the graph. The absolute value of the difference between any two nodes connected by multiple zero-delay paths  $b_i$ , where  $i \geq 1$ , represents the length of the longest zero-delay path between those two nodes. This characteristic produces the coefficients for the multi-dimensional retiming function. As mentioned before, the expander coefficient must be chosen large enough in order to allow a fully parallel solution after retiming the cycles with delays in the row-direction. Later, the migration vector must be chosen such that no cycles are created in the row direction and the transformation of dependences to the row direction does not interfere with the desired retiming function. In order to satisfy such conditions, the formulation used for computing the expander coefficient  $f_e$  and the migration vector  $c$  is obtained from the theorem below.

**Theorem 5.1** *Given an MDFG  $G = (V, E, d, t)$  submitted to an expander function with coefficient  $f_e = (f, 1, \dots, 1)$ , and a migration vector  $c = (g, -1, 0, \dots, 0)$ , resulting in an MMDFG  $G_m = (V, E, d_m, t, a)$ , there exists a multi-dimensional retiming function such that  $d_m(e) \neq (0, 0, \dots, 0), \forall e \in E$ , i.e., there exists a fully parallel solution for  $G$ , if*

(a)  $f = \max \left\{ \left\lceil \frac{LEV(v) - LEV(u) + 1}{d_1} \right\rceil \right\}, \forall d(e) = (d_1, 0, 0, \dots, 0)$  and  $d_1 > 0$

(b)  $g = \gamma * f - f * p + 1$ , for  $\gamma = \max \left\{ 0, \left\lceil \frac{LEV(v) - LEV(u) + 1 - f * d_1 - d_2 + d_2 * f * p}{d_2 * f} \right\rceil \right\}$  and  $p = \min \left\{ 0, \left\lfloor \frac{d_1}{d_2} \right\rfloor \right\}, \forall u \xrightarrow{e} v$  s.t.  $d(e) = (d_1, d_2, 0, \dots, 0), 0 < d_2 < f_2$ .

*Proof:* Let us assume  $d(e) = (d_1, d_2, \dots, d_n)$  for some edge  $u \xrightarrow{e} v$ . If there is one or more paths between  $v$  and  $u$  creating a cycle containing  $e$ , then the absolute value of the difference between the levels of  $u$  and  $v$  indicates the length of the longest cycle minus one (edge  $e$ ). Therefore,  $LEV(v) - LEV(u) + 1$  is the total length of the cycle containing  $e$ . If  $LEV(v) < LEV(u)$ , then  $e$  is a forward edge in the graph, and no cycle exists, in this case no retiming is necessary, and the length of the cycle can be interpreted as zero length. If  $LEV(v) > LEV(u)$ , then  $e$  is a feedback edge, which implies, from theorems 3.1 and 4.6, that the minimum required multi-dimensional delay in that edge must be  $d_{min} = (\beta, 0, \dots, 0)$  where  $\beta = LEV(v) - LEV(u) + 1$ . Considering the new dependences according to theorem 4.5, it is easy to show that they will be greater than the retiming requirement.  $\square$

## 6 Examples

In this section we present the application of our method to multi-dimensional applications commonly found in the image processing area. The first example consists of a two-dimensional *differential pulse-code modulation* (DPCM) device, used in image data compression [12]. It can be represented by the equations:

$$u_{i,j}^p = a_1 u_{i-1,j}^r + a_2 u_{i,j-1}^r - a_3 u_{i-1,j-1}^r$$

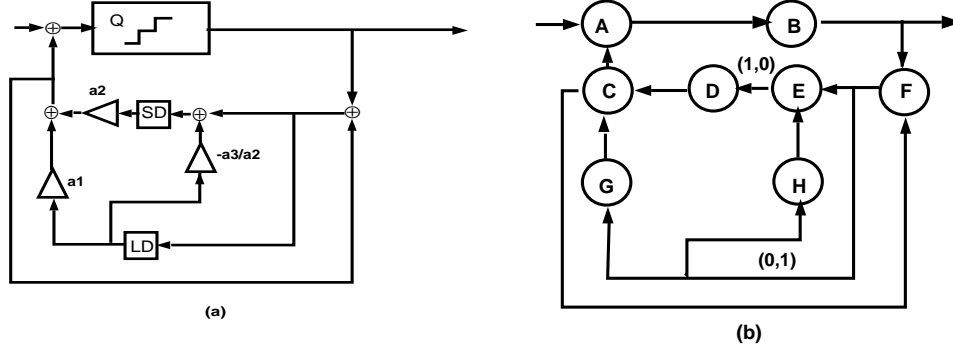


Figure 11: (a) circuit design of the differential pulse-code modulation device (b) equivalent MDFG

$$\begin{aligned}
 e_{i,j} &= u_{i,j} - u_{i,j}^p \\
 u_{i,j}^r &= u_{i,j}^p + e_{i,j}^q \\
 e_{i,j}^q &= Q(e_{i,j})
 \end{aligned}$$

The initial circuit design is shown in figure 11(a) and its equivalent MDFG is in figure 11(b). The algorithm *MdIntRet* computes  $LEV(D) = LEV(G) = LEV(H) = 0$ ,  $LEV(C) = -1$ ,  $LEV(A) = -2$ ,  $LEV(B) = -3$ ,  $LEV(F) = -4$ ,  $LEV(E) = -5$ . The expander coefficient is obtained from edge  $E \rightarrow D$ ,  $f = \frac{LEV(D) - LEV(E) + 1}{1} = \frac{0 - (-5) + 1}{1} = 6$ . The selection of the migration vector  $c$  is preceded by the choice of  $p = 0$  due to the dependence  $(0, 1)$ , and the exam of edges  $F \rightarrow H$  and  $F \rightarrow G$ . Since  $G$  and  $H$  have the same level, we can compute  $\gamma$  by  $\gamma = \left\lceil \frac{LEV(G) - LEV(F) + 1 - 6 * 0 - 1 + 1 * 6 * 0}{1 * 6} \right\rceil = \left\lceil \frac{0 - (-4) + 1 - 6 * 0 - 1 + 1 * 6 * 0}{6} \right\rceil = 1$ . Therefore,  $g = 6 * 1 - 6 * 0 + 1 = 7$ . After applying the multi-dimensional interleaving with expander coefficient  $(6, 1)$  and migration vector  $(7, -1)$ , the edge  $(1, 0)$  becomes  $(6, 0)$ , while the edge  $(0, 1)$  produces the new dependences  $(7, 0)$  and  $(-35, 1)$ . We proceed with the application of the multi-dimensional retiming function  $r(A) = (-2, 0)$ ,  $r(B) = (-3, 0)$ ,  $r(C) = (-1, 0)$ ,  $r(E) = (-5, 0)$ , and  $r(F) = (-4, 0)$ . The final design is shown in figure 12. It is easy to verify that the execution time of the new design is equivalent to  $1/6$  of the original one if each operation is assumed to be executed in one time unit. A memory improvement of 50 % is achieved if this design is compared to one submitted to the multi-dimensional retiming proposed in [23].

Another example consists of a two-dimensional filter design presented in [10, 23, 31]. It is defined by the transfer function:

$$H(z_1, z_2) = \frac{\sum_{i=0}^2 \sum_{j=0}^2 f(i,j) * z_1^{-i} * z_2^{-j}}{\sum_{i=0}^2 \sum_{j=0}^2 g(i,j) * z_1^{-i} * z_2^{-j}}$$

The design is shown in figure 13. Its equivalent MDFG is shown in figure 14. Using the algorithm *MdIntRet*, we begin by executing the topological sort, which produces the following values for the level of each node: nodes  $M_i$ ,  $1 \leq i \leq 9$ , are assigned to level 0, nodes  $A, B, C, D, F, G$  to -1, nodes  $E$  and  $H$  to -2, node  $J$  to -3, node  $K$  to -4,  $M_j$ ,  $10 \leq j \leq 17$ , to -5,  $L, N, P, Q, R$  to -6 and finally,  $S$  to -7. The expander coefficient is determined from

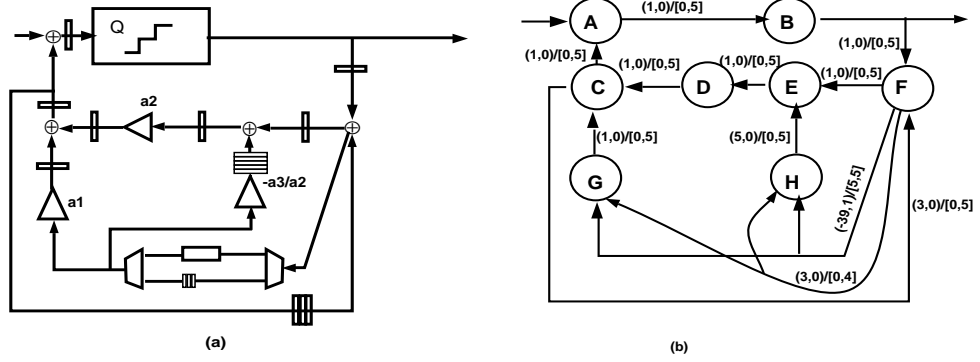


Figure 12: (a) new circuit design of DPCM (b) equivalent MMDFG

the edge  $S \rightarrow J$ ,  $f = \left\lceil \frac{-3 - (-7) + 1}{1} \right\rceil = 5$ . When computing the first iteration to be migrated, we find  $p = 0$ , i.e., the expanded iteration  $(0, 1)$  is expected to move to the empty iteration  $(\gamma * f + 1, 0)$ . The value of  $\gamma$  is computed according to the edge  $L \rightarrow K$ , and results 1, implying  $g = 6$ . Therefore, the migration vector is  $c = (6, -1)$ . The dependences are then transformed as shown in table 1. If we assume the circuit design is implemented by using CMOS standard cell technology, available in Mentor Graphics CAD tools [17], the multiplier will execute in 40 ns while the adder requires only 20 ns. Considering such execution times, the final fully parallel graph and the modified circuit design improve the execution time obtained in [10], from 60 ns to 40ns. Assuming also that the number of computational points is  $M \times M$ , and comparing our results with those obtained by using the *chained MD retiming* proposed in [23], we notice that both are able to achieve the fully parallel solution, however, the memory requirement has been reduced from 16 queues in [23] to 12 queues in this paper. Table 1 shows some other interesting results. To interpret such data, consider  $d(e)$  as the original dependence vector of edge  $e$ ,  $d_e(e)$  as the expanded dependence,  $d_m(e)$  as the interleaved dependence,  $d_r(e)$  as the dependence after retiming. The column *orig* presents the original memory requirements, while *MdIntRet* shows the required memory for our proposed method, *chained* shows the results of the *chained MD retiming* [23], the column *hyp* shows the requirements imposed by methods based solely on the selection of a new schedule or ordering vector, as in the unimodular transformations [2, 30], loop skewing [29], and other traditional wavefront methods [1, 14]. The last column, *aff*, presents results that could be obtained by modifying *affine-by-statement* methods developed for systolic arrays [16, 24, 25], and other methods focused on fine-grain parallelism that also depend on the selection of a new schedule such as the *schedule-based multi-dimensional retiming* [21].

Examining table 1, the chained MD retiming is the method that can get closer results to the multi-dimensional interleaving and retiming, but still requires a larger number of queues that depend on the problem size. The wavefront methods seem to require less memory than the chained retiming, however, the number of queues is still the same and also larger than the result obtained from the multi-dimensional interleaving and retiming. The wavefront methods also present zero delays in most of the edges, which shows that those methods are not able



<i>Edge e</i>	$d(e)$	$d_e(e)$	$d_m(e)$	$d_r(e)$	<i>orig</i>	<i>MdIntRet</i>	<i>chained</i>	<i>hyp</i>	<i>aff</i>
$M1 \rightarrow A$	(0,1)	(0,1)	(6,0)	(7,0)	M	7	M	M	M
			(-24,1)	(-23,1)		M-23			
$A \rightarrow B$	(0,1)	(0,1)	(6,0)	(6,0)	M	6	M	M	M
			(-24,1)	(-24,1)		M-24			
$M4 \rightarrow C$	(0,1)	(0,1)	(6,0)	(7,0)	M	7	M	M	M
			(-24,1)	(-23,1)		M-23			
$C \rightarrow D$	(0,1)	(0,1)	(6,0)	(6,0)	M	6	M	M	M
			(-24,1)	(-24,1)		M-24			
$M7 \rightarrow F$	(0,1)	(0,1)	(6,0)	(7,0)	M	7	M	M	M
			(-24,1)	(-23,1)		M-23			
$F \rightarrow G$	(0,1)	(0,1)	(6,0)	(6,0)	M	6	M	M	M
			(-24,1)	(-24,1)		M-24			
$B \rightarrow E$	(1,0)	(5,0)	(5,0)	(6,0)	0	6	M	M	M
$E \rightarrow H$	(1,0)	(5,0)	(5,0)	(5,0)	0	5	M	M	M
$M11 \rightarrow L$	(0,1)	(0,1)	(6,0)	(7,0)	M	7	M	M	M
			(-24,1)	(-23,1)		M-23			
$M12 \rightarrow N$	(0,1)	(0,1)	(6,0)	(7,0)	M	7	M	M	M
			(-24,1)	(-23,1)		M-23			
$L \rightarrow K$	(0,1)	(0,1)	(6,0)	(4,0)	M	4	M	M	M
			(-24,1)	(-26,1)		M-26			
$N \rightarrow P$	(0,1)	(0,1)	(6,0)	(6,0)	M	6	M	M	M
			(-24,1)	(-24,1)		M-24			
$M15 \rightarrow Q$	(0,1)	(0,1)	(6,0)	(7,0)	M	7	M	M	M
			(-24,1)	(-23,1)		M-23			
$M16 \rightarrow Q$	(0,1)	(0,1)	(6,0)	(7,0)	M	7	M	M	M
			(-24,1)	(-23,1)		M-23			
$Q \rightarrow R$	(0,1)	(0,1)	(6,0)	(6,0)	M	6	M	M	M
			(-24,1)	(-24,1)		M-24			
$P \rightarrow S$	(1,0)	(5,0)	(5,0)	(5,0)	0	5	M	M	M
$S \rightarrow J$	(1,0)	(5,0)	(5,0)	(1,0)	0	1	M	M	M
all others	(0,0)	(0,0)	(0,0)	(1,0)	0	1	1	0	O(M)

Table 1: Intermediate results of *MdIntRet* for the 2D Filter and comparison to other methods

Method	execution time of one iteration	number of queues	complexity of the algorithm
original design	200 ns	12	not applicable
manual redesign	60 ns	17	not applicable
affine-by-st.	unknown	40	ILP
sch.-based MD ret.	40 ns(optimal)	40	ILP
wavefront	200 ns	16	ILP
chained MD ret.	40 ns(optimal)	16	ILP
MdIntRet	40 ns(optimal)	12	$O( E )$

Table 2: Summary of results for the 2D filter

to achieve full parallelism. Meanwhile, the affine-by-statement methods, if modified to design single processor systems, are expected to produce a fully parallel solution as the schedule-based retiming does, however, due to the change in the schedule, these methods end up with larger memory requirements when compared to our proposed solution. Table 2 summarizes the comparison between our results and other methods. In this table, we compare the achieved execution time<sup>3</sup> for each of the mentioned methods, as well the complexity of the algorithm involved<sup>4</sup> <sup>5</sup>. We notice that when the fully parallel solution was achieved, the number of queues required by the final design and the complexity of the algorithms become the distinguishing elements on this comparison.

These examples show the efficiency of the application of this new method and the significant performance improvement obtained in the single processor design.

## 7 Conclusion

We have presented a novel technique for optimizing a circuit designed to compute multi-dimensional applications. The circuit was initially represented by a multi-dimensional data flow graph and it was transformed through the use of a multi-dimensional interleaving method that we developed. This technique combined with a multi-dimensional retiming is able to achieve full parallelism in a single processor design, i.e., simultaneous execution of all operations (nodes) of the multi-dimensional data flow graph, maintaining the original schedule vector. The transformation applied to the circuit does not require additional queues dependent on the problem size as usually occurs with other methods. The complexity of this new algorithm is  $O(|E|)$  time, where  $E$  is the set of edges of the multi-dimensional data flow graph represent-

<sup>3</sup>The execution time for the affine-by-statement methods was not determined since most of the original algorithms are applicable to systolic arrays and do not present a solution for single processors

<sup>4</sup>For the redesign proposed in [10], we assumed row-wise execution, and the complexity could not be determined since no algorithm was proposed to find such a solution.

<sup>5</sup>ILP was used to represent methods with complexity equivalent to an Integer Linear Programming solution, or eventually to an LP algorithm

ing the problem. The description of the multi-dimensional interleaving technique and how to compute the changes in the memory structure were presented. Examples of image processing problems were used to demonstrate the significant improvements in execution time and memory requirements obtained by our method when compared to results from other known techniques.

## References

- [1] A. Aiken and A. Nicolau, "Fine-Grain Parallelization and the Wavefront Method," in *Languages and Compilers for Parallel Computing*, Cambridge, Massachusetts, MIT Press, 1990, pp. 1-16.
- [2] U. Banerjee, "Unimodular Transformations of Double Loops," in *Advances in Languages and Compilers for Parallel Processing*, Cambridge, Massachusetts, MIT Press, 1991, pp. 192-219.
- [3] T. P. Barnwell and C. J. M. Hodges, "Optimal Implementation of Signal Flow Graphs on Synchronous Multiprocessor". *Proc. International Conference on Parallel Processing*, 1982, pp. 90-93.
- [4] M. A. Bayoumi, N. A. Ramakrishna, V. Israni and R. K. Jayam, "SDS: A Framework for the Design of DSP ASICs". in *Proc. 1992 IEEE International Conference on Acoustic, Speech, and Signal Processing*, 1992, pp. 545-548.
- [5] L.-F. Chao and E. H.-M. Sha, "Retiming and Unfolding Data-Flow Graphs", in *Proc. 1992 International Conference on Parallel Processing*, August 1992, pp. II 33-40.
- [6] L.-F. Chao and E. H.-M. Sha, "Static Scheduling of Uniform Nested Loops," in *Proceedings of 7th International Parallel Processing Symposium*, April, 1993, pp. 1421-1424.
- [7] L.-F. Chao, "Scheduling and Behavioral Transformations for Parallel Systems," Ph.D. dissertation, Princeton University, 1993.
- [8] C.-M. Chu, M. Potkonjak, M. Thaler and J. Rabaey, "Hyper: An Interactive Synthesis Environment for High Performance Real-Time Applications," in *Proc. International Conference on Computer Design*, pp. 432-435, 1989.
- [9] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1984.
- [10] Gnanasekaran, R., "2-D Filter Implementation for Real-Time Signal Processing," in *IEEE Transactions on Circuits and Systems*, May 1988, vol. 35, n. 5, pp. 587-590.
- [11] G. Goosens, J. Wandewalle, and H. de Man, "Loop Optimization in Register Transfer Scheduling for DSP Systems," in *Proc. ACM/IEEE Design Automation Conference*, 1989, pp. 826-831.

- [12] A. K. Jain, "Image Data Compression: a Review," in *Proceedings of the IEEE*, vol. 69, no. 3, March 1981, pp. 349-389.
- [13] R. Jain, P. T. Yang and T. Yoshino "FIRGEN: A Computer-Aided Design System for High Performance FIR Filter Integrated Circuits," in *IEEE Transactions on Signal Processing*, vol. 39, no. 7, July 1991, pp. 1655-1668.
- [14] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice Hall, 1988.
- [15] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," in *Algorithmica*, 6, 1991, pp. 5-35.
- [16] L.-S. Liu, C.-W. Ho and J.-P. Sheu, "On the Parallelism of Nested For-Loops Using Index Shift Method," in *Proceedings of the 1990 International Conference on Parallel Processing*, 1990, Vol. II, pp. 119-123.
- [17] *Design Architect User's Manual*, Software Version 8.2-5, unpublished, Mentor Graphics Corporation, 1993.
- [18] A. Nicolau, "Loop Quantization or Unwinding Done Right," in *Proc. of the 1987 ACM International Conference on Supercomputing*, Springer Verlag Lecture Notes on Computer Science 289, May 1987, pp. 294-308.
- [19] B. M. Pangrle and D. Gajski, "Slicer: A State Synthesizer for Intelligent Silicon Compilation," in *Proc. International Conference on Computer Design*, 1987, pp. 42-45.
- [20] K. K. Parhi and D. G. Messerschmitt, "Fully-Static Rate-Optimal Scheduling of Iterative Data-Flow Programs Via Optimum Unfolding," in *Proc. International Conference on Parallel Processing*, I, 1989, pp. 209-216.
- [21] N. L. Passos, E. H.-M. Sha, and S. C. Bass, "Schedule-Based Multi-Dimensional Retiming," in *Proceedings of 8th International Parallel Processing Symposium*, April, 1994, pp 195-199.
- [22] N. L. Passos, E. H.-M. Sha, and S. C. Bass, "Loop Pipelining for Scheduling Multi-Dimensional Systems via Rotation," in *Proceedings of 31st Design Automation Conference*, June, 1994, pp. 485-490.
- [23] N. L. Passos and E. H.-M. Sha "Full Parallelism in Uniform Nested Loops using Multi-Dimensional Retiming," in *Proceedings of 23rd International Conference on Parallel Processing*, August, 1994, vol. II, pp. 130-133.
- [24] P. Quinton and Y. Robert, *Systolic Algorithms and Architectures*, Hertfordshire, UK: Prentice Hall International, 1991.
- [25] A. Darté and Y. Robert, "Constructive Methods for Scheduling Uniform Loop Nests," in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, no. 8, 1994, pp. 814-822.
- [26] R. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, Pennsylvania, 1983.

- [27] C.-Y. Wang and K. K. Parhi, "High Level DSP Synthesis Using the MARS Design System," in *Proceedings of the International Symposium on Circuits and Systems*, 1992, pp. 164-167.
- [28] A. van der Werf, "Flexible Data Path Compilation for PHIDEO," in *Proceedings of Euro ASIC '91*, May, 1991, pp. 178-183.
- [29] M. Wolfe, "Loop Skewing: the Wavefront Method Revisited," in *International Journal of Parallel Programming*, Vol. 15, No. 4, pp. 284-294, August 1986.
- [30] M. Wolf and M. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, n. 4, 1991, pp. 452-471.
- [31] C.-W. Wu, "Bit-Level Pipelined 2-D Digital Filters for Real-Time Image Processing," in *IEEE Transactions on Circuits and Systems for Video Technology*, March, 1991, vol. 1, no. 1, pp. 22-34.