

# Achieving Full Parallelism using Multi-Dimensional Retiming

*Nelson Luiz Passos*

*Edwin Hsing-Mean Sha*

Dept. of Computer Science & Engin.

University of Notre Dame

Notre Dame, IN 46556

## ABSTRACT

Most scientific and Digital Signal Processing (DSP) applications are recursive or iterative. Transformation techniques are usually applied to get optimal execution rates in parallel and/or pipeline systems. The retiming technique is a common and valuable transformation tool in one-dimensional problems, when loops are represented by data flow graphs (DFGs). In this paper, uniform nested loops are modeled as multi-dimensional data flow graphs (MDFGs). Full parallelism of the loop body, i.e., all nodes in the MDFG executed in parallel, substantially decreases the overall computation time. It is well known that, for one-dimensional DFGs, retiming can not always achieve full parallelism. Other existing optimization techniques for nested loops also can not always achieve full parallelism. This paper shows an important and counter-intuitive result, which proves that we can always obtain full-parallelism for MDFGs with more than one dimension. This result is obtained by transforming the MDFG into a new structure. The restructuring process is based on a multi-dimensional retiming technique. The theory and two algorithms to obtain full parallelism are presented in this paper. Examples of optimization of nested loops, and digital signal processing designs are shown to demonstrate the effectiveness of the algorithms.

# 1 Introduction

Applications such as image processing, fluid mechanics, and weather forecasting, require high computer performance. Researchers and designers in those areas are looking for solutions to multi-dimensional problems through the use of parallel computers and/or specialized hardware, which justifies the study of multi-dimensional optimization algorithms. Efficient transformation mechanisms, for parallel and/or pipelined processing of one-dimensional problems represented by Data Flow Graphs, have been proposed in previous studies using retiming techniques [20]. The retiming technique regroups operations in iterations in order to produce a new iteration structure with higher parallelism embedded [4, 29]. An equivalent approach to the multi-dimensional case will benefit parallel compiler design [1, 13, 18], as well as high-level synthesis for VLSI design [6, 26, 23].

Computation-intensive applications usually depend on time-critical sections consisting of a loop of instructions. To optimize the execution rate of such applications, the designer needs to explore the embedded parallelism in repetitive patterns of a loop. Some research has been done on uniform nested loop scheduling, a similar view to a multi-dimensional (MD) problem. For example, loop quantization [22], loop skewing [32] and unimodular transformations [3, 34]. These techniques differ from our method since they do not change the structure of iterations, but the sequence in which the iterations are executed. Loop quantization requires the unrolling of multiple nested loops to achieve the parallel solution, increasing the problem size. Loop skewing and unimodular transformations change the loop indices in such a way to obtain parallelism among iterations. However, the sequential dependence that may exist inside the loop body is not changed. In our method, all instructions contained in the loop body can be parallelized. The algorithms are independent of the expected granularity of parallelism [25, 27], allowing the designer to specify the desired solution: fine or coarse-grain parallelism.

Other techniques that search beyond loop boundaries include perfect pipelining [1] and Doacross loops [10]. Perfect pipelining looks for a repeating pattern for a single loop with the disadvantage of unpredictability of the size of such pattern and the gain of performance. Doacross loop presents an overlap of consecutive iterations that contributes to the improvement of the execution rate. These two methods can be regarded as special cases of the one-dimensional retiming technique [7, 8].

In our study, loop bodies are represented by multi-dimensional data flow graphs (MD-FGs). Instead of simply overlapping iterations, we restructure the loop body, i.e., the existing dependence delays, preserving the original global data dependence. We model the process of restructuring as a multi-dimensional retiming. The new MDFG is constructed from the original graph through the retiming function. The retiming process is,

in essence, the same concept as loop pipelining [5]. Previous research in loop pipelining, for loops with cyclic dependencies, produced methods which focus on one-dimensional problems. Such methods appear in several systems [14, 19, 28, 31]. Chao and Sha have introduced an unified algorithm for different models, able to achieve the best possible iteration period for cyclic data flow graphs representing one-dimensional problems [8]. However, in the multi-dimensional case, the iteration period has no lower bounds [24, 26].

For the case of the one-dimensional retiming technique, negative delays represent the existence of a cycle in the execution sequence and can not be allowed. However, on the multi-dimensional case, the existence of negative delays is to be considered natural if there exists a schedule vector  $s$  that turns the graph realizable. In this paper, we present a more general solution, applicable to coded loops as well as to other multi-dimensional problems such as those found in digital signal processing (for example, multi-dimensional filters). The schedule vector  $s$  can be a *recursion schedule vector*, i.e.,  $s$  is parallel to one of the axis in the index space [16], or a non-recursive schedule vector which implies that the execution sequence is equivalent to a wavefront processing [2, 16, 17]. Nested loops coded in high level programming language have the implicit characteristic of executing under a recursion schedule vector. Digital signal processing and high level synthesis tools are examples of applications that may require a non-recursive schedule.

The concept of multi-dimensional retiming was introduced by Chao and Sha in [4]. Their algorithm does not guarantee to achieve full parallelism and is only applicable to a specific class of MDFGs, where any cycle would have a strictly non-negative total multi-dimensional delay. In this paper, two new algorithms applicable to a general form of MDFGs are presented. We call these algorithms the *incremental multi-dimensional retiming* and the *chained multi-dimensional retiming*. The first uses a legal MD retiming to successively restructure the loop body represented by an MDFG. The second improves the efficiency of the first, by obtaining the final solution in only one pass. Both techniques achieve full parallelism. In this paper, full parallelism is regarded as the possibility of simultaneously executing all operations (nodes) in an MDFG. Therefore, to obtain full parallelism is equivalent to obtain non-zero delays on all edges of the MDFG, such that the resulting MDFG can be computed by a legal schedule. In the one-dimensional case, such a condition can not always be achieved through retiming due to the constancy of the sum of delays in a cycle, and the requirement of non-negative delays.

In this paper, we focus on examples of obtaining a fine-grain parallelism. Coarser grain is easily obtained by using the same technique in graphs where each node represents group of operations [27]. For simplicity, the examples consist of two-dimensional problems without instructions between loops. The two dimensions are generically referred as  $x$  and  $y$ . The multi-dimensional case is a straightforward extension of the concepts presented. The first example consists of an IIR filter (Infinite-Extent Impulse Response)

[11], represented by the transfer function:

$$H(z_1, z_2) = \frac{1}{\left(1 - \sum_{n_1=0}^2 \sum_{n_2=0}^2 c(n_1, n_2) * z_1^{-n_1} * z_2^{-n_2}\right)} \text{ for } n_1 \neq 0 \text{ or } n_2 \neq 0$$

which can be translated into the equation

$$y(n_1, n_2) = x(n_1, n_2) + \sum_{k_1=0}^2 \sum_{k_2=0}^2 c(k_1, k_2) * y(n_1 - k_1, n_2 - k_2) \text{ for } k_1 \neq 0 \text{ or } k_2 \neq 0 \quad (1)$$

This example can be seen as a nested loop in a high-level programming language code, as well as a typical DSP problem, demonstrating the possibilities of application of our method. An MDFG<sup>1</sup> derived from the equation above is shown in Figure 1(a). An equivalent code is presented in Figure 1(b). For better understanding, the table below shows the relationship between the graph nodes and the operations on equation 1.

M1	$c(0,1)*y(n1,n2-1)$
M2	$c(0,2)*y(n1,n2-2)$
M3	$c(1,0)*y(n1-1,n2)$
M4	$c(1,1)*y(n1-1,n2-1)$
M5	$c(1,2)*y(n1-1,n2-2)$
M6	$c(2,0)*y(n1-2,n2)$
M7	$c(2,1)*y(n1-2,n2-1)$
M8	$c(2,2)*y(n1-2,n2-2)$
A1	$c(0,1)*y(n1,n2-1) + c(0,2)*y(n1,n2-2)$
A2	$c(1,0)*y(n1-1,n2) + c(1,1)*y(n1-1,n2-1)$
A3	$c(1,2)*y(n1-1,n2-2) + c(2,0)*y(n1-2,n2)$
A4	$c(2,1)*y(n1-2,n2-1) + c(2,2)*y(n1-2,n2-2)$
A5	$A1 + x(n1,n2)$
A6	$A3 + A4$
A7	$A2 + A6$
A8	$A5 + A7$

Notice that dependence vectors are represented by pairs  $(d.x, d.y)$  where  $d.x$  corresponds to the dependence distance in the Cartesian axis representing the outermost loop, while  $d.y$  corresponds to the innermost loop. This initial representation is the same as the lexicographic order proposed by Lamport in [17] and later used by Lam and Wolf in [34] and Banerjee in [3]. The proposed method, however, does not depend on such ordering and can be applied to any consistent vector representation as shown in the theoretical sections.

---

<sup>1</sup>An equation (code) can be represented by different data flow graphs, depending on the target processor, compiler order of evaluation, optimization assumptions, etc.. After considering such characteristics the resulting MDFG should be unique. This paper assumes that the MDFG is given, such that the method can be applied to any problem.

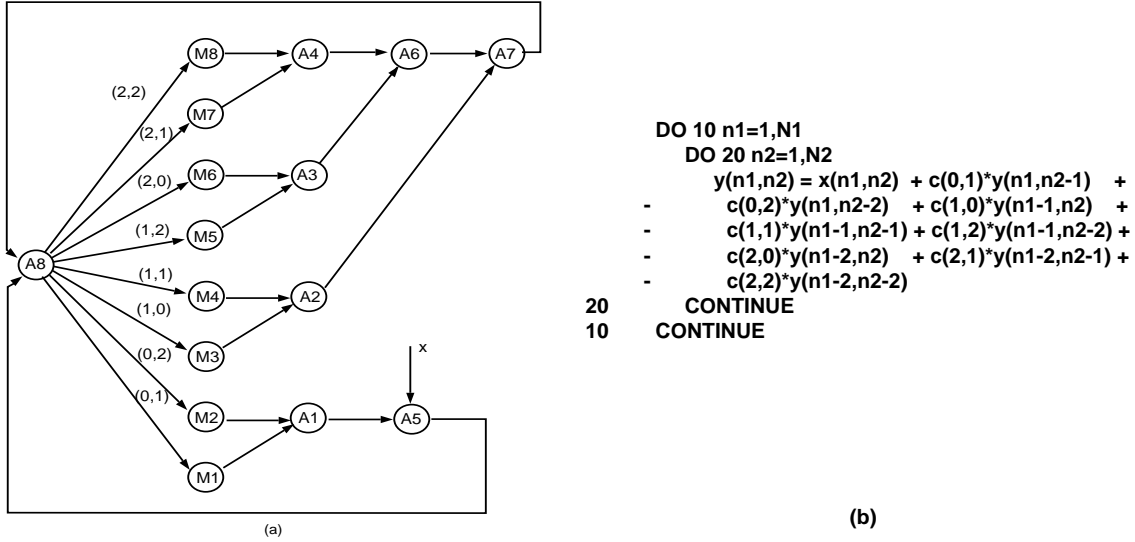


Figure 1: (a) MDFG representing an IIR filter (b) equivalent Fortran code

A bad selection of the multi-dimensional retiming value can introduce cycles in the execution sequence, resulting in an infeasible solution. To avoid this problem, our algorithm was used to compute the legal retiming values applied to this example. A retimed graph is presented in Figure 2(a), where the two-dimensional delay  $(1, -1)$  is pushed through all nodes labeled  $M_i$  for  $1 \leq i \leq 8$  on the original MDFG. Intuitively, the retimed nodes no longer precede the additions  $A_1$  to  $A_4$  within the same iteration, which allows those operations to be executed simultaneously. This new characteristic reduces the critical path of the graph, and consequently the overall execution time. Notice that the dependence vectors are no more lexicographically positive as required in other methods [34]. This problem will disappear after the code has been modified. Figure 2(b) shows the result from a new retiming operation, where the two-dimensional delay  $(2, -3)$  was pushed through nodes  $A_1$  to  $A_4$ , improving the parallelism and reducing the critical path furthermore. In the next step, we retime nodes  $A_5$  and  $A_6$  by  $r = (4, -7)$  and we obtain the graph shown in Figure 3(a). The final graph is presented in Figure 3(b) after retiming the node  $A_7$  by  $r = (8, -15)$ . Full parallelism was achieved when all edges became non-zero delay edges. As mentioned earlier, the new dependence vectors are not acceptable as legal under the lexicographic order concept, however, a new schedule vector,  $s = (15, 8)$ , makes the loop realizable, and adjusting the code to this schedule will make the dependence vectors lexicographically positive.

The remaining of this paper is organized as follows: Section 2 shows some basic concepts, such as mathematical models for Dependence Graphs and Data Flow Graphs. Also, we mention the basic idea of multi-dimensional retiming. Section 3 shows the properties

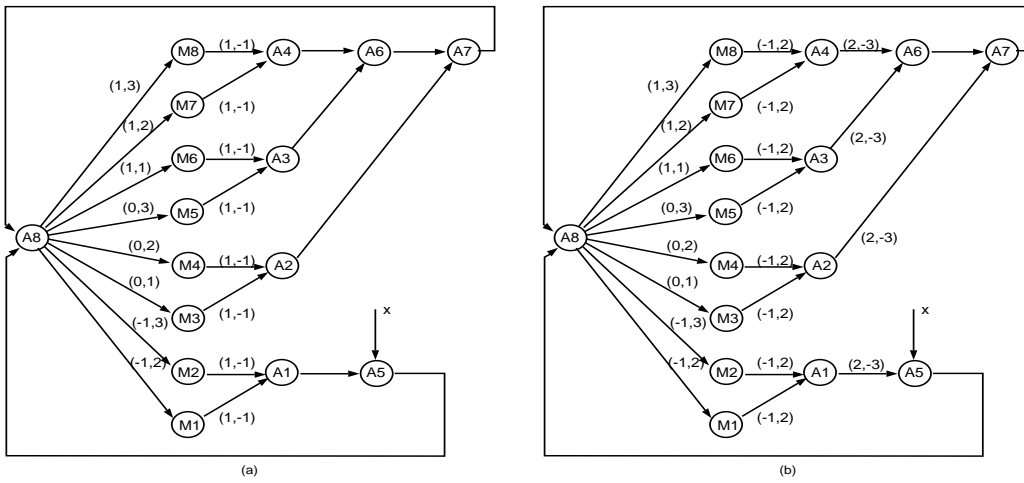


Figure 2: (a) MDFG after retiming all nodes M (b) MDFG after retiming A1 to A4

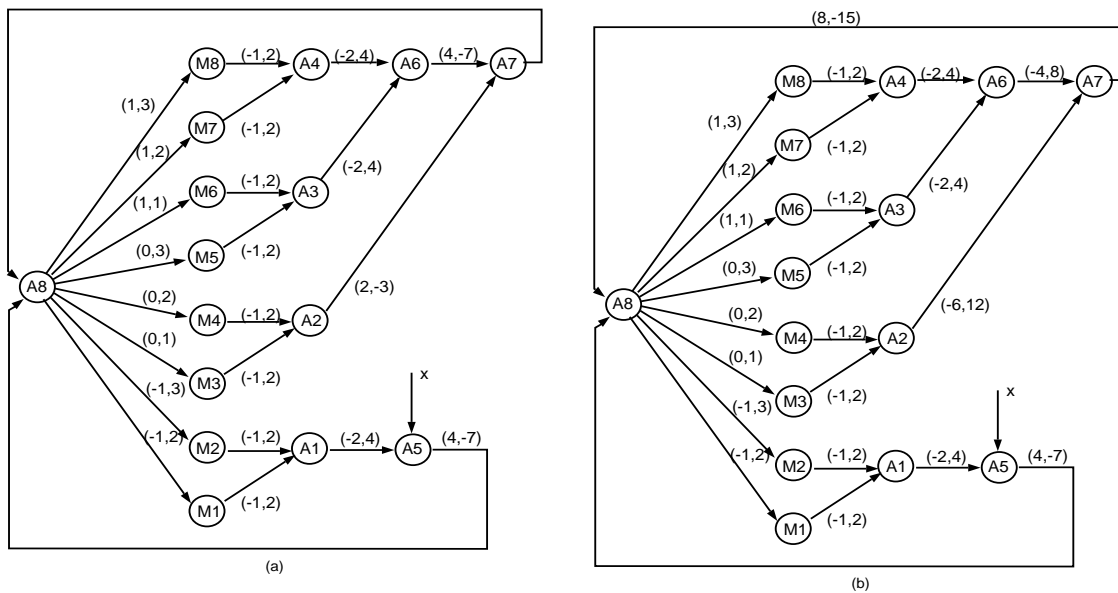


Figure 3: (a) MDFG after retiming A5 and A6 (b) fully parallel MDFG after retiming A7

necessary to have a legal MD retiming function. The *incremental multi-dimensional retiming* algorithm is described, giving us the foundations required for the theorem that proves that full parallelism can always be achieved. Section 4 presents a more efficient algorithm *chained multi-dimensional retiming* which uses the concept of a chain reaction. Section 5 synthesizes the code transformation, while Section 6 shows examples of application of our techniques, followed by a summary of the concepts introduced in this paper.

## 2 Basic Principles

### 2.1 Background

In this section we define some terms related to the interpretation of multi-dimensional data flow graphs, such as mathematical models, dependence vectors, iterations, and vector operations. We begin by presenting the multi-dimensional data flow graph definition that is used to represent the multiple nested loops or the multi-dimensional problem specifications.

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,  $E \subseteq V \times V$  is the set of dependence edges,  $d$  is a function from  $E$  to  $Z^n$ , representing the multi-dimensional delay between two nodes, where  $n$  is the number of dimensions, and  $t$  is a function from  $V$  to the positive integers, representing the computation time of each node. A two-dimensional data flow graph (2DFG)  $G = (V, E, d, t)$  is an MDFG, where  $d$  is a function from  $E$  to  $Z^2$ . We use  $d(e) = (d.x, d.y)$  as a general formulation of any delay shown in a two-dimensional DFG. An example of a two-dimensional DFG and its equivalent code is shown in Figure 4.

The execution of each node in  $V$  exactly once represents an *iteration*, i.e., the execution of one instance of the loop body. Iterations are identified by a vector  $i$ , equivalent to a multi-dimensional index, starting from  $(0, 0, \dots, 0)$ . Inter-iteration dependencies are represented by vector-weighted edges. For any iteration  $\hat{j}$ , an edge  $e$  from  $u$  to  $v$  with delay vector  $d(e)$  means that the computation of node  $v$  at iteration  $\hat{j}$  depends on the execution of node  $u$  at iteration  $\hat{j} - d(e)$ . An edge with delay  $(0, 0, \dots, 0)$  represents a data dependence within the same iteration. A legal MDFG must have no zero-delay cycle, i.e., the summation of the delay vectors along any cycle can not be  $(0, 0, \dots, 0)$ . Several techniques are available to verify that an MDFG does not have a cycle [9, 15]. We call *uniform loops* those loops that present the characteristic of constant dependence

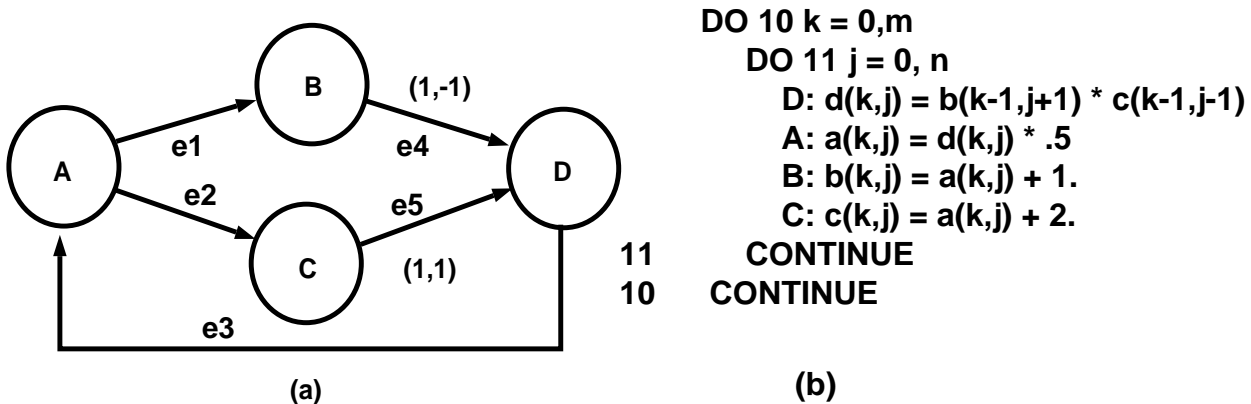


Figure 4: (a) MDFG extracted from a Wave Digital Filter (b) equivalent Fortran code

vectors, i.e., their data dependencies are at a constant distance in the iteration space.

Considering the example shown in Figure 4(a), with dependence vectors  $d(e) = (d.x, d.y)$ , such that  $d.x$  represents the distance in the x-direction of the original problem or the outermost loop index in the example, and  $d.y$  the dependencies in the y-direction or the innermost loop, the MDFG  $G = (V, E, d, t)$  would consist of  $V = \{A, B, C, D\}$  and  $E = \{e1 : (A, B), e2 : (A, C), e3 : (D, A), e4 : (B, D), e5 : (C, D)\}$  where,  $d(e1) = d(e2) = d(e3) = (0, 0)$ ,  $d(e4) = (1, -1)$ ,  $d(e5) = (1, 1)$ . For simplicity, each operation is assumed to be executed in one time unit, therefore,  $t(A) = t(B) = t(C) = t(D) = 1$ . 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  $\sum_{i=0}^k t(v_i)$ .

The period during which all computation nodes in an iteration are executed, according to existing data dependencies and without resource constraints, is called a *cycle period*. The *cycle period*  $C(G)$  of an MDFG  $G$  is the maximum computational time among paths that have no delay. For example, the MDFG in Figure 4(a) has  $C(G) = 3$ , which can be measured through the paths  $p = D \rightarrow A \rightarrow B$  or  $p = D \rightarrow A \rightarrow C$ .

The directed acyclic graph, showing the dependencies between copies of nodes representing an MDFG  $G$ , is called a *cell dependence graph* (DG) of the MDFG  $G$ . Figure 5(a) shows the replication of the MDFG in Figure 4(a), and Figure 5(b) shows its cell dependence graph. The cell dependence graph is bounded by the dimensions of the problem which it represents. In the case of multiple nested loops, such bounds are the loop indices

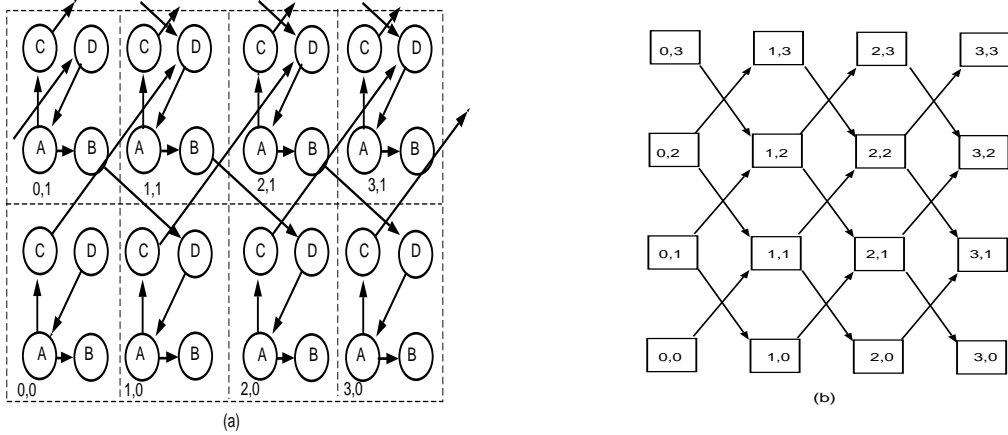


Figure 5: (a) DG based on the replication of an MDFG, showing iterations starting at  $(0,0)$ . (b) DG represented by computational cells

bounds and the cell dependence graph has the same index domain as the *iteration space* described in [34]. For other multi-dimensional cases, the boundaries are imposed by the problem description such as the dimensions of a window in an image processing application. A *computational cell* is the DG node that represents a copy of the MDFG, excluded the edges with delay vectors different from  $(0, 0, \dots, 0)$ , i.e., a complete iteration. The computational cell is considered an atomic execution unit.

A mathematical model for a cell dependence graph is presented in [21]: a tuple  $(I^n, D)$  where  $I^n$  is the index set of the cell structure, equivalent to the vectors used to identify each iteration, and  $D$  is a matrix containing all dependence vectors, also referred as delay vectors. Consider, for example, the DG shown in Figure 5. Since only a region of the iteration space is presented, let us assume that the upper bounds in any direction are the value 30. Then, the DG model is described as:

$$I^2 = \{(i, j) : 0 \leq i \leq 30, 0 \leq j \leq 30\}, \text{ and } D = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

We say that a legal MDFG  $G = (V, E, d, t)$  is *realizable* if there exists a schedule vector  $s$  for the cell dependence graph with respect to  $G$ , i.e.,  $s \cdot d \geq 0$  for any  $d \in G$  [16], and no cycle exists in its equivalent DG. A *schedule vector*  $s$  is the normal vector for a set of parallel equitemporal hyperplanes that define a sequence of execution of the cell dependence graph.

To manipulate MDFG characteristics represented on vector notation, such as the delay vectors, we make use of component-wise vector operations. For two two-dimensional

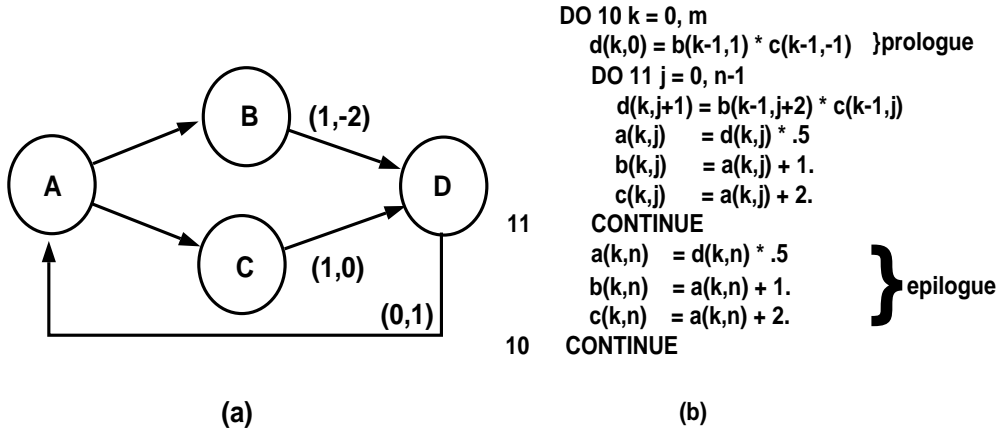


Figure 6: (a) MDFG after retimed by  $r(D)=(0,1)$  (b) equivalent code

vectors  $P$  and  $Q$ , represented by their coordinates  $(P.x, P.y)$  and  $(Q.x, Q.y)$ , an example of arithmetic operation is  $P + Q = (P.x + Q.x, P.y + Q.y)$ . The notation  $P \cdot Q$  indicates the inner product between  $P$  and  $Q$ , i.e.,  $P \cdot Q = P.x * Q.x + P.y * Q.y$ .

## 2.2 Retiming a Multi-Dimensional Data Flow Graph

This paper shows how to obtain full parallelism of a multiple uniform nested loop, by using multi-dimensional retiming. In this subsection, we show the basic ideas on multi-dimensional retiming, such as cycle period optimization, and characteristics of a legal MD retiming.

A *multi-dimensional retiming*  $r$  is a function from  $V$  to  $Z^n$  that redistributes the nodes in the cell dependence graph created by the replication of an MDFG  $G$ . A new MDFG  $G_r$  is produced, 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 dependencies, 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  $i - r(u)$ . A *two-dimensional retiming*  $r$  is a function from  $V$  to  $Z^2$  applicable to a 2DFG  $G$ . Figure 6 shows the MDFG from Figure 4(a) retimed by the function  $r(D) = (0, 1)$  and the modified code. The critical paths of this graph are the edges  $A \rightarrow B$  and  $A \rightarrow C$  with an execution time of 2 time units. The retimed DG is shown in Figure 7, where

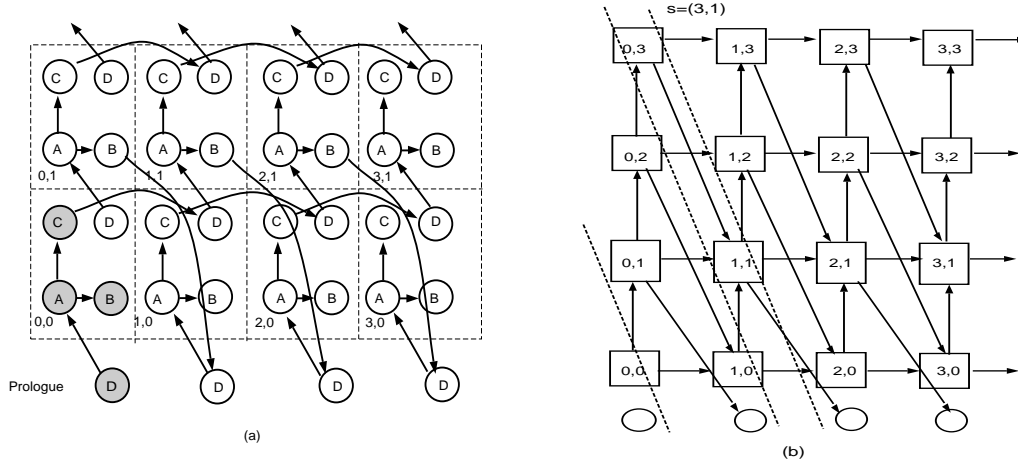


Figure 7: (a) DG based on the replication of an MDFG, after retiming. (b) same DG represented by computational cells.

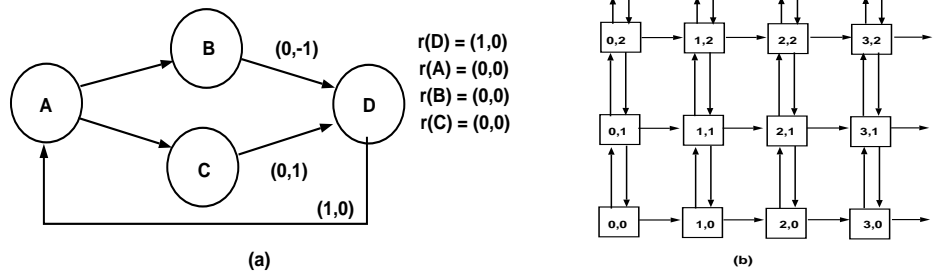


Figure 8: (a) Example of illegal retiming. (b) DG showing cycles in the y-direction.

the nodes originally belonging to iteration  $(0, 0)$  are marked. There are infinite schedule vectors that can realize this graph. A possible schedule vector for the retimed graph is  $s = (3, 1)$ . The reduction of the critical path, as seen in Figure 6, results in a saving of approximately one third of the total execution time. Figure 8 shows an illegal retiming function applied to the same example. By simple inspection of the cell dependence graph in Figure 8(b) we notice the existence of a cycle created by the dependencies  $(0, 1)$  and  $(0, -1)$ .

The set of instructions that are moved on directions  $x$  and  $y$ , in a two-dimensional retiming, is called *prologue*. The prologue must be executed to provide the necessary data for the iterative process. In our example shown in Figure 7, the instruction  $D$  becomes the prologue for that problem. An *epilogue* is the other extreme of the DG, were a complementary set of instructions will be executed to complete the process. It is trivial to notice that the prologue and the epilogue have the same length. After the

multi-dimensional retiming technique has been applied, the iteration space of the loop can be represented by a subset of the new dependence graph, with its boundaries reduced by the length of the prologue in each dimension.

## 3 Incremental Multi-Dimensional Retiming

### 3.1 Basic Concepts

The technique of multi-dimensional retiming allows changes in more than one direction [4, 24]. Our method uses this property to achieve the full parallelism, i.e., simultaneous execution of all nodes in an MDFG, by incrementally applying the MD retiming function to uniform nested loops or multi-dimensional problems. Properties, algorithm and supporting theorems for the incremental multi-dimensional retiming are presented below. We begin by defining a legal MD retiming function.

**Definition 3.1** Given a realizable MDFG  $G$ , a *legal multi-dimensional retiming* for  $G$  is the multi-dimensional retiming function  $r$  that transforms  $G$  in  $G_r$  such that  $G_r$  is still realizable.

To find a legal retiming of an MDFG, such that all its nodes can be executed in parallel is equivalent to obtaining a cycle period equal to one when we assume the execution time of each node to be one time unit. This problem is more complex than the use of traditional retiming on one-dimensional cases. In those cases, the positive delays after retiming guarantee the legal retiming. However, for the multi-dimensional problem, positive delay vectors are too restrictive since an MDFG may be still realizable even if it has negative delays. Therefore, the following theorem gives a more general set of constraints for a legal MD retiming that produces full parallelism among the nodes of an MDFG.

**Theorem 3.1** Let  $G = (V, E, d, t)$  be an MDFG with  $t(v) = 1$  for any  $v \in V$ .  $r$  is a legal multi-dimensional retiming for  $G$  such that  $C(G_r) = 1$  if and only if:

1. the cell dependence graph of the retimed MDFG  $G_r = (V, E, d_r, t)$  does not contain any cycle.
2.  $d_r(e) \neq (0, 0, \dots, 0)$  for any  $e \in E$
3. if the execution time of a path  $p$  in  $G_r$  is greater than one, then  $d_r(p) \neq (0, 0, \dots, 0)$ .

*Proof:* Let  $C(G_r) = 1$  after  $G$  has been retimed by  $r$ . Then, the first condition results from the realizability of the final cell dependence graph. If a cycle was allowed, we would find cells depending on their own results, waiting to be executed, which is a contradiction. The second and third conditions result from the properties associated with the cycle period.

Now, let us assume that  $G_r$  satisfies conditions 1 to 3. From condition 1,  $G_r$  is realizable, which implies that  $r$  is a legal multi-dimensional retiming. From conditions 2 and 3, we know that the critical path of  $G_r$  must contain only one node, therefore,  $C(G_r) = 1$ .  $\square$

Our technique enforces these constraints through the use of an incremental retiming that guarantees the realization of each new retimed graph until all edges become non-zero delay edges.

## 3.2 The Scheduling Subspace

We know that the existence of a linear schedule vector for the execution of the MDFG and the non-existence of a cycle are the necessary and sufficient conditions for its realizability [16]. We define the space domain for the schedule vectors as follows:

**Definition 3.2** A *scheduling subspace*  $S$  for a realizable MDFG  $G = (V, E, d, t)$  is the space region where there exist schedule vectors that realize  $G$ , i.e., if schedule  $s \in S$  then  $d(e) \cdot s \geq 0$  for any  $e \in E$ .

An example of scheduling subspace is shown in Figure 9 for a two-dimensional case. In this example, each of the dependencies  $d, d', d''$  define half-planes that contain vectors whose dot product with the respective dependence vector is greater or equal to zero. The intersection of these half-planes produces the final scheduling subspace.

Using such concepts, we redefine the basic conditions for a legal multi-dimensional retiming through the following lemma.

**Lemma 3.2** Let  $G = (V, E, d, t)$  be a realizable MDFG,  $r$  a multi-dimensional retiming, and  $s$  a schedule vector for the retimed graph  $G_r = (V, E, d_r, t)$ , then

- (a) for any path  $u \overset{p}{\rightsquigarrow} v$ , we have  $d_r(p) = d(p) + r(u) - r(v)$
- (b) for any cycle  $l \in G$ , we have  $d_r(l) = d(l)$
- (c) for any edge  $u \xrightarrow{e} v$ ,  $d_r(e) \cdot s \geq 0$
- (d) there is no cycle in the DG equivalent to the MDFG  $G$ .

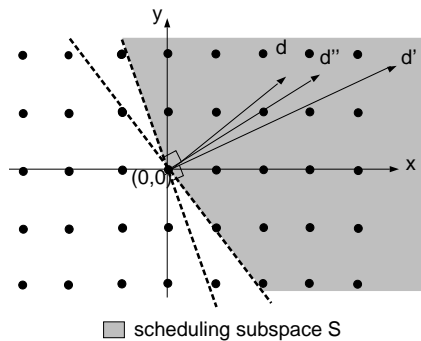


Figure 9: An example of scheduling subspace

### 3.3 The Selection of the Retiming Function

The selection of a legal multi-dimensional retiming function becomes an important step on our method to avoid “generate and test” procedures. In this segment, we show how to select a legal retiming function for a realizable MDFG. Such selection is based on properties of the scheduling subspace associated with the MDFG. We begin by defining a strictly positive scheduling subspace.

**Definition 3.3** Given a realizable MDFG  $G = (V, E, d, t)$  with a scheduling subspace  $S$ , a *strictly positive scheduling subspace*  $S^+$  is the set of all vectors  $s \in S$  such that  $d(e) \cdot s > 0$  for every  $d(e) \neq (0, 0, \dots, 0)$ .

Since a realizable MDFG can not contain cycles, it is obvious that if a scheduling subspace  $S$  is not empty, then the strictly positive scheduling subspace  $S^+$  associated with  $S$  is also not empty. This can be easily demonstrated by adding the vectors that define the boundaries of  $S$ . The resulting vector is in the interior of  $S$ , not coincident with its borders, therefore, in  $S^+$ . Using the strictly positive scheduling subspace concept, we introduce the method of predicting a legal multi-dimensional retiming function in the next theorem.

**Theorem 3.3** Let  $G = (V, E, d, t)$  be a realizable MDFG,  $S^+$  a strictly positive scheduling subspace for  $G$ ,  $s$  a schedule vector in  $S^+$ ,  $u \in V$  a node with all the incoming edges having non-zero delays. A legal MD retiming  $r$  of node  $u$  is any vector orthogonal to  $s$ .

*Proof:* From the definition of retiming, we have  $d_r(e_1) = d(e_1) - r(u)$  and  $d_r(e_2) = d(e_2) + r(u)$ , for some  $\xrightarrow{e_1} u \xrightarrow{e_2}$ . To verify that the resulting MDFG is realizable, we compute the inner product of  $s$  and each of the retimed dependence vectors. Since  $e_1$  is an

incoming edge and by assumption  $d(e_1) \neq (0, 0, \dots, 0)$  then,  $d_r(e_1) \cdot s = d(e_1) \cdot s - r(u) \cdot s = d(e_1) \cdot s > 0$ . For the outgoing edges  $e_2$ ,  $d_r(e_2) \cdot s = d(e_2) \cdot s + r(u) \cdot s = d(e_2) \cdot s$ . We have now two cases:

Case 1: if  $d(e_2) \neq (0, 0, \dots, 0)$  then  $d(e_2) \cdot s > 0$ , therefore  $d_r(e_2) \cdot s > 0$ .

Case 2: if  $d(e_2) = (0, 0, \dots, 0)$  then  $d_r(e_2) = r(u)$ . Since  $d_r(e_1) \cdot s > 0$  and for each edge  $e$ ,  $d(e) \cdot s > 0$ , it is impossible to have a linear combination of these delay vectors orthogonal to  $s$ ; i.e., parallel to  $d_r(e_2) = r(u)$ . Therefore, no cycle can be found and the conditions of lemma 3.2 have been satisfied, implying that the resulting graph is realizable.  $\square$

Using the above theory on a given set of dependence vectors, after defining its strictly positive scheduling subspace we can predict a legal multi-dimensional retiming function. These results are used in our incremental retiming algorithm. The eventual need of applying the multi-dimensional retiming function to a set of nodes require us to introduce an important corollary from theorem 3.3.

**Corollary 3.4** *Given a realizable MDFG  $G = (V, E, d, t)$ ,  $S^+$  a strictly positive scheduling subspace for  $G$ ,  $s$  a schedule vector in  $S^+$ , and an MD retiming function  $r$  orthogonal to  $s$ , if a set  $X \subseteq V$  has all incoming edges non-zero, then  $r(X)$  is a legal MD retiming.*

*Proof:* For some  $\xrightarrow{e_1} X \xrightarrow{e_2}$ , we know that  $d(e_1) \neq (0, 0, \dots, 0)$ . Considering theorem 3.3 we conclude that  $r$  is a legal MD retiming in  $G$ .  $\square$

### 3.4 The Incremental Multi-Dimensional Retiming Algorithm

The ability to predict a legal multi-dimensional retiming for any realizable MDFG allow us to define the incremental multi-dimensional retiming algorithm below:

---

#### Incremental Multi-Dimensional Retiming Algorithm( $G$ )

1. Given a realizable MDFG  $G = (V, E, d, t)$ , we use definition 3.3 to find a schedule vector  $s$  by solving the inequalities  $d(e) \cdot s > 0$  for every  $e \in E$ . For a two-dimensional problem, we choose  $s = (s.x, s.y)$  such that  $s.x + s.y$  is minimum.
2. Using theorem 3.3, select the MD retiming function from the hyperplane with  $s$  as the normal vector. For a two-dimensional problem we choose  $r = (s.y, -s.x)$ .
3. Apply the selected MD retiming function to any node that has all incoming edges with non-zero delays and at least one outgoing edge with zero delay.

4. Since the resulting MDFG is still realizable, if there is any zero delay edge in the graph, go back to step 1.

---

Using the above algorithm, a fully parallel solution is always achievable. This assertion is proven by the theorem below:

**Theorem 3.5** *Let  $G = (V, E, d, t)$  be a realizable MDFG, the incremental multi-dimensional retiming algorithm transforms  $G$  to  $G_r$ , in at most  $|V|$  iterations, such that  $G_r$  is fully parallel.*

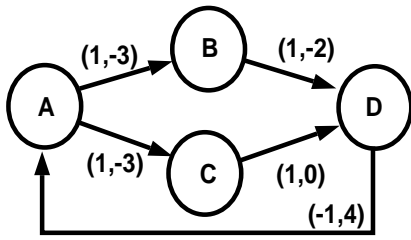
*Proof:* After an iteration of the incremental multi-dimensional retiming algorithm, the resulting MDFG is still realizable. Successive iterations of finding  $s$  and the respective incremental retiming function allow us to modify all zero delay edges to non-zero ones, obtaining a fully parallel MDFG. After each iteration, all outgoing edges of at least one new node will not be zero delay edges. After at most  $|V|$  iterations full-parallelism is achieved.  $\square$

Let's examine the example in Figure 4. The first incremental retiming is shown in Figure 6, where  $s$  has value  $(1, 0)$ . The next retiming must consider the new set of dependencies:  $\{(1, -2), (0, 1), (1, 0)\}$ . We choose  $s = (3, 1)$  then, the retiming function for node A can be either  $(1, -3)$  or  $(-1, 3)$ . Choosing  $r(A) = (1, -3)$  we obtain the graph on Figure 10(a). The equivalent code is not a simple representation of the graph because it requires a non-recursive schedule vector [16], i.e., a schedule vector not parallel to one of the axes in the original index space. We use *wavefront processing* [17] to adjust the execution sequence. The code equivalent to the final retimed graph is shown in Figure 10(b). A sketch of the cell dependence graph for this solution is shown in Figure 11, where the iteration space corresponding to the new loop has been highlighted.

## 4 Chained Multi-Dimensional Retiming

The previous section described a method to find a fully parallel solution in several iterations of a incremental retiming operation. To increase the efficiency of such a method, we introduce new concepts that support the implementation of a new algorithm that allows us to obtain the full parallelism solution in a single pass. We begin by presenting a second corollary from theorem 3.3.

**Corollary 4.1** *Given a realizable MDFG  $G = (V, E, d, t)$ ,  $S^+$  a strictly positive scheduling subspace for  $G$ ,  $s$  a schedule vector in  $S^+$ , a MD retiming function  $r$  orthogonal to*



(a)

```

DO 10 kp = 3, 3*m+n-4
  --- prologue ---
  DO 11 jp = max(0, int((kp-n+3)/3), min(m-1, int((kp-1)/3)))
    d(jp, kp-3*jp+1) = b(jp-1, kp-3*jp+2) * c(jp-1, kp-3*jp)
    a(jp+1, kp-3*jp-3) = d(jp+1, kp-3*jp-3) * .5
    b(jp, kp-3*jp) = a(jp, kp-3*jp) + 1.
    c(jp, kp-3*jp) = a(jp, kp-3*jp) + 2.
  11 CONTINUE
  --- epilogue ---
10 CONTINUE

```

(b)

Figure 10: (a) MDFG fully parallel (b) equivalent code using the new schedule vector

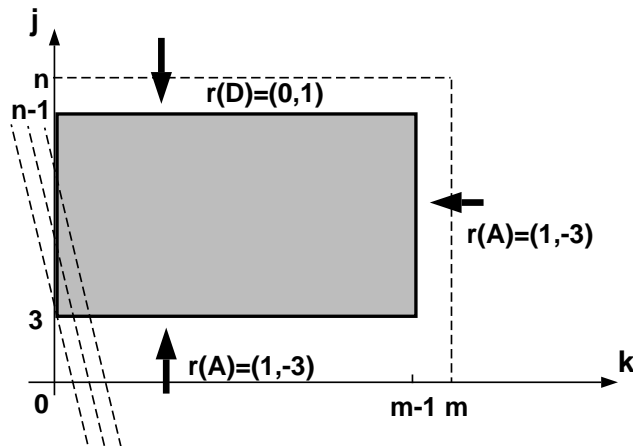


Figure 11: DG with reduced boundaries for the loop due to the retiming functions  $r(D)=(0,1)$  and  $r(A)=(1,-3)$

$s$ , a set  $X \subseteq V$  with all incoming edges non-zero, and an integer value  $k > 1$ , then  $(k \times r)(X)$  is a legal MD retiming.

*Proof:* Proved immediately from theorem 3.3 and corollary 3.4.  $\square$

The above corollary gives us the alternative of retiming a node by a multiple value of the selected retiming function. Intuitively, it is easy to observe that we can produce a realizable MDFG by applying the MD retiming to successive nodes in a path such that each node has a retiming function multiple of the selected retiming value and smaller than its predecessor nodes. This suggests that we must explore a chain reaction operation. We begin by describing how to establish the multiplier constant for the retiming functions.

A topological sort algorithm is used in conjunction with the corollary 4.1 to identify the MD retiming function for each node in the graph. Such function results from considerations on the length of zero delay paths, that we define below as a *chain*.

**Definition 4.1** A *chain* is a path  $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} v_2 \dots \xrightarrow{e_{k-1}} v_k$ , where all incoming edges for node  $v_0$  are non-zero delay edges and every edge  $v_{i-1} \xrightarrow{e_{i-1}} v_i$  is a zero delay edge, for  $1 \leq i \leq k$ . The nodes are numbered in a monotonically increasing order, and  $k$  is said to be the *length* of the chain.

We call a *multi-chain maximum length* the highest level number obtained in the construction process described above. An example of chain can be obtained from Figure 4(a). The existing chains are  $D - A - B$  and  $D - A - C$ . Node  $D$  is at level 0, node  $A$  at level 1, and nodes  $B$  and  $C$  at level 2. Those chains have a multi-chain maximum length of 2. From the concepts seen above we derive a method for identifying all chains belonging to an MDFG. We call this construction a *multi-chain structure* and it is built according to the following steps:

---

### Multi-Chain Construction Algorithm( $G$ )

1. Remove all non-zero delay edges from the MDFG. Since there is no zero-delay cycle for a realizable MDFG, the result is a directed acyclic graph (DAG).
2. Perform a modified topological sort algorithm [30] to order the nodes in levels from the end to the beginning <sup>2</sup>. Each node is labeled according to its level number, which produces the monotonically increasing characteristic of the node indices in a chain.

---

<sup>2</sup>Every node is assigned to a unique level number, even though a node belongs to multiple chains

---

Notice that the topological sort of the graph does not allow any zero delay edge connecting two nodes at the same level. To speed up the incremental retiming process, we introduce the chained retiming technique according to the algorithm below:

---

### Chained Multi-Dimensional Retiming Algorithm( $G$ )

1. Find a legal retiming function  $r$  as in the first step of the incremental retiming algorithm.
  2. Construct the multi-chain structure, labeling the nodes accordingly and computing the multi-chain maximum length  $k$ .
  3. Retime each node  $v$  by  $(k - i) \times r$ , where  $i$  is the level number of  $v$ . The result is the fully parallel MDFG.
- 

For this algorithm, only one step of incremental retiming is executed at the beginning. The rest of the algorithm requires only  $O(|E|)$  times to be performed. Therefore, this algorithm is more efficient than the incremental retiming. The next theorem proves that the MDFG obtained from the chained retiming algorithm is realizable and fully parallel.

**Theorem 4.2** *Let  $G = (V, E, d, t)$  be a realizable MDFG, the chained multi-dimensional retiming algorithm transforms  $G$  to  $G_r$ , such that  $G_r$  is realizable and fully parallel.*

*Proof:* By using the chained multi-dimensional algorithm, we find  $r$ , an incremental retiming for  $G$ ,  $k$  the multi-chain maximum length of  $G$ , and  $i$  the level number assigned to each node  $v \in V$ . The retiming of each node  $v_i$  by  $(k - i) \times r$  results in final dependence vectors of the form  $d(e) - f \times r$ ,  $d(e) + f \times r$ , or  $f \times r$  where  $1 \leq f \leq k - 1$ . By corollary 4.1, the retimed MDFG is realizable, and since there is no zero-delay edges, then  $G_r$  is fully parallel.  $\square$

Revisiting the example in Figure 4, we find out that for  $k = 2$  (length of existing chains) and  $r = (0, 1)$  (first incremental retiming function) we could have applied only one retiming step in such way that node D would be retimed by  $2 \times (0, 1)$ , i.e.,  $(0, 2)$ , and node A would be retimed by  $1 \times (0, 1)$ . The final graph and Fortran code are shown in Figure 12.

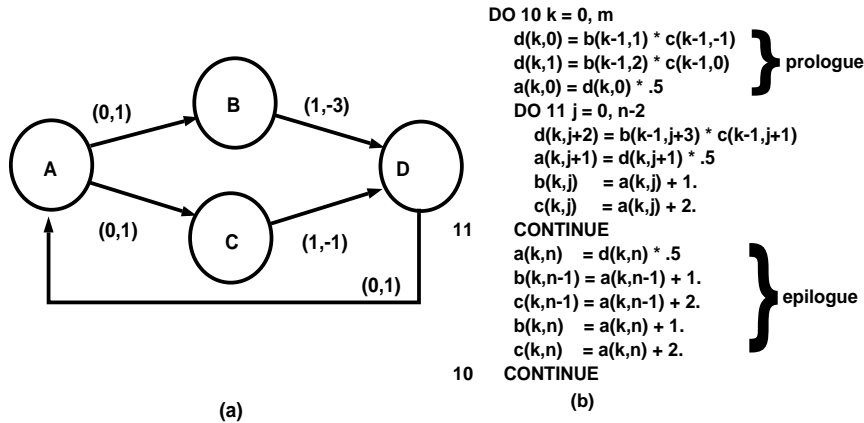


Figure 12: (a) MDFG fully parallel using chained retiming (b) equivalent code

## 5 Loop Transformation

The results of the two previous sections showed that it is possible to obtain the fully parallel solution by applying multi-dimensional retiming. The remaining actions required to obtain the transformed loop have a straightforward implementation and are concisely explained in this section.

### An overview

The entire technique of obtaining the fully parallel solution is based on three stages:

1. The loop is transformed in its equivalent MDFG, which is transformed into a fully parallel graph by one of the multi-dimensional retiming techniques.
2. The prologue, epilogue and new loop body, are constructed according to the retiming functions applied to each node (instruction). The loop indices are also changed accordingly.
3. The loop indices are transformed according to the new schedule vector.

The first stage is accomplished by applying the multi-dimensional incremental retiming or the chained multi-dimensional retiming techniques described in Sections 3 and 4. The second stage consist of the utilization of an existing technique described in [4] and [5], based on operations involving the computed retiming function from the previous stage and the loop and array indices used in the loop body. Such procedure allows the identification of the prologue, the epilogue and the new loop indices bounds. In the third

stage, the loop is transformed in order to comply with the new schedule vector. This transformation can be done by well-known techniques, such as the wavefront processing [17], unimodular transformations [34], loop skewing [32, 33], etc. We submit the example on Figure 4 to the entire procedure in order to demonstrate the application of our technique. For this example, the first stage has been discussed earlier, and the second and third stages are described below.

### Prologue, epilogue and loop body

The prologue and epilogue are direct results of the displacements (multi-dimensional retiming) applied to the nodes (instructions). Therefore an easy way to look at this problem is to add the retiming function  $r$  applied to some instruction  $I$  to the indices of any array been used in  $I$ . For example, the problem in Figure 4 has been retimed by  $r(D) = (0, 1)$  and  $r(A) = (1, -3)$ , producing the fully parallel solution shown in Figure 10. After the first stage of our technique, the instruction

$$d(k, j) = b(k - 1, j + 1) * c(k - 1, j - 1)$$

corresponding to the node  $D$  in the MDFG was retimed by  $r(D) = (0, 1)$ , therefore, the index  $(k, j)$ , applied to the array  $d$ , is replaced by  $(k, j) + (0, 1) = (k, j + 1)$ . The same is done with the remaining array references, resulting the new instruction

$$d(k, j + 1) = b(k - 1, j + 2) * c(k - 1, j).$$

In order to understand this operation, we show in Figure 13 the sequence of displacements caused by  $r(D) = (0, 1)$  and  $r(A) = (1, -3)$ , this last one, for simplicity, divided in two steps: first  $r(A) = (1, 0)$ , followed by  $r(A) = (0, -3)$ . The figures shows an small section of the iteration space with the array elements been computed at each iteration. This allow us to easily identify the displacements of each instruction as well as the lower bounds for the loop indices. Notice that due to the negative component on  $r(A)$ , the instructions  $B$  and  $C$  became also part of the prologue. The shaded regions in the figures represent the new repetitive (fully parallel) loop body. The epilogue, completing the set of instructions, is built in a similar way to the construction of the prologue, and is not presented here.

The loop bounds at this stage would look like the statements below:

$$\begin{aligned} DO\ 10\ k &= 0, m - 1 \\ DO\ 11\ j &= 3, n - 1. \end{aligned}$$

while the loop body would look like

$$\begin{aligned} d(k, j + 1) &= b(k - 1, j + 2) * c(k - 1, j) \\ a(k + 1, j - 3) &= d(k + 1, j - 3) * .5 \\ b(k, j) &= a(k, j) + 1. \end{aligned}$$

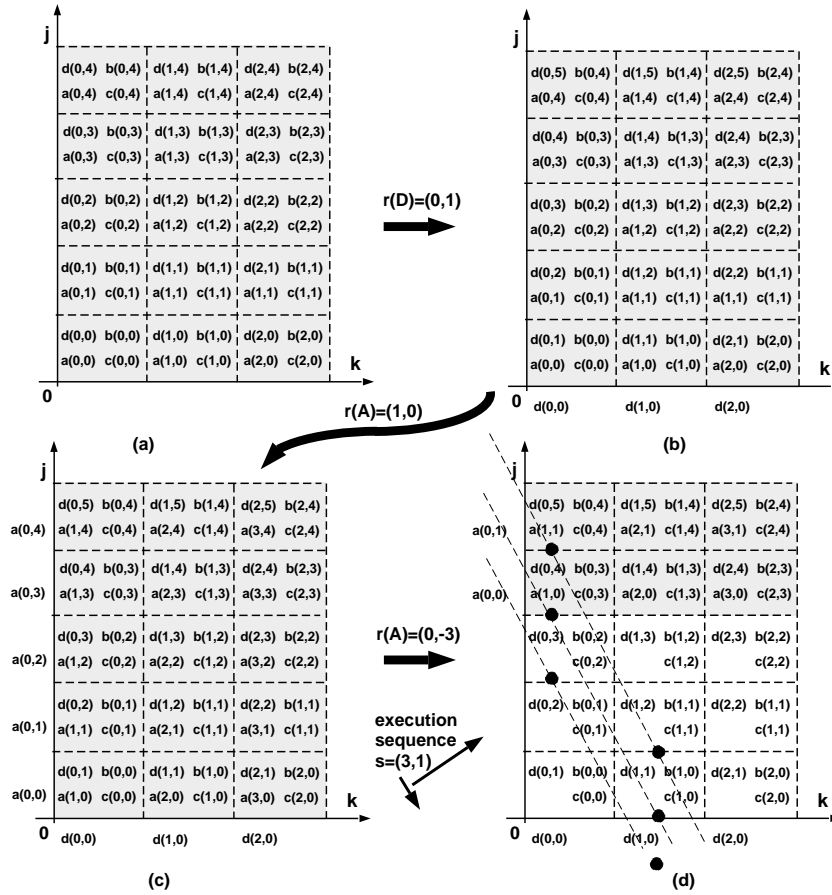


Figure 13: (a) original iteration space (b) after  $r(D)=(0,1)$  (c) after  $r(A)=(1,0)$  (d) after  $r(A)=(0,-3)$

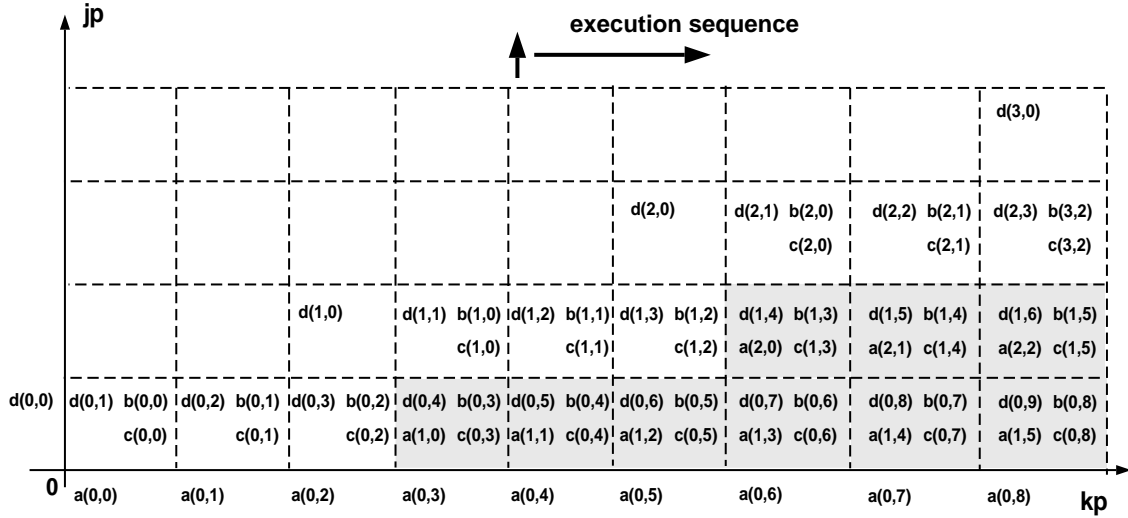


Figure 14: Iteration space after change of schedule vector

$$11 \quad c(k, j) = a(k, j) + 2.$$

### Adjusting the schedule vector

A new order of execution must be adopted in order to follow the sequence imposed by the schedule vector,  $s = (3, 1)$  obtained in the first stage of applying our technique in the example. In order to satisfy this schedule, in this paper, we applied an unimodular transformation [34] with the form

$$T = \begin{bmatrix} s.x & s.y \\ z & w \end{bmatrix}$$

where  $s.x$  and  $s.y$  are the components of the schedule vector and  $z, w$  are chosen in such a way to produce an absolute value of the determinant equal to 1. In our example, the transformation used was

$$T = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix}$$

The resulting iteration space is shown in Figure 14 with its code listed below:

$$\begin{aligned} d(0, 0) &= b(-1, 1) * c(-1, -1) \\ a(0, 0) &= d(0, 0) * .5 \\ d(0, 1) &= b(-1, 2) * c(-1, 0) \\ b(0, 0) &= a(0, 0) + 1. \\ c(0, 0) &= a(0, 0) + 2. \\ a(0, 1) &= d(0, 1) * .5 \\ d(0, 2) &= b(-1, 3) * c(-1, 1) \\ b(0, 1) &= a(0, 1) + 1. \end{aligned}$$

```

c(0, 1) = a(0, 1) + 2.
a(0, 2) = d(0, 2) * .5
d(0, 3) = b(-1, 4) * c(-1, 2)
b(0, 2) = a(0, 2) + 1.
c(0, 2) = a(0, 2) + 2.
d(1, 0) = b(0, 1) * c(0, -1)
DO 10 kp = 3, 3 * m + n - 4
lj = max(0, int((kp - n + 3)/3))
uj = min(m - 1, int((kp - 1)/3))
a(lj, kp - 3 * lj) = d(lj, kp - 3 * lj) * .5
IF((lj > 0).AND.(MOD(kp, 3) = MOD(n - 3, 3)))THEN
    a(lj - 1, n) = d(lj - 1, n) * .5
    b(lj + 1, kp - 3 * (lj + 1)) = a(lj + 1, kp - 3 * (lj + 1)) + 1.
    c(lj + 1, kp - 3 * (lj + 1)) = a(lj + 1, kp - 3 * (lj + 1)) + 2.
ENDIF
DO 11 jp = lj, uj
d(jp, kp - 3 * jp + 1) = b(jp, kp - 3 * jp + 2) * c(jp, kp - 3 * jp)
a(jp + 1, kp - 3 * jp - 3) = d(jp + 1, kp - 3 * jp - 3) * .5
b(jp, kp - 3 * jp) = a(jp, kp - 3 * jp) + 1.
c(jp, kp - 3 * jp) = a(jp, kp - 3 * jp) + 2.
11 CONTINUE
d(uj + 1, kp - 3 * (uj + 1) + 1) = b(uj, kp - 3 * (uj + 1) + 2) * c(uj, kp - 3 * (uj + 1))
b(uj + 1, kp - 3 * (uj + 1)) = a(uj + 1, kp - 3 * (uj + 1)) + 1.
c(uj + 1, kp - 3 * (uj + 1)) = a(uj + 1, kp - 3 * (uj + 1)) + 2.
IF((uj < m - 1).AND.(MOD(kp, 3) = 2))d(uj + 1, 0) = b(uj, 1) * c(uj, -1)
10 CONTINUE

```

## 6 Experiments

In this section we present the application of our method to two different examples. Our first example is the IIR filter introduced in section 1. The second example is an MDFG representing a wave digital filter designed to compute the solution for a partial differential equations problem (a transmission line problem), based on the Fettweis method [12].

### IIR Filter

We now revisit the example shown in figure 1. Assume both devices, adders and multipliers, take one time unit to compute. Using the incremental retiming concept, the first two iterations of our algorithm were presented in section 1. We began by retiming all nodes labeled  $M$  by  $r = (1, -1)$  for a schedule vector  $s = (1, 1)$ . Figure 2(a) shows the resulting MDFG. The new set of dependencies  $D$  is now given by:

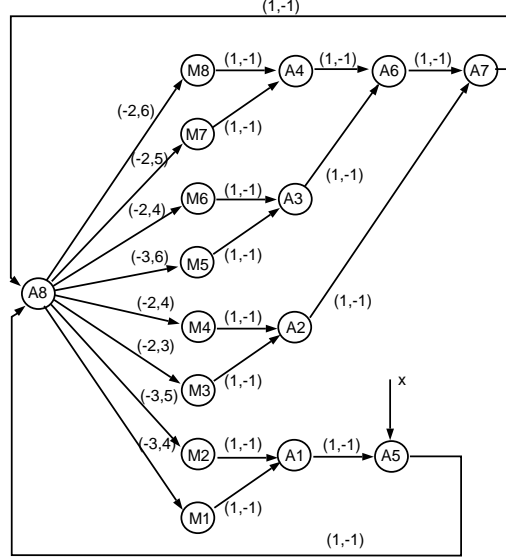


Figure 15: Fully parallel MDFG obtained through chained retiming

$$D = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & 1 \\ 3 & 2 & 1 & 3 & 2 & 1 & 3 & 2 & -1 \end{bmatrix}$$

After retiming nodes  $A1$  to  $A4$  by  $r = (2, -3)$ , for  $s = (3, 2)$ , the graph presented in Figure 2(b) was produced. In the following steps, nodes  $A5$  and  $A6$  were retimed by  $r = (4, -7)$  resulting the graph shown in figure 3(a). The final graph presented in figure 3(b) resulted of retiming node  $A7$  by  $r = (8, -15)$ .

Our second approach is to use the chained retiming algorithm. We find the topological ordering of the DAG associated with the graph in figure 1. Nodes labeled  $M5$  to  $M8$  are assigned to level 0, nodes  $M1$ ,  $M2$ ,  $M3$ ,  $M4$ ,  $A3$  and  $A4$  to level 1,  $A1$ ,  $A2$ , and  $A6$  to level 2,  $A5$  and  $A7$  to level 3, and finally,  $A8$  to level 4. Therefore the multi-chain maximum length of  $G$  is 4. We know that the first incremental retiming function for this set of dependencies is  $(1, -1)$ . Then we may apply the following retiming functions to the graph:  $r(M_i) = (4, -4)$  for  $i = 5$  to 8,  $r(M1) = r(M2) = r(M3) = r(M4) = r(A3) = r(A4) = (3, -3)$ ,  $r(A1) = r(A2) = r(A6) = (2, -2)$ , and  $r(A5) = r(A7) = (1, -1)$ . The final fully parallel graph is then shown in figure 15.

## Transmission Line Simulation

In this example, we introduce a two-dimensional transmission line problem initially transformed by using the Fettweis method (see [12] as a tutorial). The transmission line

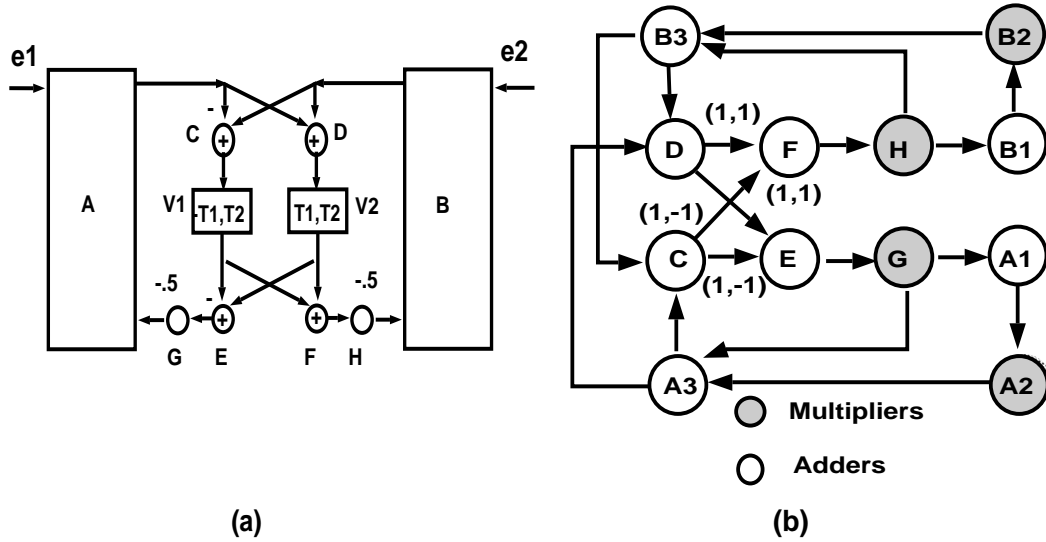


Figure 16: (a) Wave Digital Filter graph (b) equivalent MDFG

is characterized by the equations below:

$$\frac{l\partial i_1}{\partial t_2} + ri_1 + \frac{\partial u}{\partial t_1} = f_1$$

$$\frac{\partial i_1}{\partial t_1} + \frac{c\partial u}{\partial t_2} + gu = f_2$$

After applying the Fettweis transformations we obtain the Wave Digital Filter shown in figure 16(a), which is equivalent to the MDFG in figure 16(b), where the inputs  $e_1$  and  $e_2$  are zero. The two-port adaptors, A and B, are expanded to their internal configuration. For simplicity, one output port and one adder in each adaptor were deleted because the particular boundary conditions applied. Using the incremental algorithm we would obtain the graph shown in figure 17(a), after incrementally applying the following retiming functions:  $r(F) = r(E) = (0, 1)$ ,  $r(H) = r(G) = (1, -3)$ ,  $r(B1) = r(A1) = (2, -7)$ ,  $r(B2) = r(A2) = (4, -15)$ , and  $r(B3) = r(A3) = (8, -31)$ . Figure 17(b) shows the result of applying the chained retiming method to this case. The retiming functions were  $r(F) = r(E) = (0, 5)$ ,  $r(H) = r(G) = (0, 4)$ ,  $r(B1) = r(A1) = (0, 3)$ ,  $r(B2) = r(A2) = (0, 2)$ , and  $r(B3) = r(A3) = (0, 1)$ .

## 7 Conclusion

We have presented two novel techniques for optimizing a multi-dimensional data flow graph through the use of a multi-dimensional retiming method that we developed. We call

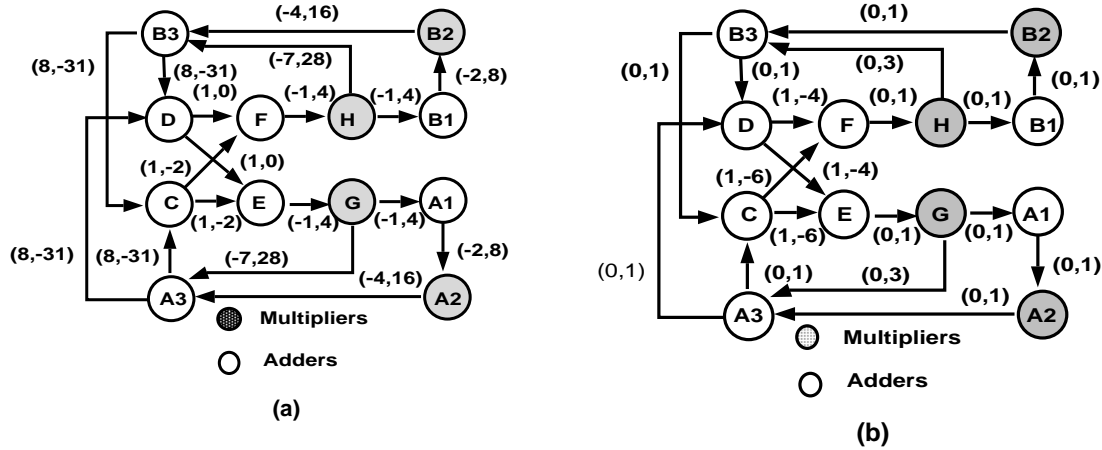


Figure 17: Final MDFG for the transmission line problem, (a) using incremental retiming (b) using chained retiming

these methods *incremental multi-dimensional retiming* and *chained multi-dimensional retiming*. Both techniques are able to achieve full parallelism, i.e., simultaneous execution of all operations (nodes) for either a multi-dimensional data flow graph or an uniform nested loop. The main technique consists of successive retiming operations to reduce the critical path length to one node, which implies the fully parallel solution. A pre-selected retiming function is the core of the algorithm. The process of selecting such a function was presented.

Differently from one-dimensional case where the retiming technique can not guarantee full parallelism, the main theorem of this paper shows that a multi-dimensional problem can always achieve full parallelism. The algorithms were presented in detail. Some examples showed the effectiveness of the new methods in optimizing the execution of loop bodies and MD data flow graphs in parallel environments.

## References

- [1] A. Aiken, "Compaction Based Parallelization". (Ph.D. thesis), Technical Report 88-922, Cornell University, 1988.
- [2] 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.
- [3] U. Banerjee, “ Unimodular Transformations of Double Loops,” in *Advances in Languages and Compilers for Parallel Processing*, Cambridge, Massachusetts, MIT Press, 1991, pp. 192-219.
  - [4] L.-F. Chao and E. H.-M. Sha, “ Static Scheduling of Uniform Nested Loops,” *Proceedings of 7th International Parallel Processing Symposium* , Newport Beach, CA, April, 1993, pp. 1421-1424.
  - [5] L.-F. Chao, “ Scheduling and Behavioral Transformations for Parallel Systems,” Ph.D. dissertation, Princeton University, 1993.
  - [6] L.-F. Chao, A. LaPaugh, and E. H.-M. Sha, “ Rotation Scheduling: A Loop Pipelining Algorithm,” *Proc. 30th ACM/IEEE Design Automation Conference* , Dallas, TX, June, 1993, pp. 566-572.
  - [7] L.-F. Chao and E. H.-M. Sha, “ Retiming and Unfolding Data-Flow Graphs,” *Proc. 1992 International Conference on Parallel Processing*, St. Charles, Illinois, August 1992, pp. II 33-40.
  - [8] L.-F. Chao and E. H.-M. Sha, “ Unified Static Scheduling on Various Models,” *Proc. 1993 International Conference on Parallel Processing*, St. Charles, Illinois, August 1993, pp. II 231-235.
  - [9] E. Cohen and Nimrod Megiddo, “ Strongly Polynomial-Time and NC Algorithms for Detecting Cycles in Dynamic Graphs,” *Proc. 21th ACM Annual Symposium on Theory of Computing*, 1989, pp. 523-534.
  - [10] R. Cytron, “ Doacross: Beyond Vectorization for Multiprocessors”. *Proc. International Conference on Parallel Processing*, 1986, pp. 836-844.
  - [11] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1984.
  - [12] A. Fettweis and G. Nitsche, “ Numerical Integration of Partial Differential Equations Using Principles of Multidimensional Wave Digital Filters.” *Journal of VLSI Signal Processing* , 3, pp. 7-24, 1991.

- [13] J. A. Fisher and B. R. Rau, "Instruction-Level Parallel Processing". *Science*, Vol. 253, September 1991, pp. 1233-1241.
- [14] G. Goosens, J. Wandewalle, and H. de Man "Loop Optimization in Register Transfer Scheduling for DSP Systems," *Proc. ACM/IEEE Design Automation Conference*, 1989, pp. 826-831.
- [15] S. R. Kosaraju and G. F. Sullivan, "Detecting Cycles in Dynamic Graphs in Polynomial Time," *Proc. 20th ACM Annual Symposium on Theory of Computing*, 1988, pp. 398-406.
- [16] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice Hall, 1988.
- [17] L. Lamport, "The Parallel Execution of DO Loops". *Communications of the ACM SIGPLAN*, 17(2) February 1974, pp. 82-93.
- [18] M. Lam, "Software Pipelining: An Effective Scheduling for VLIW Machines". *ACM SIGPLAN Conference on Prog. Lang. Design and Implementation*, 1988, pp. 318-328.
- [19] T.-F. Lee, A. C.-H. Wu, D. D. Gajski, and Y.-L. Lin, "An Effective Methodology for Functional Pipelining". *Proc. of the International Conference on Computer Aided Design*, December, 1992, pp. 230-233.
- [20] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry". *Algorithmica*, 6, 1991, pp. 5-35.
- [21] D. I. Moldovan and J. A. B. Fortes, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays". *IEEE Transactions on Computers*, January 1986, vol. c-35, pp. 1-12.
- [22] A. Nicolau, "Loop Quantization or Unwinding Done Right," *Proc. of the 1987 ACM International Conference on Supercomputing*, Springer Verlag Lecture Notes on Computer Science 289, May 1987, pp. 294-308.
- [23] N. Park and A. C. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications". *IEEE Transactions on CAD*, Vol. 7, No. 3, 1988, pp. 356-370.
- [24] N. L. Passos, E. H.-M. Sha, and S. C. Bass, "Schedule-Based Multi-dimensional retiming". To appear in *Proceedings of 8th International Parallel Processing Symposium*, Cancun, Mexico, April, 1994.

- [25] N. L. Passos and E. H.-M. Sha “ Full Parallelism in Uniform Nested Loops using Multi-Dimensional Retiming”. *Proceedings of 23rd International Conference on Parallel Processing*, August, 1994, vol. II, pp. 130-133.
- [26] N. L. Passos, E. H.-M. Sha, and S. C. Bass, “ Loop Pipelining for Scheduling Multi-Dimensional Systems via Rotation”. To appear in *Proceedings of 31th Design Automation Conference*, San Diego, CA, June, 1994.
- [27] N. L. Passos, E. H.-M. Sha, and S. C. Bass, “ Partitioning and Retiming of Multi-Dimensional Systems”. To appear in *Proceedings of the IEEE International Conference on Circuits and Systems*, London, England, May, 1994.
- [28] R. Potasman, J. Lis, A. Nicolau, and D. Gajski, “ Percolation Based Scheduling”. *Proc. ACM/IEEE Design Automation Conference* , 1990, pp. 444-449.
- [29] D. A. Schwartz, “ Cyclo-Static Realizations, Loop Unrolling and CPM: Optimal Multiprocessor Scheduling.” *Technical report, Georgia Institute of Technology - School of Electrical Engineering*, 1987.
- [30] R. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, Pennsylvania, 1983.
- [31] C.-Y. Wang and K. K. Parhi, “ High Level DSP Synthesis Using the MARS Design System”. *Proc. of the International Symposium on Circuits and Systems*, 1992, pp. 164-167.
- [32] M. Wolfe, “ Loop Skewing: the Wavefront Method Revisited”. *International Journal of Parallel Programming*, Vol. 15, No. 4, August 1986, pp. 284-294.
- [33] M. Wolfe, *Optimizing Supercompilers for Supercomputers*, MIT Press, Cambridge, MA, 1989.
- [34] M. E. Wolf and M. S. Lam, “ A Loop Transformation Theory and an Algorithm to Maximize Parallelism”. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, October 1991, pp. 452-471.