

Schedule-Based Multi-Dimensional Retiming on Data Flow Graphs

Nelson Luiz Passos
Edwin Hsing-Mean Sha
Steven C. Bass

Research Report CSE-TR-93-012
October 1993

Abstract

Transformation techniques are usually applied to get optimal execution rates in parallel and/or pipeline systems. The retiming technique is a common and valuable tool in one-dimensional problems, represented by Data Flow Graphs (DFGs) such as DSP filters, which can maximize the parallelism of a loop body represented by a DFG. Since most scientific or DSP applications are recursive or iterative, to increase the parallelism of the loop body can substantially decrease the overall computation time. Few results on retiming have been obtained for multi-dimensional problems. The previous result of multi-dimensional retiming is only applied to a restricted class of Data Flow Graphs in which every total delay vector in a cycle has to be strictly non-negative. This paper develops a novel retiming technique that considers the final schedule as part of the process. To authors' knowledge, this is the first retiming algorithm for general multi-dimensional Data Flow Graphs. The description and the correctness of our algorithm are presented in the paper. Through the experiments, results have shown that our algorithm runs efficiently. Some DSP filters are used in the paper as an example of the application of our algorithm.

1 Introduction

With the evolution of image processing applications (multi-dimensional signal processing) and the availability of parallel computing hardware, the demand for optimized solutions for multi-dimensional problems has increased. Efficient scheduling 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 to regroup operations in iterations in order to produce a new iteration structure with higher parallelism embedded [6, 15]. An equivalent approach to the multi-dimensional case will benefit parallel compiler design for VLIW, data-flow, and superscalar architectures [1, 9, 11], as well as high-level synthesis for VLSI design [5, 14].

Some research has been done on uniform nested loop scheduling, a similar view to multi-dimensional problem. For example, loop skewing [17] and loop quantization [2]. These techniques differ from our method since they don't change the structure of iterations, but the sequence in which the instructions are executed.

Other techniques that search beyond loop boundaries include perfect pipelining [1] and Doacross loops [7]. 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 our technique.

In our study, loop bodies are generically represented by multi-dimensional data flow graphs (MDFG). Instead of simply overlapping iterations, we restructure the loop body, i.e., the existing dependence delays, from a global view, preserving the original data dependence. We model the process of restructuring as a multi-dimensional retiming. When the new MDFG is executed, the iterations of the original one are implicitly, naturally overlapped, and the parallelism is explicitly displayed. The new MDFG can be constructed from the original graph through the retiming function.

Our study focuses on the characteristics of the retiming technique that can be applied to multi-dimensional problems. Chao and Sha have introduced the basic principles under the constraints of a specific class of problems where the summation of delay vectors in a cycle was always non-negative [4]. We extend such concepts to include a general form of Data Flow Graph, and derive a practical method of finding a multi-dimensional retiming function.

This paper build the framework of having a legal multi-dimensional retiming under general DFGs. To the authors knowledge, this is the first framework for such purpose. As the success of using traditional retiming of one-dimensional DFG (or circuits) in many applications [12], the generalization of our technique to other areas is promising.

Under our framework, in order to use an existing efficient linear-programming solver, a technique based on characteristics of the final execution schedule is developed. The experiments show impressive results on the parallelism obtained from the transformation of MDFGs.

For simplicity, the retiming technique is studied for the two-dimensional case where the dimensions are generically referred as x and y . The multi-dimensional case is a straight forward extension of the concepts presented.

In section 2, we recall some basic concepts, such as mathematical models for Dependence Graphs and Data Flow Graphs presented in [3, 4, 10, 13], and a review of arithmetic operations in two-dimensional vectors. Also, we mention the basic ideas of multi-dimensional retiming, and cycle period. Such concepts are useful on the following sections when developing the algorithm for *schedule-based multi-dimensional retiming*.

Section 3 introduces definitions and properties necessary to have a legal retiming function. However, these properties are not sufficient to implement an efficient algorithm. Section 4 redefines such properties, introducing the idea of an exclusion angle that reduces the domain of our retiming function, but give us the foundations required for an Integer Linear Program (ILP) solution. A final conclusion summarizes the concepts introduced in this paper and discuss its applicability.

2 Basic Principles

2.1 Background

In this section we recall some basic concepts and definitions on the interpretation of multi-dimensional data flow graph, such as mathematical models, dependence vectors, iterations, and vector operations.

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 delays 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. We adopted $d(e) = (d.x, d.y)$ as a general formulation of any delay shown in a two-dimensional DFG. An example of a two-dimensional DFG is shown in figure 1, representing a subgraph of a Wave Digital Filter described later in section 5. For this example, $V = \{A, B, C, D\}$ and $E = \{e1 : (A, B), e2 : (A, C), e3 : (D, A), e4 : (B, D), e5 : (C, D)\}$ where, $d(e1) = d(e2) = d(e3) = (0, 0)$, $d(e4) = (-1, 1)$, $d(e5) = (1, 1)$, and each operation is assumed to

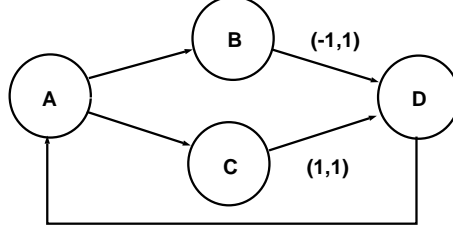


Figure 1: MDFG extracted from a Wave Digital Filter graph

be executed in one time unit, therefore, $t(A) = t(B) = t(C) = t(D) = 1$.

An *iteration* is the execution of each node in V exactly once. Iterations are identified by a vector i , equivalent to a multi-dimensional index, starting from $(0, 0, \dots, 0)$. Inter-iteration dependences are represented by vector-weighted edges.

An equivalent *cell dependence graph (DG)* G_{dg} of a MDFG G is the directed acyclic graph showing the dependences between infinite copies of nodes representing the MDFG. Figure 2(a) shows the replication of the MDFG in figure 1, and figure 2(b) shows the same DG but with each node representing a copy of the MDFG. We call a *computational cell* the DG node that represents a copy of a MDFG, excluding the edges with delay vectors different of $(0, 0, \dots, 0)$, i.e., a complete iteration. A cell is considered an atomic execution unit.

A two-dimensional data flow graph (2DFG) $G = (V, E, d, t)$ is a MDFG, where d is a function from E to Z^2 . A mathematical model for a cell dependence graph expanded from a 2DFG is presented in [13]: a tuple (I^2, D) where I^2 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 (delay vectors). Consider, for example, the DG shown in Figure 2; its model will be described as:

$$I^2 = \{(i, j) : 0 \leq i \leq 3, 0 \leq j \leq 3\}, \text{ and } D = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$$

For any iteration j in I^2 , an edge e from u to v with delay vector $d(e)$ means that the computation of node v at iteration j depends on the execution of node u at iteration $j - d(e)$. An edge with delay $(0, 0)$ represents a data dependence within the same iteration (computational cell). A legal MDFG must have no zero-delay cycle, i.e., the summation of the delay vectors along any cycle can not be $(0, 0, \dots, 0)$.

The notation $u \xrightarrow{e} v$ means 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 $p = v_0 \xrightarrow{e_0}$

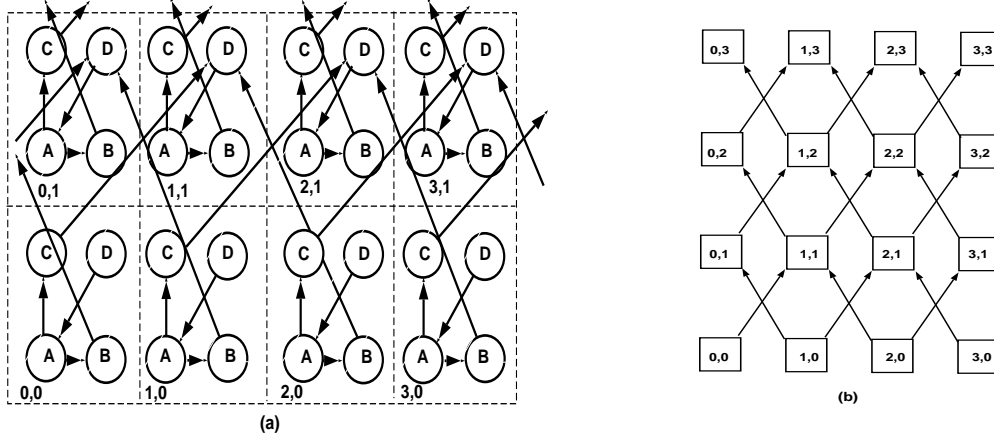


Figure 2: (a) DG based on the replication of a MDFG, showing iterations starting at $(0,0)$. (b) DG represented by computational cells

$v_1 \xrightarrow{e_1} v_2 \dots \xrightarrow{e_{k-1}} v_k$ is $d(p) = \sum_{i=0}^{k-1} d(e_i)$ and the total computation time of a path p is $\sum_{i=0}^k t(v_i)$.

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

In this subsection, we review the basic ideas on retiming, including considerations on cycle period, critical path, and characteristics of legal retiming.

The period during which all computation nodes in an iteration are executed, according to existing data dependences and without resource constraints, is called a *cycle period*. The *cycle period* $C(G)$ of a MDFG $G = (V, E, d, t)$ is the maximum computational time among paths that have no delay, i.e. $C(G) = \max\{t(p) | p \text{ is a path in } G \text{ with } d(p) = (0, 0, \dots, 0)\}$. Another way to specify the *cycle period* is based on its upper bound: $C(G) \leq c$ if and only if for every path $p \in G$, if $d(p) = (0, 0, \dots, 0)$ then $t(p) \leq c$, where c is a positive constant. For example, the MDFG in figure 1 has $C(G) = 3$, which can be measured through the paths $p = D \xrightarrow{e_3} A \xrightarrow{e_1} B$ or $p = D \xrightarrow{e_3} A \xrightarrow{e_2} C$.

A *two-dimensional retiming* r is a function from V to Z^2 that redistributes the nodes

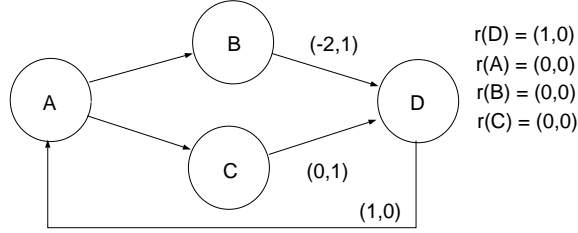


Figure 3: MDFG after retimed by r

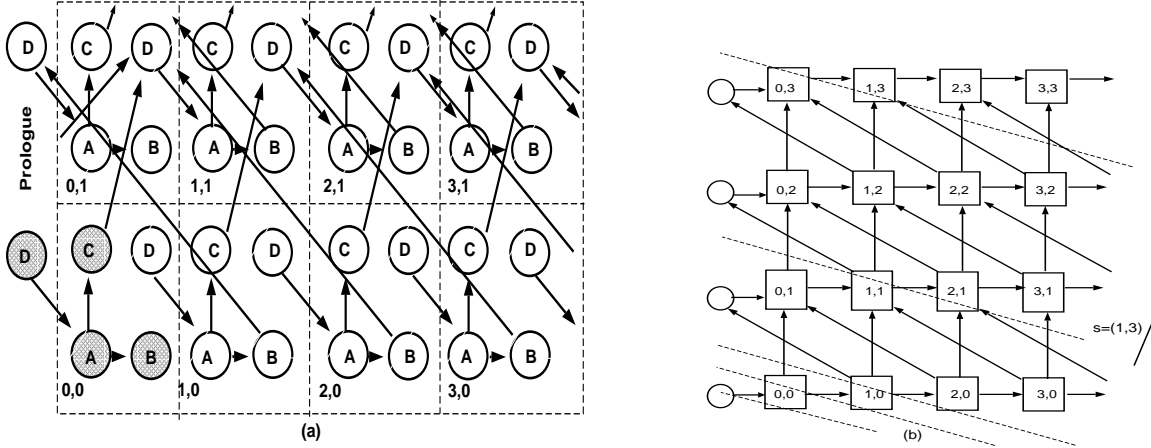


Figure 4: (a) DG based on the replication of a MDFG, after retiming. (b) same DG represented by computational cells.

in the original dependence graph created by the replication of a 2DFG G , resulting a new 2DFG G_r , such that each iteration still has one execution of each node in G , reducing the cycle period to a minimum as in the one-dimensional case. The retiming vector $r(u)$ of a node $u \in G$ represents the offset between the original iteration 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 $i - r(u)$. For example, figure 3 shows the MDFG in figure 1 retimed by the function r . Figure 3 shows that the critical path of this graph are the edges $A \rightarrow B$ and $A \rightarrow C$ which have an execution time of 2 time units, which was reduced from the original 3 time units.

On the one-dimension retiming, the existing constraint not allowing negative delays is easily explained by verifying that such delays would introduce a cycle in the equivalent DG. However, on the multi-dimensional case, the existence of negative delays is allowed

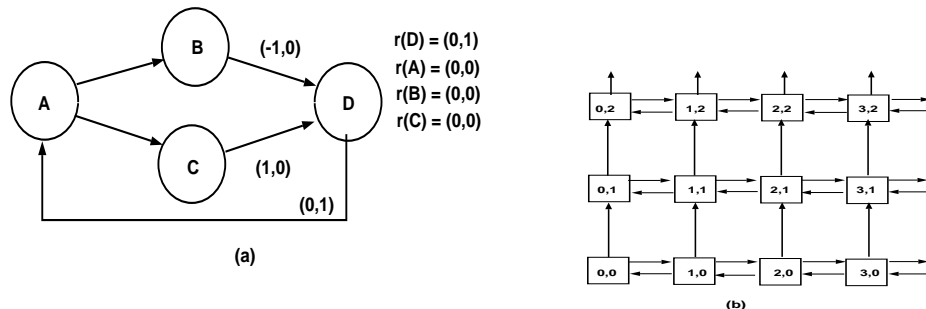


Figure 5: (a) Example of illegal retiming. (b) DG showing cycles in the x-direction.

if there exists a schedule s that turns the DG realizable. The retimed cell DG, is shown in figure 4(a) and (b), where the nodes originally belonging to iteration $(0,0)$ are marked. As we see, a possible schedule for the retimed graph is $s = (1, 3)$.

Figure 5(a) shows an illegal retiming function applied to the same example in figure 1. By simple inspection of the cell dependence graph presented in figure 5(b) we notice the existence of a cycle created by dependences $(1, 0)$ and $(-1, 0)$.

A *prologue* is the set of instructions that are moved on directions x and y , in a two-dimensional retiming, and that must be executed to provide the necessary data for the iterative process. In our example shown in figure 4, the instruction D becomes the prologue for that problem. We call *epilogue* the other extreme of the DG, were a complementary set of instructions will need to be executed to complete the process. In our example, the epilogue will include operations A,B and C. Considering that the entire problem consists of a large number of iterations, we may assume that the time required to run the prologue and epilogue are negligible, and in this case, the reduction of the critical path by one third, shown in figure 3, may result in a saving of approximately one third of the total execution time, depending on the target computer architecture.

Previous study of *multi-dimension retiming functions* were restricted to a specific class of multi-dimensional Data Flow Graphs where the total delay vector of any cycle was non-negative in every dimension. We extend those concepts to a general legal MDFG where the total delay of any cycle is non-zero, i.e. $d(l).x \neq 0$ and/or $d(l).y \neq 0$ for every cycle l in G .

3 Schedule-Based Two-dimensional Retiming

The technique of multi-dimensional retiming redistributes nodes, in a MDFG G , to iterations, such that each iteration still has one execution of each node, preserving the dependences and allowing the reduction of the cycle period of each iteration. Differently from the one-dimensional case, the retiming operation in a MDFG allows changes in more than one direction, but requiring the final result to be compatible with some schedule to guarantee the realizability of the design. This section introduces the definitions and properties for the schedule-based retiming.

To find a legal retiming of a MDFG, such that its cycle period is at most a value c , is more complex than the use of traditional retiming on one-dimensional problems. In the traditional method, the positive delays after retiming guarantee the retiming as legal. For the multi-dimensional case, however, positive delay vectors are too restrictive since a dependence graph is still realizable even if it has negative delays. Therefore, the following theorem shows us a more general set of constraints for a legal retiming.

Theorem 3.1 *Let $G = (V, E, d, t)$ be a MDFG, and c a positive integer. r is a legal retiming for G such that $C(G_r) \leq c$ 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. *given a cycle period c , if the execution time of a path p in G_r is greater than c , then $d_r(p) \neq (0, 0, \dots, 0)$.*

Proof: 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 condition results from the properties associated to the cycle period. \square

Our technique enforces these two constraints through the use of a schedule vector that supports the realization of the retimed graph and does not affect the cycle period.

Definition 3.1 *A feasible linear schedule s for a MDFG $G = (V, E, d, t)$, is the vector pointing to the normal direction of the equitemporal hyperplanes representing the sequence of execution of each cell in the equivalent cell dependence graph G_{dg} .*

The following properties are derived from the characteristics of a *feasible linear schedule*:

1. if node v depends on node u , where $u, v \in V$, then $d(u, v) \cdot s \geq 0$, i.e., there exists a sequence of execution that does not violate the dependences.

2. if cell g depends on cell f , where $f, g \in G_{dg}$, then f, g can not be in the same equitemporal hyperplane, i.e., if $u \in f$ and $v \in g$ then $d(u, v) \cdot s > 0$ to avoid cycles in the cell dependence graph.
3. if node v depends on node u , where $u, v \in V$, and $d(u, v) \cdot s = 0$ then $d(u, v) = (0, 0, \dots, 0)$, which is an immediate consequence of the two conditions above.

Definition 3.2 A *schedule-based two-dimensional retiming* r for a cycle period c is a function from V to Z^2 that redistributes the nodes in the original cell dependence graph of a 2DFG G , resulting a new 2DFG G_r , realizable according to the schedule s .

The retiming vector $r(u)$ works exactly as previously defined. The only extension in this new definition is the requirement for a feasible linear schedule compatible with the retimed graph. It is obvious that a schedule-based two-dimensional retiming is a legal retiming, according to theorem 3.1, since a cell dependence graph can not have a cycle, while having a legal schedule.

Lemma 3.2 introduces the properties that will be used in the proof of theorem 3.6 which shows how to enforce the correctness of the retiming function.

Lemma 3.2 *Let $G = (V, E, d, t)$ be a 2DFG, r a two-dimensional retiming, and s a feasible schedule vector for the retimed graph G_r . The retiming function r on the 2DFG G , resulting the 2DFG $G_r = (V, E, d_r, t)$ is characterized by:*

- (a) for any path $u \xrightarrow{p} v$, 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 $u \xrightarrow{e} v$, if $d_r(e) \neq (0, 0)$ then $d_r(e) \cdot s > 0$

Lemma 3.3 *Let $G = (V, E, d, t)$ be a 2DFG. If s is a feasible linear schedule vector for G then for any path $u \xrightarrow{p} v$, we have $d(p) \cdot s \geq 0$*

Proof: For $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} v_2 \dots \xrightarrow{e_{k-1}} v_k$ we know that $d(p) = \sum_{i=0}^{k-1} d(e_i)$. If we apply s to both sides we have $d(p) \cdot s = \sum_{i=0}^{k-1} d(e_i) \cdot s$. Since a feasible linear schedule s requires $d(e) \cdot s \geq 0$ the proof is immediate. \square

We define below, the functions $D^{(s)}$ and $T^{(s)}$ to be used to identify a set of critical paths, such that the retiming problem can be solved focusing on these critical paths.

Definition 3.3 We define the *function* $D^{(s)}(u, v)$ for a MDFG $G = (V, E, d, t)$, from $V \times V$ to N , as the minimum number of temporal hyperplanes between nodes u and v in V , according to the schedule vector s . We can formulate such a definition as:

$$D^{(s)}(u, v) = \min\{d(p) \cdot s \mid u \xrightarrow{p} v \in G\}$$

Definition 3.4 Given a MDFG $G = (V, E, d, t)$, $u, v \in V$, we define the function $T^{(s)}(u, v)$ from $V \times V$ to N , as the maximum computation time between nodes u and v , inclusive, according to the schedule vector s , such that $p = u \xrightarrow{\mathcal{P}} v$ and $d(p) \cdot s = D^{(s)}(u, v)$. We can formulate such a definition as:

$$T^{(s)}(u, v) = \max\{t(p) \mid u \xrightarrow{\mathcal{P}} v \in G \text{ and } d(p) \cdot s = D^{(s)}(u, v)\}$$

A slight modification of a shortest path algorithm can compute $D^{(s)}$ and $T^{(s)}$. The paths $u \xrightarrow{\mathcal{P}} v$ such that $d(p) \cdot s = D^{(s)}(u, v)$ and $t(p) = T^{(s)}(u, v)$ are considered the critical paths from u to v . The next lemma shows that the cycle period is independent of the schedule vector.

Lemma 3.4 Let $G = (V, E, d, t)$ be a MDFG, c a positive integer, and s a feasible schedule vector for the graph G . $C(G) \leq c$ if and only if for every path $p \in G$ if $d(p) \cdot s = 0$ then $t(p) \leq c$.

Proof: From the definition of cycle period we know that

$$C(G) = \max\{t(p) \mid d(p) = (0, 0) \text{ for every path } p \in G\}$$

Since we know that $d(p) \cdot s = 0$ if and only if $d(p) = (0, 0)$, then we may rewrite the above relation as:

$C(G) = \max\{t(p) \mid d(p) \cdot s = 0 \text{ for every path } p \in G\}$, therefore, $C(G) \leq c$ if and only if for every path $p \in G$ if $d(p) \cdot s = 0$ then $t(p) \leq c$. \square

The following theorem expresses the cycle period of G in terms of functions $D^{(s)}$ and $T^{(s)}$.

Theorem 3.5 Let $G = (V, E, d, t)$ be a 2DFG, s a feasible linear schedule vector for the graph G , and c a positive integer. Then the cycle period $C(G) \leq c$ if and only if for all nodes u and v in V , if $D^{(s)}(u, v) < 1$ then $T^{(s)}(u, v) \leq c$

Proof: To prove the only if part by contradiction, we assume $C(G) \leq c$ and that there exist $u, v \in V$ such that $D^{(s)}(u, v) < 1$ and $T^{(s)}(u, v) > c$. Consider p a critical path from u to v with $d(p) \cdot s = D^{(s)}(u, v)$ and $t(p) = T^{(s)}(u, v)$. Thus, there exists a path p in G , such that $t(p) > c$ and $d(p) \cdot s < 1$. From lemmas 3.4 and 3.3, $d(p) \cdot s = 0$, and $C(G)$ must be greater than c , which is a contradiction.

For the if part, we assume that for all $u, v \in V$ if $D^{(s)}(u, v) < 1$ then $T^{(s)}(u, v) \leq c$. We claim that for every path p in G , either $d(p) \cdot s \geq 1$ or $t(p) \leq c$, which implies $C(G) \leq c$, from lemmas 3.4 and 3.3. Let p be a critical path in G , then we know that $t(p) \leq c$, then the claim is true. Otherwise, if p is not a critical path in G , we have two

cases:

case 1: if $d(p) \cdot s = D^{(s)}(u, v)$ then $t(p) \leq T^{(s)}(u, v) \leq c$,

case 2: if $d(p) \cdot s \geq D^{(s)}(u, v) + 1$ then $d(p) \cdot s \geq 1$ since $D^{(s)}(u, v) \geq 0$.

Therefore, $d(p) \cdot s \geq 1$ or $t(p) \leq c$. \square

The next theorem shows the necessary and sufficient conditions for a retiming with $C(G_r) \leq c$ in terms of $D^{(s)}$ and $T^{(s)}$.

Theorem 3.6 *Let $G = (V, E, d, t)$ be a 2DFG, r a two-dimensional retiming on G , s a feasible linear schedule vector for the retimed graph G_r , and c a positive integer. Then r is a legal schedule-based retiming on G , such that $C(G_r) \leq c$ if and only if*

(a) $r(v) \cdot s - r(u) \cdot s < d(e) \cdot s$ or $r(v) - r(u) = d(e)$, for every edge $u \xrightarrow{e} v$
and

(b) if $T^{(s)}(u, v) > c$ then $r(v) \cdot s - r(u) \cdot s < D^{(s)}(u, v) - 1$, for every pair u, v in G

Proof: For part (a), we know from Lemma 3.2 that r is a legal retiming if and only if $d_r(e) \cdot s > 0$ or $d_r(e) = (0, 0)$. Since $d_r(e) = d(e) + r(u) - r(v)$, the proof is obvious.

For part (b), we begin to prove that the retiming function effects on $D^{(s)}$ and $T^{(s)}$ are in the same way as on functions d and t , i.e.,

1. (1) $D_r^{(s)}(u, v) = D^{(s)}(u, v) + r(u) \cdot s - r(v) \cdot s$
2. (2) $T_r^{(s)} = T^{(s)}$

For (1), $D_r^{(s)}(u, v) = \min\{d_r(p) \cdot s \mid u \rightsquigarrow^p v \in G_r\}$, since $d_r(p) = d(p) + r(u) - r(v)$, we have

$$D_r^{(s)}(u, v) = \min\{d(p) \cdot s + r(u) \cdot s - r(v) \cdot s \mid u \rightsquigarrow^p v \in G_r\},$$

$$D_r^{(s)}(u, v) = \min\{d(p) \cdot s \mid u \rightsquigarrow^p v \in G_r\} + r(u) \cdot s - r(v) \cdot s,$$

$$D_r^{(s)}(u, v) = D^{(s)}(u, v) + r(u) \cdot s - r(v) \cdot s$$

For (2), $T_r^{(s)} = \max\{t(p) \mid u \rightsquigarrow^p v \in G_r \text{ and } d_r(p) \cdot s = D_r^{(s)}(u, v)\}$,
similarly to (1), $T_r^{(s)} = T^{(s)}$

From theorem 3.5 we have $C(G) \leq c$ if and only if for all nodes u and v in V , if $D_r^{(s)}(u, v) < 1$ then $T_r^{(s)}(u, v) \leq c$. Substituting according (1) and (2), we obtain our claim: if $T^{(s)}(u, v) > c$ then $r(v) \cdot s - r(u) \cdot s < D^{(s)}(u, v) - 1$, for every path $u \rightsquigarrow^p v$ in G . \square

At this point, we have the formulation of the constraints for a legal multi-dimensional retiming function. We reduce the universe of possible schedule vectors, adding on lemma 3.7 a constraint on the choice of such vectors, based on the fact that the summation of delay vectors in a existing cycle, in the MDFG, remains constant through the retiming operation (lemma 3.2).

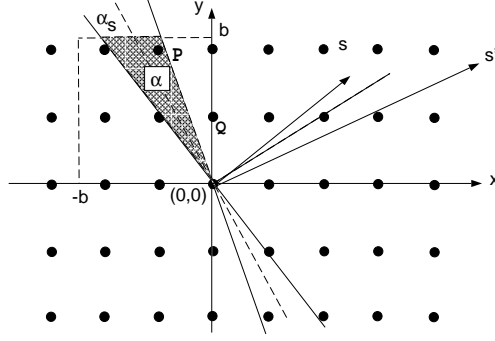


Figure 6: Region where dependence vectors are not allowed

Lemma 3.7 *Given a 2DFG $G = (V, E, d, t)$, and r a two-dimensional retiming on G , then a feasible linear schedule s for G_r must have $d(l) \cdot s > 0$ for any cycle $l \in G$.*

Proof: From the definition of s , $d_r(e) \cdot s > 0$ or $d_r(e) = (0, 0)$. Since $d_r(l) \neq (0, 0)$ due to realizability constraints, and $d_r(l) = \sum_{i=0}^{k-1} d_r(e_i)$ for $l = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} v_2 \dots \xrightarrow{e_{k-1}} v_k$, with $v_k = v_0$, then $\sum_{i=0}^{k-1} (d_r(e_i) \cdot s) > 0$. Since s is common to every term on this summation, we can rewrite it as $(\sum_{i=0}^{k-1} d_r(e_i)) \cdot s > 0$ or $d_r(l) \cdot s > 0$. From definition 3.2, $d_r(l) = d(l)$ thus $d(l) \cdot s > 0$ \square

4 Our Algorithm

In this section, we show a way to translate the formulation presented on theorem 3.6 to an ILP program. The first problem to be solved is how to formulate the existent *or condition* in part (a) of that theorem. We introduce some new concepts that will give us a heuristic method to deal with this situation.

We know that for a MDFG $G = (V, E, d, t)$, $e \in E$, and a feasible linear schedule s , $d_r(e) \cdot s \geq 0$. To satisfy the constraint $d_r(e) \cdot s > 0$ or $d_r(e) = (0, 0)$, presented in theorem 3.6, the condition $d_r(e) \cdot s = 0$, with $d_r(e) \neq (0, 0)$, needs to be avoided. From the above constraints, we need to exclude the possibility of any vector perpendicular to the schedule s . In practice, we need two extra vectors to enforce such exclusion, defining a region that contains the vectors to be excluded. This region is defined through the sides of an acute angle designated α , as shown in figure 6, where the point P suggests a vector contained in such region and that would be excluded from the possible solution with those vectors perpendicular to s .

Definition 4.1 *A rotational exclusion angle α_s for a MDFG $G = (V, E, d, t)$, $e \in E$, is the angle by applying a rotation to a schedule s , creating a new vector s' , such that*

dependences in the region bounded by the sides of α_s are compatible with s , i.e. $d(e) \cdot s \geq 0$, but are not compatible with s' , i.e., $d(e) \cdot s' < 0$. We designate s_c as the vector resulting of s being rotated clockwise and s_{cc} for a counterclockwise rotation.

Since the rotational exclusion angle does not restrict the exclusions to vectors perpendicular to the schedule vector, we impose that only those delay vectors in α , not perpendicular to the schedule vector, that have at least one of its components larger than a minimum value can be excluded, which implies reducing the angle α in such a way to prevent it from containing delay vectors with components smaller than the imposed limit. Using the example in figure 6, we want to choose α_s such that $P.x$ or $P.y$, or both, is greater than the imposed limit.

Definition 4.2 A *b-rotational exclusion angle* $\alpha_{s,b}$ is the rotational exclusion angle that excludes dependence vectors with at least one of its components larger than b .

The existence of a b-rotational exclusion angle is shown in the next lemma.

Lemma 4.1 Let $G = (V, E, d, t)$ be a 2DFG, $s = (s.x, s.y)$ a feasible schedule vector for G , b an integer constant, and $\alpha_s \in \{\alpha_s^c, \alpha_s^{cc}\}$ a rotational exclusion schedule angle, such that s_c is the result of a clockwise rotation of s by the angle α_s^c and s_{cc} is the counterclockwise rotation of s by the angle α_s^{cc} . The dependences excluded by α_s have magnitude larger than b , if α_s is chosen by

$$s_c = \begin{cases} (b * s.x, b * s.y - \frac{s.x}{|s.x|}) & s.x \neq 0 \\ \frac{s.y}{|s.y|} * (1, b * s.y) & s.x = 0 \end{cases}$$

and

$$s_{cc} = \begin{cases} (b * s.x - \frac{s.y}{|s.y|}, b * s.y) & s.y \neq 0 \\ \frac{s.x}{|s.x|} * (b * s.x, 1) & s.y = 0 \end{cases}$$

Proof: For $s.x = 0$ or $s.y = 0$ the proofs are immediate.

For $s.x \neq 0$ and $s.y \neq 0$, let's assume s in the first quadrant, i.e., $s.x > 0$ and $s.y > 0$: From definition 4.1, we delimit the exclusion region by $d(e) \cdot s \geq 0$ and $d(e) \cdot s_c < 0$, for $e \in E$. For $d(e) = (d.x, d.y)$, we want to exclude only vectors that have $d.x \geq b$ or $d.y \geq b$. These boundaries are shown graphically in figure 6. Note that dependence vectors are integral vectors, so we prove by contradiction that there is no integral dependence vector inside the exclusion region, except those perpendicular to s . Let's assume that $P = (P.x, P.y)$ is inside the exclusion region and P is not perpendicular to s . Then, $P \cdot s > 0$ and $P \cdot s_c < 0$, with $P.x < b$ and $P.y < b$

We know that the vectors representing the lines perpendicular to s and s_c have coordinates $(-s.y, s.x)$ and $(-b * s.y + 1, b * s.x)$ respectively.

Let's assume that $P.y$ and $P.x$ are integer values, and P is inside the exclusion region. Then $\frac{-s.y}{s.x} * P.y < P.x < \frac{(-b * s.y + 1)}{b * s.x} * P.y$

since $s.y * P.y$ is an integer, we substitute this product by N , N integer, (for clarity), then $-\frac{1}{s.x} * N < P.x < -\frac{1}{s.x} * N + \frac{1}{b*s.x} * P.y$
since $P.y < b$, then $P.x$ is not an integer value, which is a contradiction.
Similarly, we prove for s_{cc} and for the remaining quadrants. \square

With these new definitions, we can derive an ILP solution reformulating theorem 3.6.

Theorem 4.2 *Let $G = (V, E, d, t)$ be a 2DFG, r a two-dimensional retiming on G , s a feasible schedule vector for the retimed graph G_r , $\alpha_{s,b}$ a b -rotational exclusion angle, and c and b positive integers. Then r is a legal schedule-based retiming on G , such that $C(G_r) \leq c$, excluding dependences in $\alpha_{s,b}$ that have at least one vector component larger than b if and only if*

$$r(v) \cdot s_c - r(u) \cdot s_c < d(e) \cdot s_c \text{ and } r(v) \cdot s_{cc} - r(u) \cdot s_{cc} < d(e) \cdot s_{cc}, \text{ for every edge } u \xrightarrow{e} v$$

and

$$\text{if } T(u, v) > c \text{ then } r(v) \cdot s - r(u) \cdot s < D(u, v) \cdot s - 1, \text{ for every path } u \xrightarrow{P} v \text{ in } G$$

Proof: For part (a), it is clear, using lemma 4.1 Part (b) is proven on theorem 3.6. \square

A heuristic method has been developed considering the constraints presented on theorem 4.2. Other constraints can be added to the choice of the final schedule vector, such as computing the minimal execution time in a massive parallel processing architecture, i.e., for a DG $G_{dg} = (I^2, D)$, and $m = \max\{i \in I^2\}$, then $s \cdot m$ is minimum, or considering an optimal pipeline rate when mapping the DG to an array processor [10].

With the schedule vector chosen, a shortest path algorithm, modified from Floyd algorithm [16], was used to compute the functions $D^{(s)}$ and $T^{(s)}$. Finally, an ILP program written according to the constraints on theorem 4.2 has been used to compute the retiming function.

5 Experiments

We submitted our algorithm to some practical cases that we report on this section. The first example, already presented in figure 1 is a subset of the graph representing a Wave Digital Filter, designed according to the method developed by Fettweis to solve Partial Differential Equations [8], in our case applied to the solution of a Transmission Line problem.

The procedure of finding the retiming function shown in figure 7 consisted of the following steps:

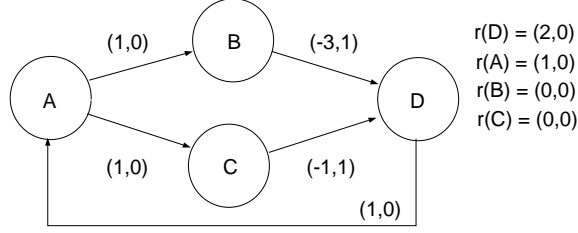


Figure 7: MDFG after retimed by r

1. Choose a range of possible values for the schedule vector components, and a value for the lower limit b in the b -rotational exclusion angle.
2. From the range defined in step 1, select the schedule vector within the constraints imposed in lemma 3.6. For our example:

$$\begin{aligned} s.x + s.y &> 0 \\ -s.x + s.y &> 0 \end{aligned}$$

3. Compute functions $D^{(s)}$ and $T^{(s)}$ using the modified Floyd algorithm.
4. Calculate s_c and s_{cc} , to determine the b -rotational exclusion angle.
5. Run the ILP solver, considering the b -rotational exclusion angle and a target cycle period = 1. The set of constraints were automatically generated according to theorem 4.2. This set included constraints:

- for each path with delay $D^{(s)}(u, v) = (0, 0)$ and $T^{(s)}(u, v) > 1$. For a target cycle period equals 1, these inequalities are more restrictive than applying the vectors s_c and s_{cc} to each edge, so the part (a) of theorem 4.2 was not used;
- for each path with delay $D^{(s)}(u, v) = (1, 1)$ and $T^{(s)}(u, v) > 1$;
- for each path with delay $D^{(s)}(u, v) = (-1, 1)$ and $T^{(s)}(u, v) > 1$. Note that the inequality for path $A \rightarrow D$ is repeated for $D^{(s)}(u, v) = (1, 1)$, which means that depending on the value of the schedule vector, either one or the other will be used but not both.

6. If no solution, go back to step 2, and select a new schedule.

The retiming function was found for a schedule vector $s = (1, 4)$, achieving the target cycle period, which represents a reduction to approximately one third of the initial execution time.

Another example, is a multidimensional filter that has the transfer function $H(z_1, z_2) = \frac{1}{1 - a*z_1^{-1} - b*z_2^{-2}}$, which is equivalent to the MDFG shown in figure 8(a). This MDFG has been submitted to the algorithm and a cycle time of 2 units has been reached with the function shown in figure 8(b) for a schedule vector $s = (2, 1)$.

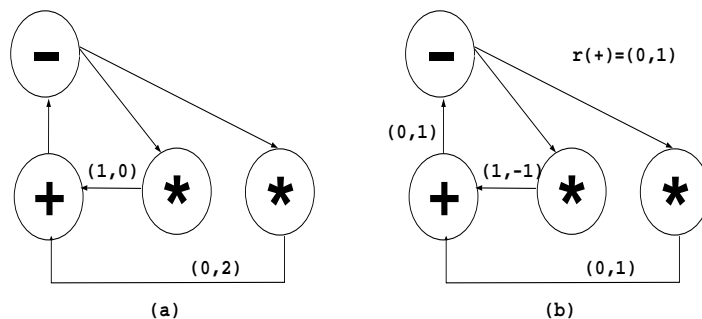


Figure 8: (a) original MDFG (b) retimed MDFG, cycle time = 2

6 Conclusion

We have introduced a novel technique on transforming a multi-dimensional data flow graph to obtain a high level of parallelism. This new technique, named *schedule-based multi-dimensional retiming* considers a feasible linear schedule as part of the process of redistributing operations along iterations, while keeping the original data dependences. The method consists of applying some constraints in the process of identification of the appropriate schedule vector for the expected retimed graph, and formulating a set of inequalities to be used in an ILP solver to get the final results.

Definitions and theorems supporting such technique have been presented and a heuristic method has been developed to compute the retiming function. Practical experiments, in the DSP area, have shown satisfactory results and two examples have been included in this paper. The generalization of such a technique to different application fields seems to be promising, considering the success of the one-dimensional retiming.

References

- [1] A. Aiken, “ Compaction Based Parallelization”.(Ph.D. thesis), Technical Report 88-922, Cornell University, 1988.
- [2] A. Aiken and A. Nicolau, “ Loop Quantization: An Analysis and Algorithm,” Technical Report 87-821, Department of Computer Science, Cornell University, March 1987.
- [3] J. Bu, E. F. Deprettere and P. Dewilde, “ A Design Methodology for Fixed Size Systolic Arrays”. *Proc. International Conference on Application Specific Array Processors*, Princeton, New Jersey, 1990, pp. III 591-602.

- [4] L.-F. Chao and E. H.-M. Sha, "Static Scheduling of Nested Loops," *Proceedings of 7th International Parallel Processing Symposium*, Newport Beach, CA, April, 1993, pp. 1421-1424.
- [5] 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.
- [6] L.-F. Chao and E. H.-M. Sha, "Unified Static Scheduling on Various Models". *Proc. 1993 International Conference on Parallel Processing*, pp. 80-84, August 1993.
- [7] R. Cytron, "Doacross: Beyond Vectorization for Multiprocessors". *Proc. International Conference on Parallel Processing*, 1986, pp. 836-844.
- [8] 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.
- [9] J. A. Fisher and B. R. Rau, "Instruction-Level Parallel Processing". *Science*, Vol. 253, September 1991, pp. 1233-1241.
- [10] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice Hall, 1988.
- [11] M. Lam, "Software Pipelining: An Effective Scheduling for VLIW Machines". *ACM SIGPLAN Conference on Prog. Lang. Design and Implementation*, 1988, pp. 318-328.
- [12] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry". *Algorithmica*, 6, 1991, pp. 5-35.
- [13] 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.
- [14] 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.
- [15] D. A. Schwartz, "Cyclo-Static Realizations, Loop Unrolling and CPM: Optimal Multiprocessor Scheduling." *Technical report, Georgia Institute of Technology - School of Electrical Engineering*, 1987.
- [16] R. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, Pennsylvania, 1983.
- [17] M. Wolfe, "Loop Skewing: the Wavefront Method Revisited,". *International Journal of Parallel Programming*, Vol. 15, No. 4, August 1986, pp. 284-294.