HAL
open science

Execution Time Optimization Using Delayed Multidimensional Retiming

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui

To cite this version:

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui. Execution Time Optimization Using Delayed Multidimensional Retiming. IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, Oct 2012, Dublin, Ireland. hal-01800762

HAL Id: hal-01800762

https://hal.science/hal-01800762v1

Submitted on 28 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Execution Time Optimization Using Delayed Multidimensional Retiming

Yaroub Elloumi[1,2], Mohamed Akil
[1]Université Paris-Est, ESIEE Paris
Laboratoire d'Informatique Gaspard Monge, Equipe A3SI
93162 Noisy-le-Grand, France
Emails: {elloumya, akilm}@esiee.fr

Mohamed Hedi Bedoui
[2]University of Monastir, Faculty of Medicine of Monastir
Laboratory of Biophysics, TIM Team
5019, Monastir, Tunisia
Email: medhedi.bedoui@fmm.rnu.tn

***Abstract*— Multidimensional retiming is an efficient optimization approach that ensures increasing a parallelism level in order to optimize the execution time. Two existing techniques called incremental and chained multidimensional retiming are based on this approach, which aim at achieving a full parallelism on loop body in order to schedule applications with a minimum cycle period. However, the cycle number increases in terms of parallelism level which presents a limiting factor to respect the execution time constraint of real-time applications.**

In this paper, we show how the minimal cycle period is achieved in multidimensional applications without applying a full parallelism. We present the theory of a novel technique, called delayed multidimensional retiming. Firstly, two efficient algorithms are presented where the first one insures the extraction of timing and data dependency properties of the application and the second one selects the set of data path for retiming. Then, we propose theorems to deduce a retiming function for the selected paths. Finally, a third algorithm describing the optimization approach is introduced. The experimental results show that our technique improves execution times in comparison to existing techniques. It achieves average improvements on the execution time of 41.57% compared to the Incremental technique and 11.55% compared to the Chained technique.

Keywords: Parallelism, modeling, nested loop, Optimization.

I. INTRODUCTION

Real-time systems are characterized by increasing computer performances. However, many software real-time applications are based on iterative and recursive structures, such as the ones used on high-definition vision, remote sensing and medical imaging, for instance. This loop body generally presents the most critical section in terms of execution time. It fills an important section of the execution time of the whole application.

Thus, several optimization approaches are proposed which aim at increasing parallelism in the repetitive patterns, in order to optimize timing constraints. They always proceed to model applications by data flow graphs and apply a graph transformation in order to enhance performances. Many software pipelining techniques [6,7,8] are proposed to optimize loop bodies. They allow exploring the instruction level parallelism of one loop. When they are applied to optimize nested loops, their performance improvement is very limited.

The Multi-Dimensional (MD) retiming [1,3] is an effective approach for optimizing in MD applications. It ensures increasing parallelism level in order to minimize the cycle period, and hence enhancing the computing performance. The software and hardware constraints for applying the MD retiming are presented in [5], which consists in a limiting parallelism level by the iteration numbers of nested loops. Two techniques based on this approach are proposed [1,3] which are called "incremental" and "chained" MD retiming. They aim at scheduling MD applications with the minimal cycle period. Their processes consist of successively increasing the parallelism level of the application, until executing all computations in full parallel. However, achieving a full parallelism requires adding instructions outside the loop structure, leading to increasing the cycle number. The more the parallelism increases, the more the disadvantages are aggravated. Therefore, solutions provided with such timing characteristics are not adequate to achieve constraints of real-time systems. Another MD retiming technique is described in [4,10], which proposes to retime the whole data paths. It applies the process of the retiming technique of synchronous applications [9] to the MD ones. However, this process is based on the non-negativity of data dependency which is not the case of MD applications. Furthermore, operations cannot always be redistributed due to the overlapping of data dependencies between iterations.

In this paper, we show how to schedule the MD application with the minimal cycle period without achieving full parallelism. We propose a new technique of MD retiming, called "delayed multidimensional retiming". It ensures exploring the characteristics of data dependencies and computation times on MD applications, in order to retime data paths. Thus, the parallelism level is reduced while scheduling applications with the minimal cycle period. It provides solutions with enhanced execution times compared to those provided by the "incremental" and "chained" techniques.

The rest of the paper is organized as follows. In section 2, we present the basic concepts of modeling and retiming MD applications. In section 3, we present the theory of the delayed multidimensional retiming technique by describing its principles and the corresponding algorithms. Experimental results are presented in section 4, followed by concluding remarks in section 5.

II. BACKGROUND

In this section, we introduce some basic concepts which will be used in later sections. We start by introducing how to model nested loops with the Multidimensional Data Flow Graph. The multidimensional retiming approach is described in the second paragraph. The third one discusses the evolution of execution time in terms of existing MD retiming techniques.

A. Multidimensional Data Flow Graph

The Multidimensional Data Flow Graph (MDFG) is an extension of the classic data flow graph that allows representing nested iterative and recursive structures. It is modeled by a node-weighted and edge-weighted directed graph in a way that $G = (V, E, d, t)$, where V is the set of computation nodes, $E \subseteq V \times V$ is the set of edges, and $d(e_i)$ is a function from E to Z^n representing the multidimensional delay between two nodes, where n is the number of dimensions, also $t(v_j)$ is a function from V to positive integers, representing the computation time of the node v_j. Once running all nodes in the MDFG is similar to one iteration execution. An edge delay in the MDFG with n dimensions is presented as $d(e) = (c_1, c_2, \dots, c_n)$. For $e: v_i \rightarrow v_j$, the index c_k presents the difference between the iteration executing v_j and the iteration executing v_i of the loop k. As example, the wave digital filter is modeled as two-dimensional Data Flow Graph (2DFG) in Fig. 1(a), which its code is composed by two nested loops as shown in Fig. 1(b). Each edge in 2DFG is assigned by a delay $d(e) = (d.x, d.y)$, whose terms "$d.x$" and "$d.y$" are respectively in relation with the outermost loop and the innermost loop. For an edge $e: v_i \rightarrow v_j$, the delay $d(e) = (0, x)$ means that v_i and v_j are executed in the same iteration of the outermost loop. For the innermost loop, if the node v_i is executed in the iteration k, the node v_j is executed in the iteration $(k + x)$. An edge with a zero delay $d(e) = (0,0)$ means that both nodes are executed in the same iteration, such as the edges $D \rightarrow A$, $A \rightarrow B$ and $A \rightarrow C$ shown in Fig. 1(a).

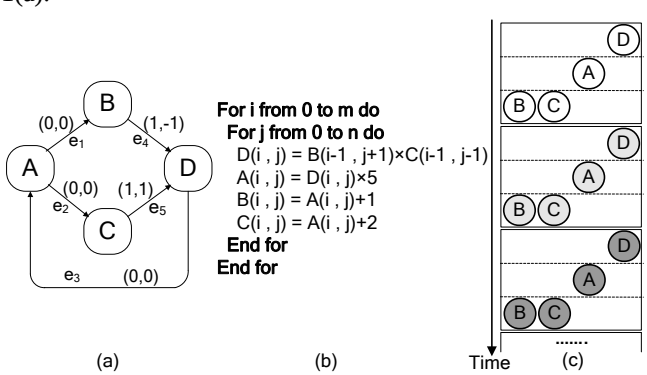

Figure 1. (a) The wave digital filter MDFG; (b) The code of the nested loops; (c) The static schedule

We use the notation $p: v_i \xrightarrow{e_m} \dots \xrightarrow{e_n} v_j$ to mean that p is a path from v_i to v_j. The delay vector and the total computation time of a path p are respectively equal to $d(p) = \sum_{k=m}^{k=n} d(e_k)$ and $t(p) = \sum_{k=i}^{k=j} t(v_k)$. The period during which all computation nodes in 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 is the maximal computation time among paths that have a zero delay. For example, assuming that each node is executed in one time unit $t(A) = t(B) = t(C) = t(D) = 1$, the cycle period of the wave digital filter is $C(G) = 3$, as shown in the static schedule in Fig. 1(c), whose nodes belonging to the same iteration are modeled by a same pattern. It can be measured through the paths $p: D \rightarrow A \rightarrow B$ or $p: D \rightarrow A \rightarrow C$.

The execution pattern of a nested loop can be illustrated by an iteration space as shown in Fig. 2(a). It presents an integral point in a Cartesian space. Each dotted square in the iteration space is a copy of the MDFG which are identified by the loop control indexes. The cell assigned by $(0,0)$ is the first iteration to be executed. This graph is transformed into an acyclic graph, called Cell Dependency Graph (CDG), on which each cell presents a complete iteration and is bounded by the loop indexes. It allows showing the execution sequence of a nested loop, and illustrates clearly data dependencies between iterations on the MDFG, such as the CDG in Fig. 2(b) of the wave digital filter.

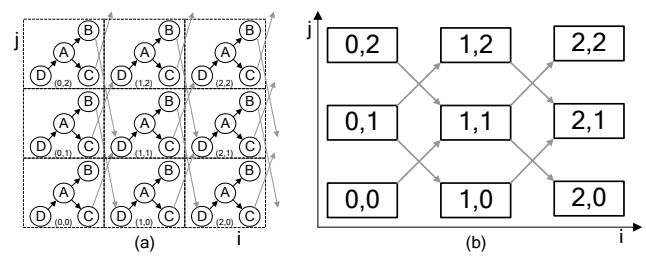

Figure 2. (a) The iteration space of the wave digital filter MDFG; (b) The cell dependency graph

This graph is used to identify an execution order of the whole application, called schedule vector. An MDFG $G = (V, E, d, t)$ is realizable if a schedule vector s for the CDG in respect to G exists. It means that $s \times d(e) \geq 0$ for each $e \in E$, and no cycle exists in its corresponding CDG. The CDG, shown in Fig. 3(b), can be executed by a row-wise execution sequence, i.e., the schedule vector $s = (1,0)$.

B. Multidimensional Retiming

The multidimensional retiming approach increases the loop parallelism in order to reduce critical path, while preserving data dependencies of the original MDFG. The retiming vector $r(u) = (r_1, \dots, r_n)$, where $u \in V$ and n is the loop dimension, presents the offset between the original iterations containing u and the ones after retiming: for each r_k index such as $1 \leq k \leq n$, the execution of the node u in the iteration i is moved to the iteration $i - r_k$. Fig. 3(a) shows the wave digital filter after applying $r(D) = (0,1)$. Thus, each ith copy of D is shifted up and executed in the previous iteration of the innermost loop. Some nodes are shifted upstream the retimed loop which are called prologue, such as

instructions D executed outside the innermost loop on the code in Fig. 3(b). Correspondingly, the complementary nodes are executed after the loop body to complete the process, which is called epilogue.

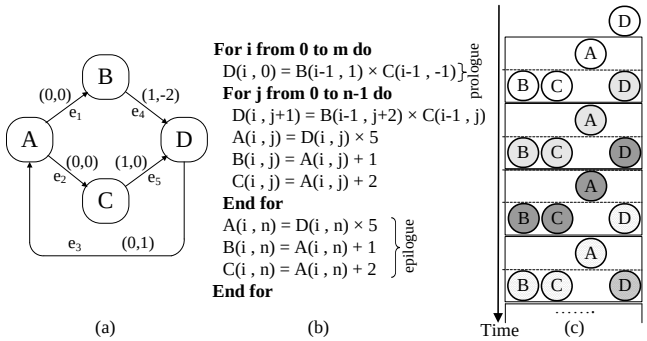

Figure 3. (a) The retimed Wave digital filter MDFG with r(D)=(0,1); (b) The code of the retimed MDFG; (c) The static schedule

We notice that instruction D inside the innermost loop of the code in Fig. 3(b) has not any data dependency with other instructions executed in the same iteration. It allows executing instructions in parallel, such as executing node D in parallel to node A, as shown in the static schedule of Fig. 3(c). Thus, the cycle period is reduced from three to two time units, assuming that all execution time nodes are equal to one. However, the code size is increased due to prologue and epilogue instructions where each one corresponds to an additional cycle period. This transformation is the principle of the MD retiming which attempts to parallelize execution nodes in order to reduce the cycle period, despite of increasing the code size.

A retiming function is called legal if it provides a realizable MDFG. Its value is deduced from schedule vectors of the original graph. It consists in identifying a strictly positive scheduling sub-space S^+ that contains all $s \in S$ vectors such that $d(e) \times s > 0$ for every $d(e) \neq (0, \dots, 0)$. A legal MD retiming r of node u is any vector orthogonal to s, whose u has all the oncoming edges having non-zero delays. In the case of the wave digital filter, the original MDFG can be scheduled following the vector $s = (1,0)$, which verifies that $r(d) = (0,1)$ provides a realizable MDFG: the CDG in Fig. 4(b) can be executed following the scheduling vector $s = (3,1)$.

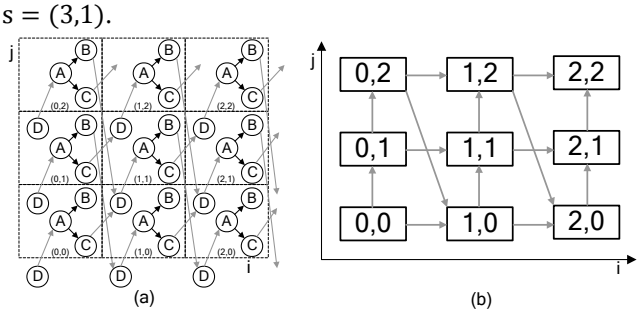

Figure 4. (a) The iteration space of the retimed MDFG with r(D)=(0,1); (b) The cell dependency graph

All MD retiming techniques proceed to apply successively such transformation until achieving a fully parallelized MDFG. All edges on the final MDFG should have a non-zero delay. For any $p: v_i \to \cdots \to v_j$ path of the MDFG, MD retiming techniques proceed to execute each v_k node in a cycle period separately, where $i \leq k \leq j$. We show in Fig. 5(a) the full parallel wave digital filter MDFG, after applying $r(D) = (0,2)$ and $r(A) = (0,1)$. The static schedule in Fig. 5(c) shows that the full parallelism is achieved, whose nodes belonging to the same iteration in the original loop are distributed into three different ones. Accordingly, each iteration requires one time unit to be executed. However, the graph transformation implies adding three instructions as prologue, and five instructions as epilogue in the corresponding code, in comparison to the original one.

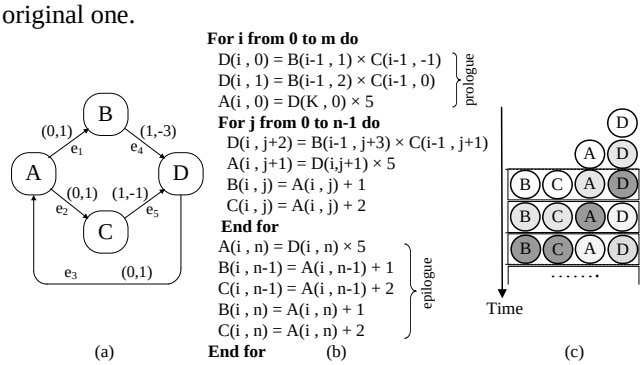

Figure 5. (a) The Full parallel MDFG; (b) The code of the full parallel MDFG; (c) The static schedule

C. Limitations of Existing Techniques

MD retiming affects codes of original loop bodies. Firstly, the code size has increased due to prologue and epilogue sections: For an MD retiming function $r(u) = (di, \dots, dj)$, each index $dj (\neq 0)$ implies executing the $|dj|$ occurrences of u before the loop j and the $|dj|$ occurrences of complementary nodes after the same loop. Secondly, the retiming transformation implies redefining the loop bounds and loop indexes. Hence, computation instructions are added on the algorithm, before each affected loop. Those added code sections are executed in upstream loop, which leads to increasing cycle number. Furthermore, they are not executed in parallel, which requires an important number of cycle periods compared to that required to execute the whole loop body. Consequently, the cycle number increases significantly in terms of MD retiming.

These characteristics depend on the value of retiming function. For $r(u) = (di, \dots, dj)$, the more the index grows $|dj|$, the more the size of prologue and epilogue is great. To reduce disadvantages, incremental and chained MD techniques select a scheduling vector $s = (S_x, S_y)$ where $S_x + S_y$ is minimal and aims at deducing the retiming function that provides an optimal MDFG. In addition, the cycle number increases in the same way as the number of MD retiming. All existing techniques require the same number of the MD retiming function in order to achieve full parallelism: If the critical zero-delay path is composed by X nodes then any existing technique employs $(X - 1)$ MD retiming. They are applied incrementally from the first node

having zero delay outcoming edges to the last one on the critical path. Code sections are added after each retiming function. They correspond not only to the loops structures, but also to the existing prologues and epilogues. Thus, cycle numbers increase exponentially and the disadvantages are heavily aggravated, in terms of the retiming number.

We proceed to identify the evolution of the cycle number in terms of retiming functions in the case of the infinite impulse response filter schematized in Fig. 6. Based on its data dependencies, this MD application requires four retiming functions to obtain a fully parallelized MDFG. We use the chained technique with an optimal Retiming function $r = (1, -1)$. After each MD retiming, we provide the retimed MDFG and we deduce its codes in order to compute the cycle number. These steps are repeated until providing the full parallel MDFG which cycle number values are illustrated in Fig. 7.

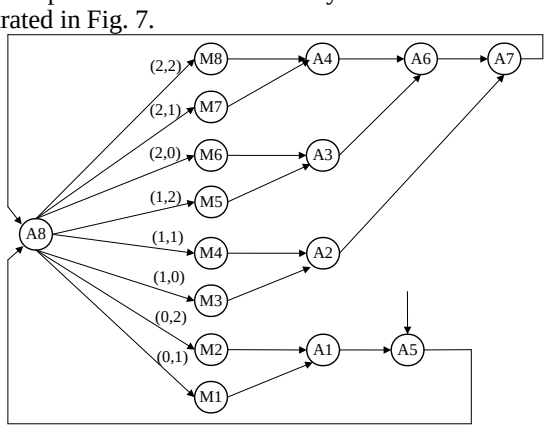

Figure 6. The MDFG of the infinite impulse response filter

Based on cycle number values, we conclude that the more we apply a MD retiming functions, the more the cycle number increases: It develops by 58% on the fully parallel solution, compared to the original MDFG. Thus, we conclude that final solutions, provided by the existing MD retiming techniques, do not allow respecting strict timing constraints. Therefore, although full parallel applications are scheduled with minimum cycle period, they are not suitable to be implemented in real-time systems.

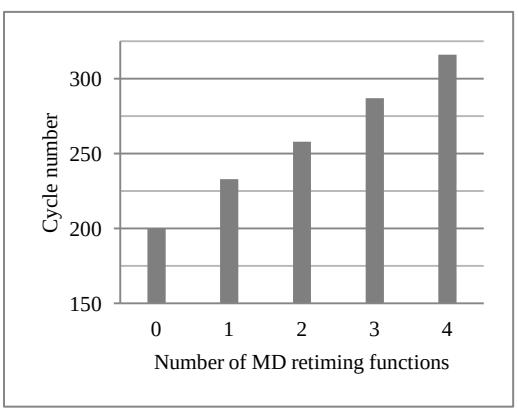

Figure 7. Evolution of cycle number in term of Retiming function

III. Theory of Delayed Multidimensional Retiming

A. Principles

The minimal cycle period in a data flow graph is defined as the computation time of the longest zero-delay path $(\max\{t(p), d(p) = 0\})$ [4]. For general executing-time cases, a cycle period of a full parallelized MDFG is equal to the maximal execution time among the computation nodes $(\max\{t(v), v\epsilon V\})$. Taken the example of the wave digital filter, the MDFG contains two node types which are the multiplier and adder. Assuming that the first type requires one time unit while the second one requires two, the full parallel MDFG in Fig. 5(a) is scheduled with $c_{min} = 2$. Any existing technique requires two MD retiming to achieve a full parallelism, whose static schedule is showed in Fig. 8. Nodes belonging to the same iteration in the original MDFG, which are illustrated with the same color, are divided into three different iterations.

The full parallelized graph in Fig. 5(a) can be executed following the schedule vector $s = (1,0)$ that has two orthogonal vectors $(0,1)$ or $(0, -1)$. While both edges e_4 and e_5 are non-zero delay, we proceed to retime node D by $r(D) = (0, -1)$, which results in the MDFG in Fig. 9(a). It is easy to verify that the provided MDFG is not fully parallelized. However, it keeps scheduling the application with the minimal cycle period $c_{min} = 2$. We deduce from the static schedule in Fig. 9(c) that cycle periods are optimally used by reducing the lost time between providing and consuming data. Nevertheless, the code contains only two instructions in both prologue and epilogue, as shown in Fig. 9(b). Therefore, the cycle number is reduced by 7.69% compared to the full parallel graph in Fig. 5(a), and so is the execution time. We deduce that an optimal execution time can be realized without achieving a full parallelism.

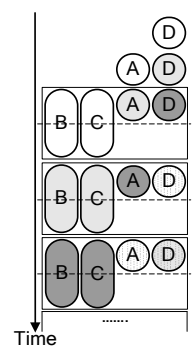

Figure 8. The static schedule of the full parallel wave digital filter MDFG

Based on the new delays, the MDFG keeps the same data cycle delays from the original one. The $d_r(p) = d(p) + r(v) - r(u)$ equation [1,3] is verified for all paths in the MDFG. Moreover, the CDG in Fig. 10(b) does not contain any cycle and hence the existence of infinite schedule vectors that can ensure a legal execution order, so we cite as an example $s = (3,1)$. As a consequence, we conclude that this transformation provides a realizable MDFG and preserves the functionality of the whole application.

Let us focus on the transformation applied to provide the MDFG showed in Fig. 9(a). We note that nodes belonging to the same iteration in the original MDFG are shared on two cycles, as illustrated in Fig. 9(c). This node distribution is similar to applying one MD retiming function. Furthermore, the e_3 edge have a zero delay, which means that the e_3 delay is kept through retiming with respect to the original MDFG, and so D and A are still executed in the same iteration. Besides, the iteration space in Fig. 10(a) shows that each whole $p: D \rightarrow A$ path, originally executed in the iteration (i), becomes executed in the iteration $(i-1)$. Moreover, each path p belonging to the first iteration on the innermost loop is pushed outside cellules of the iteration space.

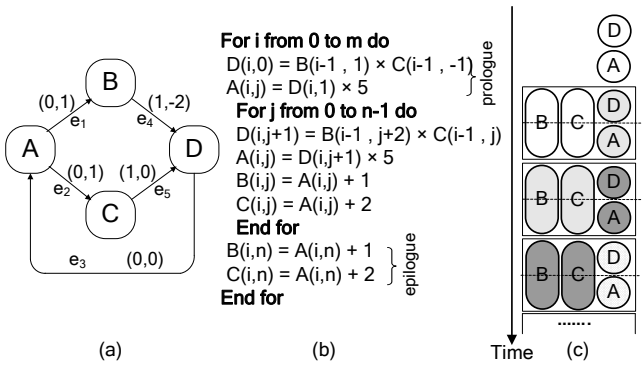

Figure 9. (a) The wave digital filter MDFG provided by the delayed MD retiming; (b) The code of the retimed MDFG; (c) The static schedule

Based on those characteristics, this modification can be considered as applying the MD retiming (0,1) to the path $p: D \rightarrow A$ on the original MDFG in Fig. 1(a). It consists of subtracting delays from e_4 and e_5, which are the incoming edges of the first node D, and adding them to e_1 and e_2, which are the outcoming edges of the last node A. The retimed path is executed in a two time units. While the path is a zero delay, hence its execution in the same iteration, the MDFG is still scheduled with the minimal cycle period. This result is achieved by applying just one MD retiming whose overheard is added once. It provides an enhanced execution time compared to the existing techniques.

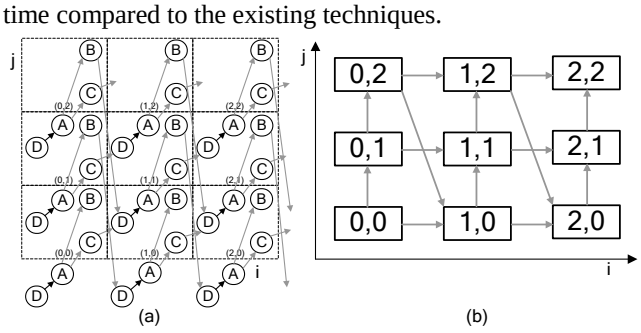

Figure 10. (a) The iteration space of MDFG in Fig. 9.a ; (b) The cell dependency graph

We propose a new approach of MD retiming that aims to schedule the MDFG with a minimal cycle period, without achieving a full parallelism. It proceeds to identify zero-delay paths where execution times are below or equal to c_{min}. Thereafter, it selects MD retiming functions that will be applied to the extracted paths. Contrary to the existing techniques, retiming functions are not applied to successive nodes but distant nodes. Thus, we call our technique Delayed MD retiming.

B. Zero-Delay Path Identification

In this section, we describe our process to identify the zero delay paths that should be retimed. First, we introduce a model to extract data dependency and timing proprieties of all paths in the MDFG. Second, we show how to explore those proprieties to identify the set of paths for retiming. In this paper, our technique is presented with two dimensional notations. It can be easily extended to multi-dimensions.

1) Data dependency and timing proprieties of MDFG

Our approach aims at scheduling the MDFG with minimal cycle period. It means that each zero delay path p should be executed in c_{min}. Thus, we proceed to sweeping the MDFG in order to identify the delay and execution time of each path. As known, any two nodes u and v can be connected by several paths. If only one path is zero delay then it is susceptible to be retimed. We define $D(u,v)$ as the minimum delay between the paths connecting u and v, such as $D(u,v) = \min\{d(p), p: u \rightarrow v\}$. If only one path among those connecting u and v has a zero-delay then $D(u,v) = (0,\dots,0)$; else, $D(u,v) \neq (0,\dots,0)$. In addition, two nodes can be connected with several zero-delay paths whose execution time should be taken into account. Based on our approach, all paths should be executed in the bound of the cycle period. Thus, we define $T(u,v)$ as the maximum execution time among the zero delay path connecting u and v, such as $T(u,v) = \max\{t(p) \text{ where } p: u \rightarrow v \text{ and } d(p) = D(u,v)\}$. Our approach aims at providing MDFGs whose paths should have a non-zero delay if their execution time exceeds c_{min}. This principle is described in theorem 1.

Theorem 1. *Let G =(V,E,d,t) a realizable MDFG and c a cycle period. The following are equivalent:*

- $\varphi(G) \leq c$
- *For all nodes* $u, v \in V$, *if* $T(u,v) > c$ *then* $D(u,v) \neq 0$.

Proof. *We suppose that* $\varphi(G) \leq c$, *and* $u, v \in V$ *such as* $T(u,v) > c$. *If* $D(u,v) = 0$, *then it exists a p path from u to v having an execution time* $t(p) = T(u,v)$ *that exceeds c, and a delay* $d(p) = D(u,v) = 0$, *which are in contradiction with the first supposition. We suppose now that the second condition is verified, and let* $u \rightarrow v$ *be any p path with zero delay in G, then we have* $D(u,v) = d(p) = 0$ *which implies that* $t(p) \leq T(u,v) \leq c$.

Based on the previous theorem, we proceed to compute D and T of each pair of nodes, and apply MD retiming in order to respect theorem 1. The results are ranged in two matrices called D and T, whose computing process is indicated in algorithm 1.

As an example, we run algorithm 1 in the case of the wave digital filter whose MDFG is showed in Fig. 1(a). It provides values of the matrices D and T as illustrated in Table 1. To distinguish the zero-delay paths, we proceed to model their corresponding cells in matrices D and T by a gray color.

Algorithm 1. Compute D and T

Input : a realizable MDFG $G = (V, E, d, t)$
Output : matrices D and T

Begin
For each edge in G
  Add element $(u \xrightarrow{e} v, d(e), t(u) + t(v))$ to ARC list
End for
$TEST \leftarrow ARC$
For each element $(u \xrightarrow{e} v, x, y)$ of TEST
  $D(u, v) \leftarrow x$
  $T(u, v) \leftarrow y$
End For
Repeat
  For each element $(u_i \overset{p}{\Rightarrow} v_i, x_i, y_i)$ of TEST
    For each element $(u_j \xrightarrow{e} v_j, x_j, y_j)$ of ARC
      If $v_i = u_j$ & $(u_i \overset{p}{\Rightarrow} v_j, d, t) \notin R$ **then**
        Add $\left(u_i \overset{p}{\Rightarrow} v_j, x_i + x_j, y_i + y_j\right)$ to R
      Else if $v_i = u_j$ & $(u_i \overset{p}{\Rightarrow} v_j, d, t) \in R$ **then**
        Add $\left(u_i \rightarrow v_j, \min\left(d, x_i + x_j\right), \max\left(t, y_i + y_j\right)\right)$ to R
      End if
    End For
  End For
  For each element $(u \overset{p}{\Rightarrow} v, x, y)$ of R
    $D(u, v) \leftarrow x$
    $T(u, v) \leftarrow y$
  End For
  $TEST \leftarrow R$
Until D and T are totally filled
End

TABLE I. MATRICES D AND T OF THE ORIGINAL WAVE DIGITAL FILTER

$D(u, v)$

u\v	A	B	C	D
A	(0,0)	(0,0)	(0,0)	(1,1)
B	(1,1)	(0,0)	(1,1)	(1,1)
C	(1,1)	(1,1)	(0,0)	(1,1)
D	(0,0)	(0,0)	(0,0)	(0,0)

$T(u, v)$

u\v	A	B	C	D
A	1	3	3	3
B	4	2	6	3
C	4	6	2	3
D	2	4	4	1

2) Path selection

We proceed now to identify the set of paths that should be retimed. Those paths are deduced directly from the matrices D and T, as described in algorithm 2. We start by selecting the nodes having all the incoming edges with a non-zero delay. For each node u, we extract from the corresponding line in the matrix D the cells having a zero delay edge. The paths $p: u \rightarrow v$ to retime are those having $D(n, x) = (0, \ldots, 0)$ and $T(n, x) \leq c_{min}$.

The extraction process may result in a redundancy in the paths. Moreover, an extracted path may present a sequence of another one, where our approach should select the one to retime. In fact, our approach aims at decreasing the parallelism level by employing the minimum number of retiming functions. It means that the path to be retimed must contains the maximum successive nodes, while respecting the cycle period constraint. This principle is followed in the last instruction of algorithm 2 when optimizing the set of paths.

We run algorithm 2 to the wave digital filter, whose node D is the only one having all incoming edges with non-zero delays. We show from matrix D that four paths starting from D are zero delay, which are $\{D, D \rightarrow A, D \rightarrow A \rightarrow B, D \rightarrow A \rightarrow C\}$. Only the two first paths are added to the list L, which are able to be retimed. While they are superimposed, the last instruction in algorithm 2 provides the path $D \rightarrow A$ to retime.

Algorithm 2. Path selection

Input : a realizable MDFG $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{d}, \boldsymbol{t})$, matrices D and T
Output : list of paths

Begin
Identify nodes n having all incoming edges with a non-zero delay and at least one outcoming edge with a zero delay
For each node n
  For each node x in GFDM
    If $\boldsymbol{D}(\boldsymbol{n}, \boldsymbol{x}) = (\boldsymbol{0}, \ldots, \boldsymbol{0})$ and $\boldsymbol{T}(\boldsymbol{n}, \boldsymbol{x}) \leq \boldsymbol{c_{min}}$ **then**
      Add path $\boldsymbol{p}: \boldsymbol{n} \rightarrow \boldsymbol{x}$ to $\boldsymbol{L}$
    End if
  End for
End for
Optimize $\boldsymbol{L}$
End

C. Multidimensional Retiming Selection

We describe in this section how to select a retiming function for a zero delay path. In fact, selecting a retiming function consists in choosing an orthogonal vector to s that verifies $s \times d(e) \geq 0$. The retiming function is deduced from the values of all non-zero delays in the MDFG, and hence their independence from the nodes that will be retimed. However, the selected retiming is only applied to nodes, which have all incoming edges with non-zero delays. We try to define from this selected function a legal retiming for a zero delay path. First of all, we describe in theorem 2 the way to deduce a legal MD retiming of a node that has an incoming zero-delay edge.

THEOREM 2. *Let $G = (V, E, d, t)$ be a realizable MDFG, and $u, v \in V$ where v has only one incoming edge $e: u \rightarrow v$ where d(e)=(0,...,0). If r a legal multidimensional retiming of u, then r is a legal multidimensional Retiming of v.*

Proof. *Let $p: \ldots \xrightarrow{(a,b)} u \xrightarrow{(0,0)} v \xrightarrow{(x,y)} \ldots$ be a path in G. $r(u) = (i, j)$ is a legal MD retiming of G, which means that there exists a scheduling vector s where $(i, j) \times s \geq 0$ and $(x, y) \times s \geq 0$. Adding those two inequalities shows that $(x + i, y + j) \times s \geq 0$. Knowing that $(a - i, b - j) \times s \geq 0$ and so $(0,0) \times s \geq 0$, the MDFG containing $p: \ldots \xrightarrow{(a-i,b-j)} u \xrightarrow{(0,0)} v \xrightarrow{(x+i,y+j)} \ldots$ is realizable, which can be executed following the scheduling vector s. Thus, r(v) is a legal MD retiming.*

This theorem proves that the legal MD retiming for a node can be applied to its successor, if this last has only one incoming edge. Moreover, it is easy to observe that the previous theorem can be used for successive nodes that respect the same condition. Therefore, theorem 2 allows identifying a legal retiming function for a zero delay path. We consider a node with an outcoming zero delay edge as the first node of a zero delay path and we proceed to test nodes following the data dependency direction. If the successor node tallies the theorem condition, we add it to the

path that will be retimed. This test can be repeated incrementally while the execution time path is less than c_{min}.

The growing size of MDFG is proportional to the set of the selected zero-delay paths, and hence to the number of required MD retiming transformation. However, defining a retiming function for each selected path cannot be an adequate solution. Thus, we aim at reducing it by identifying paths that can be retimed with the same function. In fact, the set of nodes that have all incoming edges with non-zero delays can be retimed using the same retiming function [2]. We introduce the method of predicting an MD retiming for several paths theorem 3.

THEOREM 3. *Let $G = (V, E, d, t)$ be a realizable MDFG, $X, Y \subseteq V$, $X \cap Y = \emptyset$ and $e: x \xrightarrow{(0,\dots,0)} y$ for all y incoming edges, where $y \in Y$ and $x \in X$. If $r(X)$ is a legal MD retiming, then $r(Y)$ is a legal MD retiming.*

Proof. *Proved immediately from theorem 2.*

This theorem gives us the alternative to retiming several paths, which start by the nodes having incoming edges with a non-zero delay, using just one MD retiming function.

D. The Delayed Multidimensional Retiming Algorithm

This step ensures selecting an MD retiming function and providing the MDFG scheduled in a minimal cycle period. Our technique starts by identifying the minimal cycle period and computing the matrices D and T as described in algorithm 1. Thereafter, it selects a legal MD retiming r of the MDFG as proceeded in the existing techniques, and it runs algorithm 2 to identify the set of paths L, in order to apply on them the retiming function r. These steps are repeated until all zero delay paths are executed in c_{min}, as described in algorithm 3.

Algorithm 3. Delayed multidimensional retiming

Input : a realizable MDFG $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{d}, \boldsymbol{t})$,
Output : a realizable MDFG $\boldsymbol{G_r} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{t}, \boldsymbol{d_r})$ with cycle period $\boldsymbol{\varphi(G_r) = c_{min}}$

Begin
Identify the minimal cycle period $\boldsymbol{c_{min}}$
Compute the matrices $\boldsymbol{D}$ and $\boldsymbol{T}$ (as described in algorithm 1)
While $\exists \mathbf{p}$ **such as** $\mathbf{d(p) = (0, \dots, 0)}$ **and** $\mathbf{t(p) > c_{min}}$ **Do**
  Find a scheduling vector $\mathbf{s = (s.x, s.y)}$ where $\mathbf{s.x + s.y}$ is minimal
  Select an MD retiming function $\boldsymbol{r}$ orthogonal to $\boldsymbol{s}$
  Identify the list of the path L (as described in algorithm 2)
  For each path $\boldsymbol{ch}$ of $\boldsymbol{L}$
    Apply $\boldsymbol{r(ch)}$
  End for
  Update matrices $\boldsymbol{D}$ et $\boldsymbol{T}$
End while
End

Using the delayed MD retiming, an MDFG scheduled with the minimal cycle period is always achievable, which efficiency is proved in theorem 4.

THEOREM 4. *Let $G = (V, E, d, t)$ be a realizable MDFG, the Delayed MD retiming algorithm transforms G to G_r, in at most $O(V^2 + E)$ times, in a way that G_r is scheduled with a minimal cycle period.*

Proof. *Algorithm 1 allows computing D and T for each pair of nodes in the MDFG, which requires $O(E + V^2)$ times. Thereafter, algorithm 2 tests at the most all cells of the matrices D and T, which can be performed in $(n \times V)$ instructions where n is an integer value $n > 0$. Finally, algorithm 3 selects one MD retiming function, which requires at most E instructions, and applies it for each selected paths. Thus, delayed MD retiming technique requires only $O(V^2 + E)$ times to be performed.*

We run algorithm 3 to the MDFG in Fig. 1(a) which c_{min} is equal to two time units, and the matrices D and T are showed in Table 2. On the first iteration of the algorithm 3, the identified scheduling vector s is equal to (1,0), and hence the retiming function $(0,1)$. Algorithm 2 provides the path $p: D \rightarrow A$ that is retimed in the following step. The new matrices D and T are showed in Table 2 that all cells respect theorem 1 constraints. As a result, the final MDFG is as illustrated in Fig. 9(a).

TABLE II. MATRICES D AND T OF THE WAVE DIGITAL FILTER PROVIDED BY THE DELAYED MD RETIMING

$D(u, v)$

u\v	A	B	C	D
A	(0,0)	(0,1)	(0,1)	(1,1)
B	(1,1)	(0,0)	(1,1)	(1,1)
C	(1,1)	(1,1)	(0,0)	(1,1)
D	(0,0)	(0,1)	(0,1)	(0,0)

$T(u, v)$

u\v	A	B	C	D
A	1	3	3	3
B	4	2	6	3
C	4	6	2	3
D	2	4	4	1

IV. EXPERIMENTAL RESULTS

In our experiments, we compare our technique to the "incremental" and "chained" MD retiming ones. We proceed to apply the three techniques to the same set of MDFGs, in order to deduce their execution times. We conduct experiments from two-dimensional loops which are the Infinite Impulse Response Filter (IIRF), the Wave Digital Filter (WDF) after applying the Fettweis transformation [2], and the Walsh-Fourier Transform (WFT) [8]. All MD graphs are composed of adder and multiplier nodes. Knowing that our technique applies MD retiming in terms of execution-time nodes, we model each graph with different cases of timing parameters. We proceed to model each one with several pairs of execution-time nodes $(t(A), t(M))$, where the adder and multiplier execution-times are respectively indicated by $t(A)$ and $t(M)$. We indicate that each one of three techniques achieves the same final MDFG when using a pair of execution time nodes $(n \times t(A), n \times t(M))$ which are the multiple of the pair $(t(A), t(M))$. For this, we just employ four pairs of execution time nodes as indicated in Table 3. Those values are chosen among the most frequent experiments. The minimal cycle period values are illustrated in Table 3.

To realize the contribution of our technique and guarantee a reliable comparison, we select optimal retiming functions to be used for the three retiming techniques. Our technique employs less MD retiming functions than the existing techniques. As described in paragraph 2.3, while incremental and chained techniques achieve the fully parallelized MDFGs of the IIR filter using 4 retiming functions, our technique employs no more than two. The retiming functions are decreased from five to four in the case of the WDF, and from two to one in the case of the WFT.

After providing the final MDFGs, we generate their respective algorithms to extract their execution times [1].

TABLE III. EXECUTION TIME NODES AND NUMBER OF MD RETIMING FUNCTION IN TERM OF TECHNIQUES

Benchmark	MDFG	Timing parameters			MD retiming number	
		$t(A_i)$	$t(M_j)$	c_{min}	Incremental or Chained MD retiming	Delayed MD retiming
IIRF	G1	1	2	2	4	2
	G2	1	3	3	4	2
	G3	1	4	4	4	1
	G4	2	5	5	4	2
WDF	G5	1	2	2	5	4
	G6	1	3	3	5	4
	G7	1	4	4	5	4
	G8	2	5	5	5	4
WFT	G9	1	2	2	2	1
	G10	1	3	3	2	1
	G11	1	4	4	2	1
	G12	2	5	5	2	1

We present in Table 4 the execution-time values in terms of time units of each MDFG in function of MD retiming techniques. We note that the cycle numbers are directly deduced from the execution time. The row "improve" presents the benefit in terms of execution-time of the results generated by our techniques compared to those generated by the existing ones, which accounts for an average improvement of 41.57% compared to the Incremental technique and 11.55% compared to the chained technique.

TABLE IV. EVOLUTION OF EXECUTION-TIME IN TERMS OF MD RETIMING TECHNIQUES

MDFG	Incremental MD retiming	Chained MD retiming	Delayed MD retiming
G1	956	632	516
G2	1434	948	774
G3	1912	1264	932
G4	2390	1580	1290
G5	3548	1800	1760
G6	5322	2700	2640
G7	7096	3600	3520
G8	8870	4500	4400
G9	326	276	238
G10	489	414	357
G11	652	512	476
G12	815	690	595
Improve	41.57%	11.55%	

V. CONCLUSION

In this paper, we have described the theory of the delayed multidimensional retiming technique. It allows scheduling the MD applications with the minimal cycle period using the least retiming functions in order to avoid achieving a full parallelism. It provides MD applications with optimized execution times compared to those provided by the "incremental" and "chained" techniques. The improvement averages deduced from the experimental results prove that delayed technique is beneficial when applying it on real-time applications.

In our future works, we will be interested to identify the evolution of code sizes in terms of our MD retiming. This study will allows us to extend the delayed technique in order to optimize both execution time and code size in MD applications.

REFERENCES

[1] N. L. Passos and E. H.-M. Sha, "Achieving full parallelism using multi-dimensional retiming," J. IEEE Trans. Par. Dist. Syst., Vol. 7, Iss. 11, nov. 1996, pp. 1150-1163, doi:10.1109/71.544356.

[2] M. Sheliga, N. L. Passos and E. H.-M. Sha, "Fully parallel hardware/software codesign for multidimensional DSP applications," Proc. 4th int. Workshop on Hardware/Software Co-Design (CODES'96), mar. 1996, pp. 18-25, doi:10.1109/HCS.1996.492222.

[3] N. L. Passos and E. H.-M. Sha, "Scheduling of Uniform Multi-Dimensional Systems under Resource Constraints," IEEE Trans. on VLSI Syst. Vol. 6, Iss. 4, dec. 1998, pp. 719-730, doi:10.1109/92.736145.

[4] Q. Zhuge, C. Xue, M. Qiu, J. Hu and E. H.-M. Sha, "Timing Optimization via Nest-Loop Pipelining Considering Code Size," J. Microproc. Microsyst., vol. 32, Iss. 7, oct. 2008, pp. 351-363, doi:10.1016/j.micpro.2008.02.002.

[5] N. L. Passos , D. C. Defoe, R. J. Bailey, R. H. Halverson, and R. P. Simpson, "Theoretical Constraints on Multi-Dimensional Retiming Design Techniques," Proc. visual information processing (SPIE), Vol. 4388, Apr. 2001, pp. 238-245.

[6] B.R. Rau, "Iterative modules scheduling: an algorithm for software pipelining loops," Proc. Int. symp. on MICROarchitecture (MICRO-27), Dec. 1994, pp. 63-74, (New York, USA, Nov. 30-Dec. 02, 1994)., doi:10.1145/192724.192731.

[7] T. O'Neil, S. Tongsima, and E. H.-M. Sha, "Extended retiming: Optimal scheduling via a graph-theoretical approach," Proc. the Acoustics, Speech, and Signal Processing (ICASSP '99), Mar. 1999, volume 4, pp. 2001–2004, doi:10.1109/ICASSP.1999.758320.

[8] T. O'Neil, S. Tongsima, and E. H.-M. Sha, "Optimal scheduling of data-flow graphs using extended retiming," Proc. Int. Conf. Parallel & Distributed Computing System (ISCA 12th), Dec. 1999, pp. 292-297.

[9] E. Leiserson and B. Saxe, "Retiming synchronous circuitry," J. of Algorithmica, vol. 6, Jun. 1991, pp. 5-35, doi:10.1007/BF01759032.

[10] Q. Zhuge, Z. Shao and E. H.-M. Sha, "Timing optimization of nested loops considering code size for DSP applications," Proc. of the 2004 Int. Conf. on Parallel Processing (ICPP 2004), Aug. 2004, pp. 475-482, doi:10.1109/ICPP.2004.1327957.