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. International Journal of High Performance Systems Architecture (IJHPSA), 2015, 5 (3), pp.178-191. 10.1504/IJHPSA.2015.070393 . hal-01796770

HAL Id: hal-01796770

https://hal.science/hal-01796770v1

Submitted on 22 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
Fax : + 33 1 45 92 66 99
Email: yaroub.elloumi@esiee.fr
Email: mohamed.akil@esiee.fr

Mohamed Hedi Bedoui
[2]University of Monastir, Faculty of Medicine of Monastir
Laboratory of Medical Technology and Image Processing
5019, Monastir, Tunisia
Fax:+216 73 46 07 37
Email: medhedi.bedoui@fmm.rnu.tn

Abstract Multidimensional Retiming (MR) is a software pipelining approach that ensures increasing the instruction-level parallelism across all the nested loops. All the MR techniques aim at achieving a full parallelism in order to schedule applications with a minimal cycle period. However, the growth of code sizes in terms of parallelism level engenders the rise in cycle period numbers. Thus, fully parallel multidimensional applications frequently face limiting factors when implemented on real-time systems.

This paper presents a novel technique, called delayed MR, which schedules nested loops with a minimal cycle period, without achieving full parallelism. It is formulated into two efficient steps whose first one sweeps the nested loops with the target of selecting and ordering paths, whereas the second one applies an optimal MR to the selected paths. Our technique is verified by implementing several nested loops in NVIDIA architectures. The experimental results show that our technique achieves average improvements on execution time of 32.8% compared to the incremental technique and 19.35% compared to the chained one.

Key Words: nested loops, parallelism, Software pipelining, loop transformation.

Biographical notes: Yaroub Elloumi received his PhD degree in co-supervision with the University of Paris-Est Marne-la-Vallée (France) and the University of Sfax (Tunisia) in 2013. He received his research MS degree in real-time systems from the University of Sousse (Tunisia) in 2008. He currently teaches and does research with the position of assistant Professor in computer sciences. He is a member of both Gaspard Monge Computer Science Laboratory (LIGM CNRS UMR 8049) and Medical Technology and Image Processing Laboratory (LabTIM LR12ES06). His research interests are high level design of real-time system, parallelism techniques, and performance estimation.

Mohamed Akil received his PhD degree from Montpellier University (France) in 1981 and his doctorat d'état from the Pierre et Marie curie University (Paris, France) in 1985. He is currently a Professor in the Computer Science Department, ESIEE, Paris. He is now Dean of the faculty. He is a member of Institut Gaspard-Monge, unité mixte de recherche CNRS-UMLPE-ESIEE, UMR 8049. He serves on the technical programme committees of international conferences. His research interests include dedicated and parallel architecture for image processing, image compression and virtual reality. His main research topics are reconfigurable architecture and FPGA, high-level design methodology for multi-FPGA, mixed architecture (DSP/FPGA) and System on Chip (SoC), parallel architectures. Dr. Akil has published more than 80 research papers in the above areas.

Mohamed Hedi Bedoui received his PhD degree from Lille University in 1993. He currently teaches with the position of Professor of biophysics in the Faculty of Medicine of Monastir at the University of Monastir (Tunisia). He is actually the director of the Technology Medical Image

Laboratory (LabTIM LR12ES06) in the Faculty of Medicine of Monastir and is the president of the Tunisian Association of Promotion of Applied Research. His research interests are real-time and embedded systems, image & signal processing and hardware/software design in medical field, and electronic applications in biomedical instrumentation.

1. INTRODUCTION

A large proportion of real-time applications include loop nests, such as the ones used on signal and image processing, high-definition vision and remote sensing. The loop bodies generally present the most critical section in terms of execution time. In this context, several parallelization techniques are proposed to reduce the loop body execution time. The parallelism principle aims to select the independent processing in order to execute them into the same time interval. These techniques can be classified in terms of granularity, which are the iteration level parallelism and the instruction level parallelism. The first type guarantees exploring the nested loops in order to identify and parallelize the independent iterations [Grosser et al. 2013; Liu et al. 2011; Qasem and Kennedy 2008; Liu et al. 2009, Xue et al. 2007; Wang et al. 2014]. The second one consists in applying the software pipelining in order to parallelize the independent instruction [Rong et al. 2007; Khan 2011; Zhuge et al. 2008]. Other optimization approaches targets applying both parallelism levels to enhance the performances of the provided implementations [O'neil and Sha 2005; Pouchet et al. 2008 ; Elloumi et al. 2013].

Regarding the instruction level parallelism, several techniques have been suggested for a single loop. Some ones consist of a software pipelining a specific loop level like the innermost loop [O'neil et al. 1999; Fellahi et al. 2007; Ferlin et al. 2011] or the outermost one [Turkington et al. 2008]. Others define the loop to pipeline in terms of parallelism performance enhancement [Rong et al. 2007]. Nevertheless, their performance improvement is limited since they parallelize one loop level. Few techniques explore the instruction level parallelism across all the nested loops [Morvan et al. 2011; Muthukumar and Doshi 2001] whose Multidimensional Retiming (MR) is distinguished as one of the most important technique [Passos and Sha 1996; Sheliga et al. 1996; Zhuge et al. 2008]. It models the loop body into a multidimensional Data Flow Graph (MDFG) [Xue et al. 2007; Lee et al. 2005; Liu et al. 2011; Zhuge et al. 2008] which allows explicitly featuring the granularity of instructions and their data dependencies. Then, it formulates the parallelism as a graph transformation theory.

All the MR techniques aim at scheduling nested loops with a minimal cycle period. For this purpose, they iteratively apply the parallelism until achieving a full parallelism in the instruction level. Due to the loop-carried dependencies, the parallelism leads to shift instructions in both sides of the nested loops, which are called prologue and epilogue. The code size goes up proportionally with the parallelism level. Accordingly, reaching a full parallelism requires adding a large code overhead. Yet, the code rise implies several implementation disadvantages. First, the shifted code size needs a similar cycle period number to be executed, which represents a wasted execution time [Zhuge et al. 2008]. Second, it leads to raise the amount of local and cache memories [Khan 2011; Fellahi and Cohen 2009]. Therefore, with the purpose of achieving a minimal cycle period, the full parallel instruction in the nested loops results in a non-optimal execution time.

In this paper, we show how to schedule the loop body with the minimal cycle period without getting a full parallelism. We put forward a new technique of MR, called "delayed MR". It efficiently insures exploring the characteristics of data dependencies and computation times in order to retime data paths. Then, it employs a theoretical process to select and thereafter iteratively performs an MR function. Thus, the parallelism level, and consequently the overhead are reduced, compared to the existing

techniques, while scheduling applications with the minimal cycle period. As a result, it provides implementations with enhanced execution times.

The rest of the paper is organized as follows. In section 2, we present the basic concepts of modeling and retiming multidimensional applications. In section 3, we present the theory of extracting the algorithmic characteristics, selecting an MR function, and iteratively applying the selected function. In section 4, we describe the delayed MR technique and we present the corresponding algorithms. The experimental results are presented in section 5, followed by concluding remarks in section 6.

2. BASIC CONCEPTS

2.1 Multidimensional data flow graph

The MDFG is an extension of the classic data flow graph where each node presents an instruction and each edge presents a data dependency. The main characteristic is the ability to represent nested iterative and recursive structures. In fact, one iteration is similar to executing all nodes once. The repetitive aspect is formulated by two concepts: the dimension n of the MDFG which is the nested loop number, and the delays which are edge weights formulating the loop-carried dependencies. An MDFG is modelled as $G = (V, E, d, t)$, where V is the node set, E is the edge set, $d(e)$ is the multidimensional delay of edge e, and $t(u)$ is the computation time of the node u. For $e: u \rightarrow v$, A $d(e)$ edge delay is modelled by a vector with n indexes such as $d(e) = (c_1, c_2, \dots, c_n)$. Each index corresponds to a single loop in a way that c_k presents the difference between the iteration executing v and the other one executing u of the loop k: If the node u is executed in the iteration x of the loop k, then the node v is executed in the iteration $(x + c_k)$.

As an example, the Jacobi algorithm [Bondhugula et al. 2008], shown in Fig. 1(a), is modelled by the MDFG in Fig. 1(b). It is composed by four nodes like the number of the innermost loop instructions. While the algorithm includes two nested loops, each edge in the MDFG is labelled by a delay with two indexes, where $d(e) = (d.x, d.y)$. The "$d.x$" and "$d.y$" terms are in relation with the outermost loop and the innermost one, respectively. The $A1$ and $A2$ instructions are computed in the same iteration whether for the innermost or the outermost loops. For this purpose, the $A1 \rightarrow A2$ edge is labelled by the delay $d(e_1) = (0,0)$ which is called “zero-delay edge”. For the data dependencies between D2 and A2, if D2 is executed in the iteration i of the outermost loop, then A2 is executed in the iteration $i + 1$. Similarly to the innermost loop, D2 is executed in the previous iteration of the A2 execution. For this purpose, the delay value of the edge $D2 \rightarrow A2$ is equal to $(1, -1)$.

The notation $p: v_i \xrightarrow{e_m} \dots \xrightarrow{e_n} v_h$ is used to mean that p is a path from v_i to v_h. The delay and the computation time of a p path are respectively equal to $d(p) = \sum_{k=m}^{k=n} d(e_k)$ and $t(p) = \sum_{k=i}^{k=h} t(v_k)$. A p path, whose $d(p) = (0, \dots, 0)$, is called “zero-delay path”. Critical p_{cr} paths are the ones having the maximum computation time among all zero-delay paths in the MDFG ($t(p_{cr}) = \max\{t(p), d(p) = 0\}$). 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 $C(G)$, where $C(G) \geq t(p_{cr})$. For example, the critical paths is structured as $p: A1 \rightarrow A2 \rightarrow D1 \rightarrow D2$. Assuming that $t(A1) = t(A2) = 1$ and $t(D1) = t(D2) = 2$, the cycle period of the Jacobi algorithm is $C(G) = 6$. The theoretical schedule is represented in Fig. 2 where the nodes belonging to the same iteration are modelled by the same pattern. In the case where $T = 10$ and $N = 10$, the algorithm requires 90 cycle periods and hence 540 time units to be executed.

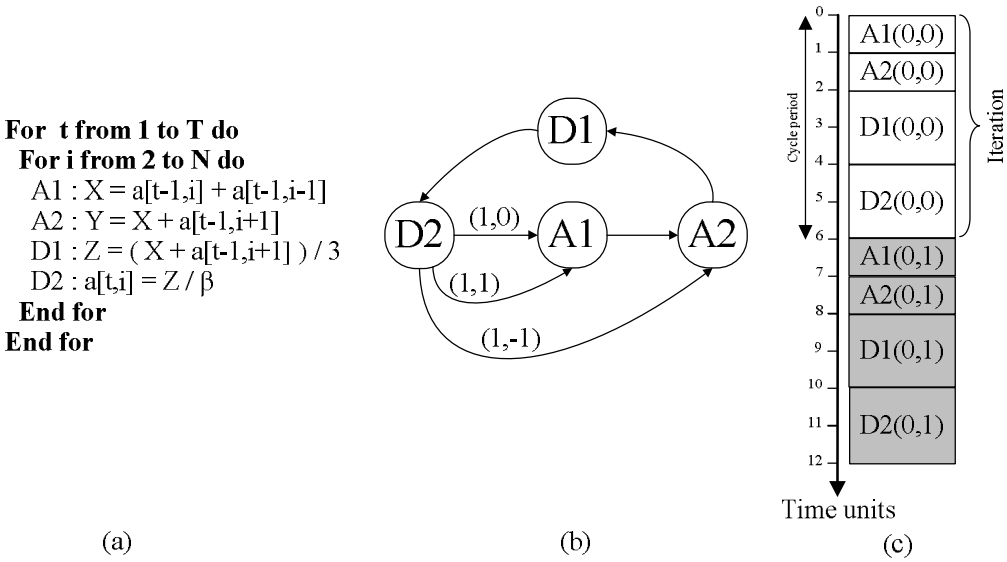

Fig. 1. The Jacobi algorithm : (a) code, (b) MDFG, (c) static schedule

2.2 Multidimensional retiming

The MR aims at reducing the cycle period of the nested loop execution. It proceeds to reduce the critical path execution times by decreasing their node numbers. With this objective, it shifts nodes from their original iteration in order to execute them inside another one. Thus, it implies increasing parallelism at the instruction level with the aim of optimizing the cycle period. For a retimed node, modifying its belonging iteration consists in modifying their edge delays. Therefore, the MR is modelled as a graphical transformation that modifies the MDFG delays, while preserving the functional behaviour of the initial MDFG [Passos and Sha 1996]. The MR is modelled as a retiming vector $r(u) = (r_1, \dots, r_n)$, where $u \in V$ and n is the loop dimension. The $r(u)$ function presents the offset between the original iterations containing u and the ones after retiming; i.e., 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 $i - r_k$. Figure 2(b) shows the Jacobi algorithm after applying $r(A1) = (0,1)$. Each ith occurrence of $A1$ is shifted up and executed in the previous iteration of the innermost loop, where $1 \leq i \leq n$. The first $A1$ occurrence belonging to the first iteration of the innermost loop is shifted upstream the retimed loop as the first instruction in Fig. 2(a), which is called prologue. This implies that all $A1 = (i, 0)$ instructions are executed outside the innermost loop whatever the i index is. Correspondingly, the complementary instructions of the last iteration, which are called epilogue, are executed downstream the innermost loop.

By focusing on the innermost loop algorithm in Fig. 2(a), we notice that instruction $A1$ does not have any direct data dependency to the other instructions belonging to the same iteration. Accordingly, the $A1$ instruction is to be executed in parallel with the remaining instructions. As an example, we choose to execute $A1$ instructions in parallel with $A2$ ones, as shown in the static schedule of Fig. 2(c). Thus, the cycle period is reduced from 6 to 5 time units. However, the code size is doubled due to the prologue and epilogue instructions. Therefore, in the case where $T = 10$ and $N = 10$, the Jacobi algorithm requires 100 cycle periods. Despite the fact that the cycle period number is raised, the execution time is reduced from 540 to 500 time units, due to the cycle period value.

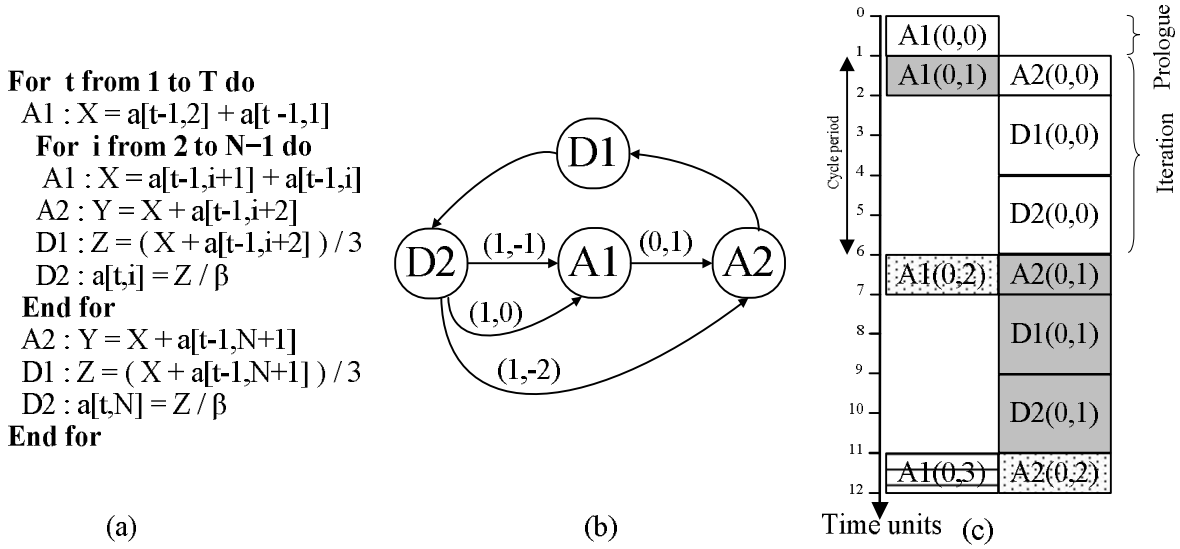

Fig. 2. Jacobi algorithm after $(A1) = (0,1)$: (a) code, (b) MDFG, (c) static schedule

A retiming function is called legal if it preserves the functionalities of the original code. In this context, the retiming function selection is based on the MDFG scheduling vector. It consists in identifying the set of all s vectors where $d(e) \times s > 0$ for every $d(e) \neq (0, \ldots, 0)$. A legal MR r is any vector orthogonal to s [Passos and Sha 1996; Sheliga et al. 1996]. The provided MDFG is called realizable and can be executed following an updating scheduling vector. In the case of the Jacobi algorithm, the original MDFG can be scheduled following the vector $s = (1,0)$, hence $r(D) = (0,1)$ is a legal MR of the Jacobi algorithm. So, it provides a realizable MDFG that can be executed following the scheduling vector $s = (3,1)$.

2.3 Multidimensional retiming techniques

All MR techniques aim at reaching the minimal cycle period by achieving a fully parallel algorithm, and hence a final MDFG without any zero-delay edge. The incremental and chained MR techniques [Passos and Sha 1996; Sheliga et al. 1996] re-apply successively the MR transformation to shift each time the first nodes of critical paths. Therefore, if $p: v_1 \rightarrow \cdots \rightarrow v_n$ is a critical path, then any existing technique employs $(n-1)$ MR transformations. To fully parallelize the Jacobi algorithm, the MR is applied respectivelly to A1, A2 and D1 nodes. Accordingly, the minimal cycle period is got where each iteration requires two time units to be executed.

On the other hand, when analyzing the static schedule in Fig. 3(c), we deduce that the data provided by $A1$ and $A2$ nodes are used one time unit after their generations, which presents an untapped time. Furthermore, the fully parallel scheduling implies adding 6 instructions in both prologue and epilogue in comparison to the original one, as shown in Fig.3(a). Knowing that the overhead is executed with a low parallelism level, it requires 6 cycle periods added to each outmost loop iteration execution. Consequently, in the case where $T = 10$ and $N = 10$, the cycle periods of the fully parallelized MDFG increase from 90 to 120 and then requires 240 time units to be executed. As a result, although fully parallel applications are scheduled with a minimum cycle period, they do not allow respecting strict timing constraints.

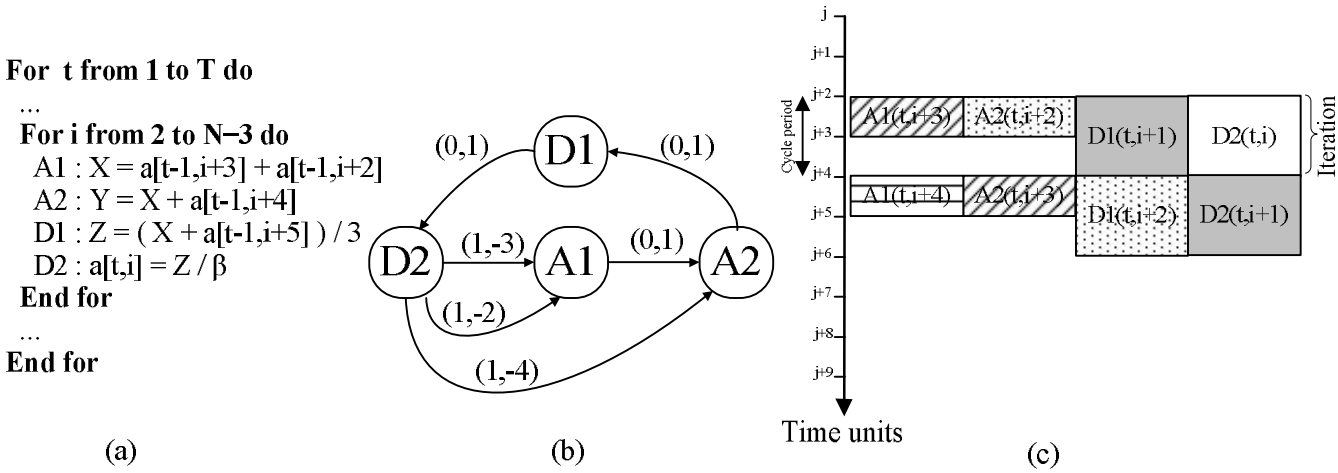

Fig. 3. Fully parallel Jacobi algorithm : (a) code, (b) MDFG, (c) static schedule

An MR technique, called Software Pipelining for NEsted loops (SPINE) [Zhuge et al. 2008], applies the acyclic graph retiming [Leiserson and Saxe 1991] to the MDFG by simulating the delays as registers. This technique is not performing in all cases. Firstly, the classical retiming theory is based in the non-negativity of register numbers belonging to data dependencies, while the MR allows using negative delays. Secondly, some multidimensional delays are invariant while the classical retiming [Leiserson and Saxe 1991] redistributes all registers. Another MR technique was described in [Elloumi et al. 2011], which groups nodes in the same cycle period after achieving the full parallel MDFG, to decrease the prologue and epilogue codes. Despite the originality of the idea, this technique is constrained by the data dependencies of the full parallel applications.

3. THEORY OF DELAYED MULTIDIMENSIONAL RETIMING FOR NESTED LOOPS

3.1 Motivating example and approach overview

The MDFG shown in Fig.3(b) can be executed following the schedule vector $s = (1,0)$ and hence retimed through $(0,1)$ or $(0,-1)$. While all $A1$ incomming edges are non-zero delays, the $r = (0,-1)$ function is to be applied to the $A1$ node, which results in the MDFG in Fig. 4(b). Based on the delay values, the transformation provides a realizable MDFG and preserves the functionality of the whole application [Passos and Sha 1996; Sheliga et al. 1996]. Indeed, the $A1 \rightarrow A2$ path has a zero-delay; i.e., $A1$ and $A2$ nodes are still executed in the same cycle period and thus in the same iteration. Moreover, the $d(A1 \rightarrow A2) = (0,0)$ and $d(A2 \rightarrow D1) = (0,1)$ delays mean that if $A1(i,j)$ and $A2(i,j)$ are executed in the (i) iteration, then $D1(i,j)$ is executed in the $(i+1)$ iteration, as shown in the static schedule of Fig.4(c). This modification can be considered as performing the MR to the whole path $p: A1 \rightarrow A2$.

These delay values lead to the algorithm given in Fig.4 (a), which contains 6 instructions as a prologue and 3 instructions as an epilogue. The overhead is then decreased by 25% compared to the fully parallel algorithm in Fig. 3(a). Thereafter, the cycle number is reduced from 120 to 110. Besides, all zero-delay paths are executed in two cycle periods, and accordingly scheduling the algorithm with the minimal cycle period C(G)=2. Consequently, the execution time is decreased from 240 to 220 time units, which presents an improvement of 8.33% even though the MDFG is not fully parallelized.

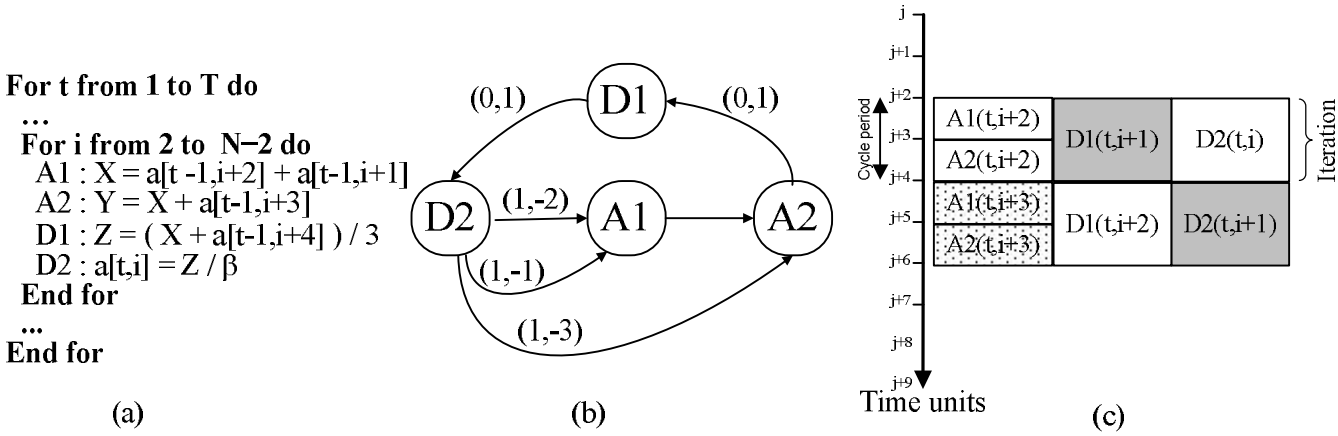

Fig. 4. The Jacobi algorithm provided by delayed MR: (a) code, (b) MDFG, (c) static schedule

Therefore, we present in this paper a novel approach that schedules an MDFG with a minimal cycle period, without achieving a full parallelism. Retiming the whole path is carried out instead of a node only. Contrary to the existing techniques, retiming functions are not applied to successive nodes but to distant ones. Thus, we call our technique "delayed MR".

3.2 Path selection

Our work aims to select zero-delay paths $p : u \to \cdots \to v$ in order to retime them. Knowing that each u and v nodes can be connected by several paths, the zero-delay condition should be verified by all paths connecting u and v. Accordingly, based on [Leiserson and Saxe 1991], $D(u,v)$ is defined as the minimum delay among all delay paths connecting u and v, as described in Equation (1). If just one path among those connecting u and v has a zero-delay, then $D(u,v) = (0,\ldots,0)$; else, $D(u,v) \neq (0,\ldots,0)$.

$$D(u,v) = \min\{d(p), \text{where } p: u \to v, u \in V \text{ and } v \in V\} \quad (1)$$

In addition, the u and v nodes can be connected with several zero-delay paths having different execution times. Based on our approach, all paths should be executed in the bound of the cycle period. Thus, $T(u,v)$ is defined as the maximum execution time among all zero-delay path connecting u and v nodes, as described in Equation (2).

$$T(u,v) = \max\{t(p) \text{ where } p: u \to v, u \in V, v \in V \text{ and } d(p) = D(u,v)\} \quad (2)$$

Our technique principle consists in scheduling the algorithm under the minimal cycle period. Generally, scheduling with a c cycle period means that all zero-delay paths are executed under c. This objective is formulated using the D and T parameters, as described in theorem 1.

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

1. $\varphi(G) \leq c$
2. 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 $u,v \in V$ where $T(u,v) > c$. If $D(u,v) = 0$, then there 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 is in contradiction with the first supposition. We suppose now that the second condition is verified, and let $u \to v$ be any zero-delay path p in G, then we have $D(u,v) = d(p) = (0,0)$ which implies that $t(p) \leq T(u,v) \leq c$. □

Based on theorem 1, achieving the first condition is synonymous to providing an MDFG whose D and T matrices respect the second one; i.e, no path having an execution time exceeding $c_{\min}$ should have a non-zero delay. Therefore, the paths to retime are the ones having $D(u,v) = (0,\dots,0)$ and $T(u,v) \leq c_{\min}$. Consequently, the delayed multidimensional retiming technique proceeds to explore the $D(u,v)$ and $T(u,v)$ values in order to select the paths to retime.

So, those values are defined and ranged respectively in two matrices $(V \times V)$ called D and T, in a way that each $p : u \rightarrow v$ path is indexed by the cell with the u line and the v column, as indicated in algorithm 1. It starts by computing $D(u,u)$ and $T(u,u)$ which represent the diagonal line values of both matrices, and $D(u,v)$ and $T(u,v)$ corresponding each edge. Afterwards, each path is associated to its outcoming edges to compute its $D(u,v)$ and $T(u,v)$. This step is incrementally repeated in the direction of the data dependencies. The D and T matrices of the Jacobi algorithm are respectively exposed in Table I and Table II where zero-delay path cells are represented by a gray color.

ALGORITHM 1. D and T matrices generation

```
Input : a realizable MDFG G =(V,E,d,t)
Output : matrices D and T
/* fill the diagonal lines of D and T matrices */
For each node u∈V do
    D(u,u)←(0,0)
    T(u,u)←t(u)
End for
/* fill the cells that correspond to the edges*/
For each edge e∈E where e:u→v do
    D(u,v)←d(e)
    T(u,v)←t(u)+t(v)
    Add element (u, v, D(u,v), T(u,v)) to EDGE list
End for
PATH←EDGE
/* fill the path cells of D and T matrices */
Repeat
    For each element (u, v , D(u,v), T(u,v)) of PATH do
        For each element (x, y, D(x,y), T(x,y)) of EDGE do
            If v = x then
                D(u,y) ← D(u,v) + D(x,y)
                T(u,y) ← T(u,v) + T(x,y)
                Add (u, y, D(u,y), T(u,y))  to R list
            End if
        End For
    End For
    PATH←R
Until D and T are totally filled
```

3.3 Path retiming

The work described in [Sheliga et al. 1996; Passos and Sha 1998] admits that any r retiming function is deduced only from the non-zero-delay edges. The theoretical choice is independent from the node that will be retimed. However, the previous techniques are restricted to applying the MR only to the nodes having all incoming edges with non-zero-delays. Our objective consists in verifying if the

selected function is able to retime a whole path. For this purpose, theorem 2 proves that a legal retiming function of a u node can shift a zero-delay path composed by two nodes.

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 is a legal multidimensional retiming of u, then r is a legal multidimensional Retiming of v.

PROOF. Let $p: \ldots \xrightarrow{(a_1,\ldots,a_n)} u \xrightarrow{(0,\ldots,0)} v \xrightarrow{(b_1,\ldots,b_n)} \ldots$ be a path in G. $r(u) = (r_1, \ldots, r_n)$ is a legal multidimensional retiming of G, which means that there exists a scheduling vector s where $(r_1, \ldots, r_n) \times s \geq 0$ and $(b_1, \ldots, b_n) \times s \geq 0$. Adding those two inequalities shows that $(b_1 + r_1, \ldots, b_n + r_n) \times s \geq 0$. Knowing that $(a_1 - r_1, \ldots, a_n - r_n) \times s \geq 0$ and so $(0, \ldots, 0) \times s \geq 0$, the MDFG containing $p: \ldots \xrightarrow{(a_1-r_1,\ldots,a_n-r_n)} u \xrightarrow{(0,0)} v \xrightarrow{(b_1+r_1,\ldots,b_n+r_n)} \ldots$ is realizable, which can be executed following the scheduling vector s. Thus, r(v) is a legal multidimensional retiming. □

This theorem proves that the legal MR for a node can be applied on its successor, if the latter has only one incoming edge. Moreover, it is easy to observe that the previous theorem can be used on successive nodes that respect the same condition. Therefore, theorem 2 allows identifying a legal retiming function for a zero-delay path.

Table I. D matrix of the initial Jacobi Algorithm

u\v	A1	A2	D1	D2
A1	(0,0)	(0,0)	(0,0)	(0,0)
A2	(1,0)	(0,0)	(0,0)	(0,0)
D1	(1,0)	(1,0)	(0,0)	(0,0)
D2	(1,0)	(1,0)	(1,0)	(0,0)

Table II. T matrix of the initial Jacobi Algorithm

u\v	A1	A2	D1	D2
A1	1	2	4	6
A2	6	1	3	5
D1	5	6	2	4
D2	3	4	6	2

The growing size of the MDFG is proportional to the set of the selected zero-delay paths, and thus to the number of required MR transformation. Nevertheless, defining a retiming function for each path cannot be an adequate solution. In fact, the set of nodes that have all incoming edges with non-zero-delays can be retimed using the same MR function. So, we identify the zero-delay paths that can be retimed with the same function, as described in theorem 3.

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

PROOF. Let s a scheduling vector where $d(e) \times s > 0$ and r is orthogonal to s. based on [Passos and Sha 1996], if X is a node set whose all incoming edges having non-zero delays, then r(X) is a legal multidimensional retiming function. Taking into account the theorem 4.2, the r(Y) is a legal multidimensional retiming function. □

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

3.4 Successive path retiming

Basically, achieving the c_{min} requires applying the MR several times. Taking the example of the Jacobi algorithm, reaching the c_{min} needs applying three retiming functions. In fact, the values of the retiming indexes $d.x$ and $d.y$ increase in terms of retiming function rank; i.e., the greater the retiming rank is, the more important the index values are. Theorem 4 verifies that the first retiming function is the one having the smallest index values in relation with the successive function indexes.

Theorem 4. Let $G = (V, E, d, t)$ be a realizable MDFG, $r(u) = (r_1, \dots, r_n)$ a legal multidimensional retiming of G, and $G_r = (V, E, d_r, t)$ be the MDFG provided by retiming G by r. If $R(u) = (R_1, \dots, R_n)$ a legal multidimensional retiming of G_r then $r_i < R_i$ where $1 \leq i \leq n$.

PROOF. Let $(a_1, \dots, a_n)$ a non-zero delay edge of G, $s = (s.x, s.y)$ a scheduling vector of G_r, which means that $(a_1 - r_1, \dots, a_n - r_n) \times (s.x, s.y) \geq 0$ and so $(r_1, \dots, r_n) \times (s.x, s.y) \geq 0$. Adding those two inequalities shows that $(a_1 - r_1 - R_1, \dots, a_n - r_n - R_n) \times (s.x, s.y) \geq 0$, which means that $r_i < R_i$. □

On the one hand, the rise in the index values implies a similar increase in the prologue and epilogue size. Knowing that those codes are executed with a low parallelism level, they lead to a significant growth of the cycle number and hence in the execution time. On the other hand, the MR feasibility admits a restriction with respect to the iteration bounds of the nested loops [Passos et al. 2001]. The retiming function term "$d.x$" should not exceed the iteration bounds n of the corresponding loop. Taking as an example the IIR filter, the first retiming function is (1,-1), whereas the last one is (8, -15) [Passos and Sha 1996], which means that 15 innermost iterations are to be overlapped. Consequently, extracting and applying several retiming functions results in providing implementations with a large overhead and so with non-optimal execution times.

The delayed technique principle consists in applying optimal retiming functions. Indeed, Theorem 4 proves that the MR having the minimal index values is the first one to be applied. In this context, theorem 5 presents the way to extract several MR functions directly from only one.

Theorem 5. 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 $x \in X$, and an integer $k > 1$. If $r(X)$ is a legal MD retiming then $(k \times r)(Y)$ is a legal multidimensional retiming.

PROOF. r(X) is a legal multidimensional retiming function implies the existence of a scheduling vector s where $r \perp s$ and then $r \times s = 0$. Knowing that $k > 0$, the equation $(k \times r) \times s = 0$ is prooved. Thus, $(k \times r)(X)$ is a legal multidimensional retiming. Based on theorem 3, $(k \times r)(Y)$ is also a legal mltidimensional retiming. □

This theorem allows retiming two successive paths with the same function r. We take as an example the initial MDFG of the Jacobi algorithm whose the critical path is $p_{ch}: A1 \rightarrow A2 \rightarrow D1 \rightarrow D2$. There are two zero-delay paths to be retimed which are $p_1: A1 \rightarrow A2$ and $p_2: D1$. Since (0,1) is a legal MR, theorem 5 allows applying $(2 \times (0,1))$ to the p_1 path. Therefore, the $d(A2 \rightarrow D1)$ delay enables retiming p_2 with $(1 \times (0,1))$ leading to the MDFG, shown in Fig. 4(b), whose code is composed by 19 instructions and then executed in 462 cycle periods. However, if $p_1: A1 \rightarrow A2$ is retimed with (0,1), $p_2: D1$ is to be retimed with (1,-2), then the final code made up of 35 instructions and requires 557 cycle periods to be executed. Consequently, using the minimal MR function leads to implementations with reduced code sizes, and thereafter execution times whose improvement average is 36.66%.

Thus, based on theorem 5, successive zero-delay paths can be retimed using the same function; i.e., each one has a retiming function $(k \times r)$ where k is a strict positive integer and is smaller than the one allocated to the predecessor path. So, using the optimal MR function provides an implementation with a minimal execution time compared to the one provided by using several MR functions.

4. DELAYED MULTIDIMENSIONAL RETIMING TECHNIQUE

The present work aims to achieve the c_{min} with a reduced overhead in order to optimize the execution time. The target idea consists in retiming whole paths instead of nodes. Theorem 5 gives us the alternative to use the optimal MR function to successive zero-delay paths. It is based on associating each path's integer level whose values are decreased in the direction of data path. Thus, all paths should be identified before retiming them.

For that purpose, our technique follows two steps that are described in sections 4.1 and 4.2, respectively. The first one explores the MDFG in order to identify the paths to retime and their level values. The selected paths are modelled in a similar graph called "Labelled Multidimensional Data Flow Graph (LMDFG)". The second step ensures the selection of an MR function r and applies it on the selected paths indicated in the LMDFG.

4.1 LMDFG generation

The MR function should have the minimal values to provide an MDFG with optimal code sizes and execution times. Therefore, the k levels must have the minimal values, and hence start from 1 and increase by just one unit. To guarantee this criterion, identifying and allocating the k level should be done in the opposite direction of the data flow. So, our approach consists in swepping the MDFG incrementally in the opposite direction of the data dependencies, starting by the nodes having all outcoming edges with non-zero-delays.

Actually, the p path to retime should respect the second condition of theorem 1. Furthermore, to minimize the retiming function, the selected path should integrate nodes as many as possible. These constraints will be proved by exploring the D and T matrices, as described in algorithm 2.

Each p path is defined by its first and last node, respectively u and v. For each v node having all outcoming edges with non-zero-delays, we verify the v column in the matrices D and T as follows. If the cells $D(u,v) = (0,\dots,0)$ and $T(u,v) \leq c_{min}$, the node u is added to the R list. Afterwards, the R list is filtered to remove redundancy and save the longest paths. The R list contains the u first nodes of the path to be retimed. Accordingly, the predecessor nodes of the R list are labelled by a level value $k = 1$ and stored in the S list. Thereafter, the sweeping process is repeated starting from the S list nodes. The next selected path is labelled by an incremented level value. The process is stopped when all the MDFG nodes are tested.

In the case of the Jacobi algorithm, the sweeping process starts by the D2 node. Knowing that it is executed in two cycle periods, it is identified as paths to retime, and hence labeling its D1 predecessor node by a $k = 1$ level. Likewise, sweeping is pursued from the D1 node to labelling the A2 node by $k = 2$, as shown in the final LMDFG in Fig.5.

4.2 Delayed multidimensional retiming algorithm

The current technique aims at scheduling the MDFG with the c_{min}. An MR function r is extracted in order to apply it on the selected path. For this purpose, the selection of a legal MR r is carried out, as it is done in the existing techniques [Sheliga et al. 1996]. Then, algorithm 2 is used to generate the LMDFG that indicates the paths to retime and deduce the maximal k level.

ALGORITHM 2. LMDFG generation

```
Input: a realizable MDFG G =(V,E,d,t)
Output: LMDFG, level k
Identify the minimal cycle period c_min
Compute the matrices D et T (as described in algorithm 1)
/*identify the nodes having all incoming edges with non-zero delays*/
For e∈E where e:u→v do
    If d(e)≠(0,…,0) then
        Add v to S list
    End if
End for
init k ← 1
Repeat
/*explore the D and T columns of the S nodes */
    For each node v∈S do
        For each node x∈V do
            If D(x,v)=(0,…,0) and T(x,v)≤c_min and x∉R then
                Add x to R list
            End if
        End for
    End for
/* identifying the nodes to retime*/
    For each node p∈R do
        For each node q∈R do
            If D(p,q)=(0,…,0) then
                Delete q from R
            End if
        End for
    End for
/*labelling the nodes to retime*/
    For each node x∈R do
        For each u predecessor node of x do
            Add u to S1 list
            Label u by k
        End for
    End for
    Increase k
    S←S1
Until all nodes are tested
```

Afterthat, the selected MR r is applied on the LMDFG in the direction of the data dependencies. It starts from the nodes S having all incoming edges with non-zero-delays. Then, it sweeps the graph until finding the ones labelled by the maximal k level, which are called A nodes. Thereafter, it computes the $D_r = r \times k$, subtracts it from all the delays of the S incoming edges, and adds it to the outcoming edges of the labelled A nodes. These steps are repeated in a descending order of level values, until getting nodes whose levels are equal to 1, as described in Algorithm 3.

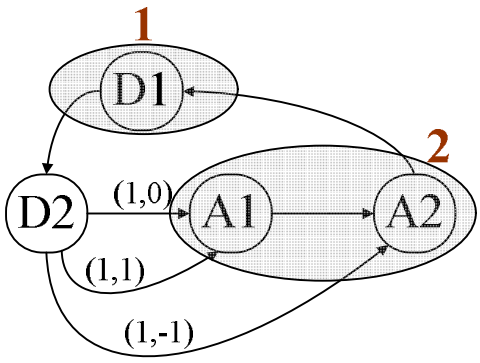

Fig. 5. Jacobi Algorithm LMDFG

Using the delayed MR, an MDFG scheduled with the minimal cycle period is always achievable and its efficiency is proved by theorem 6.

Theorem 6. Let $G = (V, E, d, t)$ be a realizable MDFG, the delayed multidimensional 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 the computation of D and T for each pair of nodes in the MDFG, which requires $O(E + V^2)$ times. Thereafter, algorithm 2 tests at most all cells of D and T matrices, which can be performed in $(n \times V)$ instructions where n is an integer value $n > 0$. Finally, algorithm 3 selects one multidimensional retiming function, which requires at most E instructions, and applies it on each selected paths. Thus, the delayed multidimensional retiming technique requires only $O(V^2 + E)$ times to be performed. □

ALGORITHM 3. Delayed multidimensional retiming

Input: a realizable MDFG G =(V,E,d,t)
Output: a realizable MDFG G_r=(V,E,d_r,t) with C(Gr)=cmin
Find a schedule vector s=(Sx, Sy) such as |Sx |+|Sy | is minimum
Select the MD retiming function r = (|Sy |, -|Sx |)
Provide the LMDFG and the maximal level k (as indicated in algorithm 2)
Define the S set of nodes having the incoming egdes with non-zero delay
For j from k to 1 **do**
    Define the set A of nodes assigned by j
    For each n∈S **do**
        Subtract the (j × r) from the n incoming edges
    End for
    For each n∈A **do**
        Add the (j × r) to the n outcoming edges
    End for
    S←(S-A) + successor (S∩A)
End for

5. EXPERIMENTAL RESULTS

In our experiments, the delayed technique is compared to the “incremental” and “chained” ones in terms of execution time. Our benchmarks include a set of 2D nested loops which are the Jacobi Algorithm (JA) [Bondhugula et al. 2008], the Wave Digital Filter (WDF) [Elloumi et al. 2013], the

Walsh-Fourier Transform (WFT) [Passos et al. 2001], and the Infinite Impulse Response Filter (IIRF) [Passos and Sha 1996]. To evaluate the parallelism level of the retiming techniques, the experiments should be conducted with a parallel architecture integrating an important core number. For this purpose, the provided implementations are evaluated using GPU architectures. In this context, we use the QUADRO-600 and the G-105-M NVIDIA devices, whose GPU-core numbers and frequencies are indicated in Table V. We recall that the parallelism in the delayed technique is chosen based on execution-time nodes. Therefore, the different computational instructions are identified in order to measure their execution times, as described in [Lai and Seznec 2011]. The used multidimensional applications are composed by adders, multipliers and dividers, whose execution time values are shown in the fifth column of Table V. to take it simpler, these execution time values are converted from seconds to cycle periods, which are indicated in the last column of Table V.

Table V. Timing parameters of the NVDIA architectures

Architecture	GPU-cores	GPU frequency (GHZ)	Memory frequency (MHZ)	Execution time instruction (10^{-5} second)			Execution time node (cycle period)		
				Add.	Mul.	Div.	Add.	Mul.	Div.
QUADRO-600	96	1.28	800	0.3	0.31	0.61	1	1	2
G-105-M	8	1.23	790	0.59	0.61	1.19	1	1	2

Thereby, the execution time nodes are used when running the MR techniques whose the employed MR functions and code sizes resulting from the provided algorithms are shown in Table VI. The delayed technique has used MR functions either less than the existing techniques or with decreased index values. Thus, the code size values mention that our technique achieves an average improvement of 25.52% for the incremental technique and 66.39% of the chained one.

Table VI. Code sizes in terms of multidimensional retiming techniques

Multidim. Appl.	Delayed technique		Chained technique		Incremental technique	
	MR functions	Code size	MR functions	Code size	MR functions	Code size
JA	(0,2),(0,1)	13	(0,3),(0,2), (0,1)	16	(0,1),(1,-3),(2,-4)	37
WDF	(0,1)	8	(0,2), (0,1)	12	(0,1), (1,-3)	41
WFT	(1,-1)	16	(2,-2), (1,-1)	32	(1,-1), (2,-3)	68
IIRF	(4, -4), (3,-3), (2,-2), (1,-1)	144	(4, -4), (3,-3), (2,-2), (1,-1)	144	(1,-1), (2,-3), (4,-7),(8,-15)	256
Improve				25.52%		66.39%

Accordingly, each algorithm is implemented with different input matrix sizes among (64*64, 96*96, 128*128). Each code is edited with the parallel computing platform CUDA. The parallel instructions are implemented in separately warps and the data dependencies are insured using the "_syncthreads()" directive. Subsequently, each code is compiled and executed in both architectures. The execution time values of the four multidimensional applications are respectively shown in Fig.6, Fig.7, Fig. 8 and Fig. 9.

For each matrix size, the implementation provided by the delayed technique presents a benefit in terms of execution time compared to the implementation generated by any other existing technique. This improvement is in relation to the reduced prologue and epilogue code size. The delayed technique accounts for an average improvement of 32.8% compared to the incremental technique and 19.35% compared to the chained one.

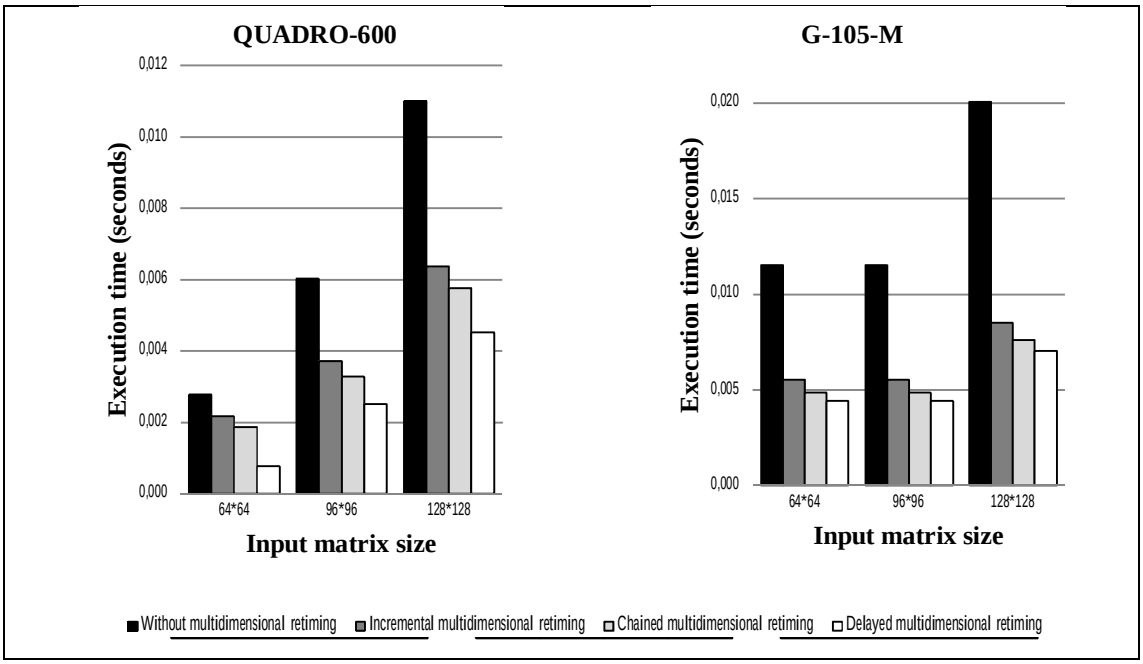

Fig. 6 Execution times of the Jacobi Algorithm

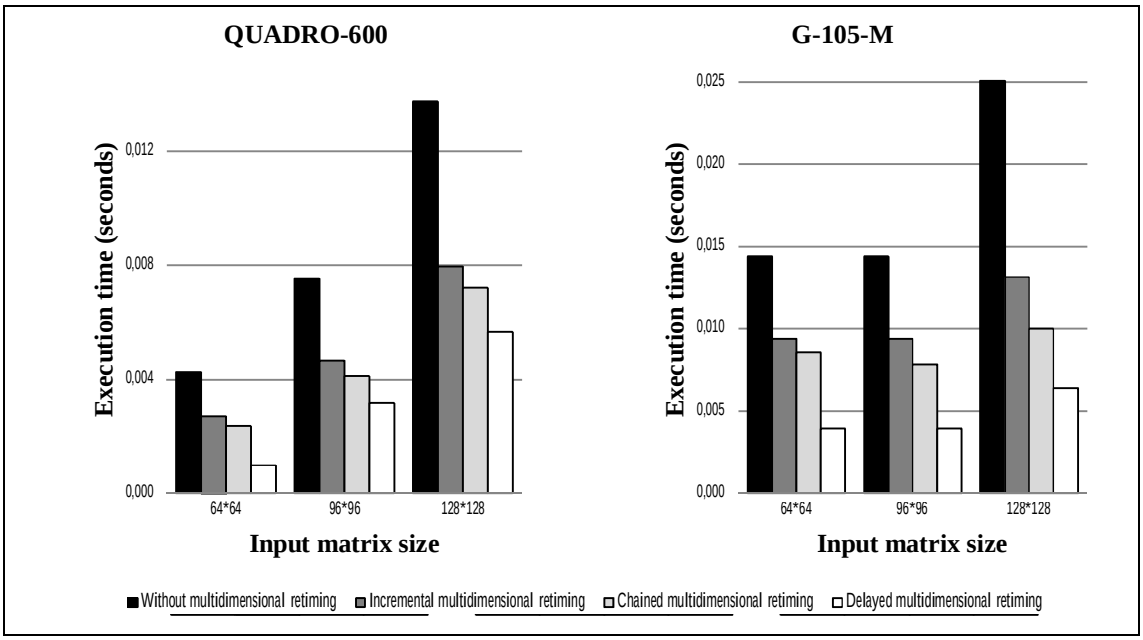

Fig. 7 Execution times of the Wave Digital Filter

Moreover, the delayed technique offers a significant execution time enhancement with respect to the initial code, whatever the input matrix size is. Furthermore, for each input matrix size, the execution time improvement of the delayed technique is verified whether for the QUADRO-600 or the G-105-M, with respect to the code provided by the incremental technique or the chained technique. It indicates that our technique contribution is guaranteed independently from the used parallel architecture.

For instance, this improvement is important where the loops bounds are shortened, achieving more than 25% when implementing the WFT with a 64*64 input matrix size. However, the existing techniques do not allow an execution improvement in all cases. The incremental technique leads to an opposite effect by increasing the execution times in the case of the WFT with a 64*64 input matrix size. Using the same implementation, the chained technique presents an improvement average of only 9.2% with respect to the initial code execution time.

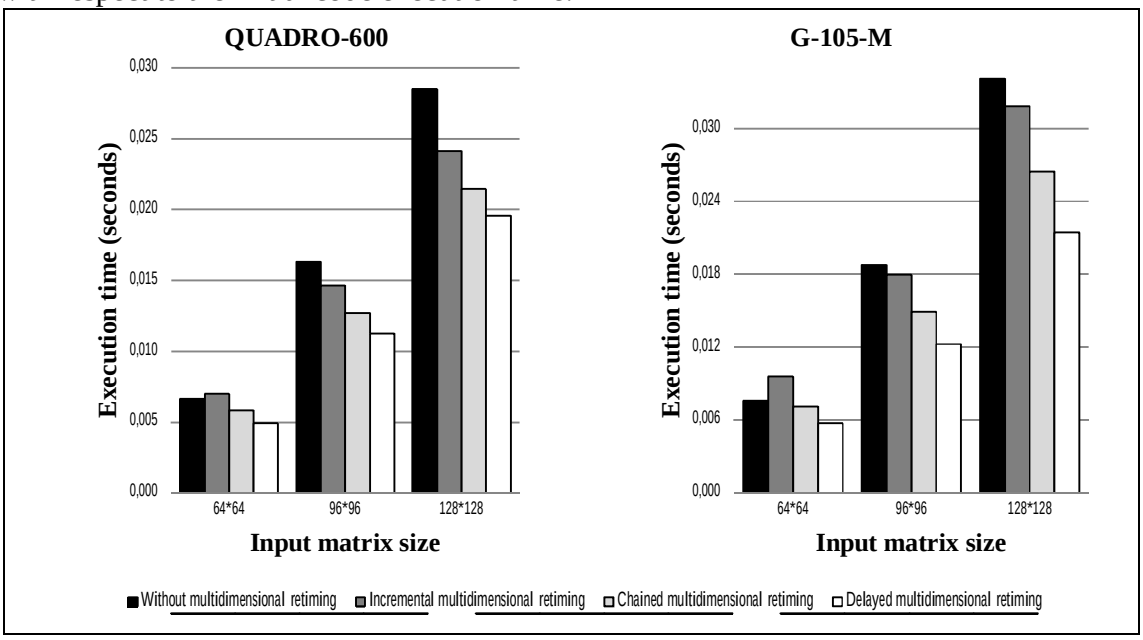

Fig. 8 Execution times of the Walsh Fourier Transform

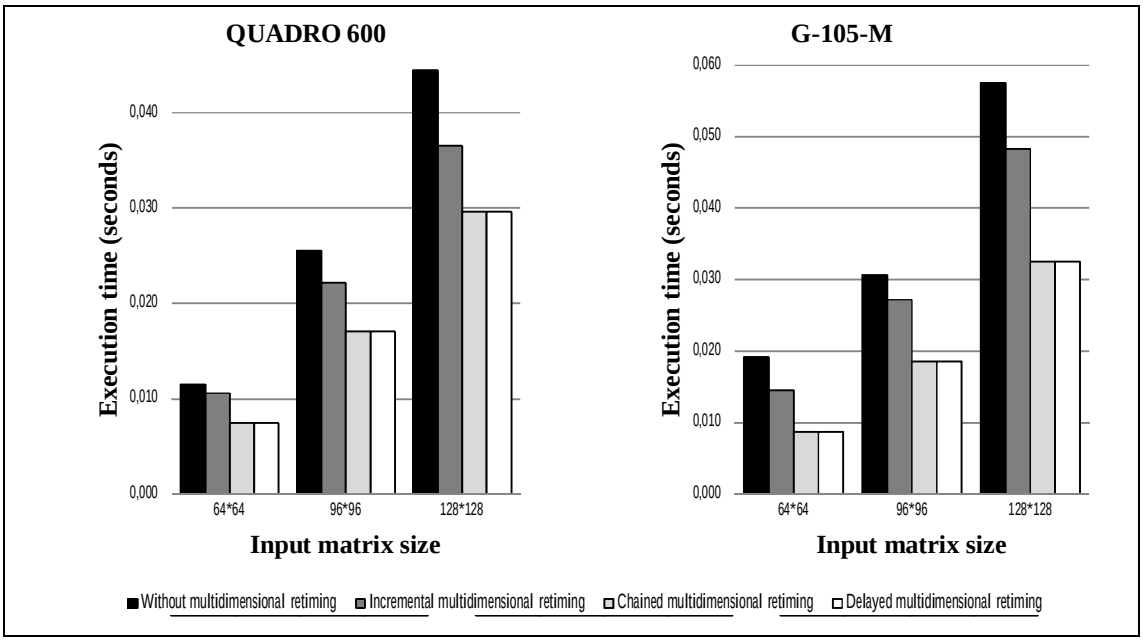

Fig. 9 Execution times of the Infinite Impulse Response Filter

This effect is caused by the ratio between the overlapped iterations and the loop bounds after the MR. in the case of the IIRF, the delayed and chained technique employ the same MR functions that engender generating the same implementation and hence the same execution time.

6. CONCLUSION

In this paper, we have proposed a new multidimensional retiming technique that allows scheduling the multidimensional applications with a minimal cycle period. The objectif is accomplished without an inevitable achievement of full parallelism. The experiment described in the last section proves that the current technique provides enhanced implementations in terms of execution time compared to those provided by the existing techniques. It should be noted that it is appropriate whatever the computational instructions are. Therefore, it is beneficial when applied on real-time systems.

In our future works, we will be interested in extending our technique in order to parallelize the instruction level on imperfect nested loops. In addition, we aim to conduct the suggested parallelism approach to optimize material resources such as memory and processor units. Furthermore, we will focus on using our multidimensional retiming technique with other optimization approaches (such as loop stripping, loop fusion …).

REFERENCES

Chun Jason Xue, Zili Shao and Edwin Hsing-Mean Sha. 2007. Maximize parallelism minimize overhead for nested loops via loop Striping. J. VLSI Sig. Proc. Syst. 47, 2 (May 2007), 153–167.

Nelson Luiz Passos and Edwin Hsing-Mean Sha. 1996. Achieving full parallelism using multi-dimensional retiming. J. IEEE Trans. Par. Dist. Syst.7, 5 (Nov. 1996), 1150-1163.

Michael John Sheliga, Nelson Luiz Passos and Edwin Hsing-Mean Sha. 1996. Fully parallel hardware/software codesign for multidimensional DSP applications. In Proceedings of the 4th International Workshop on Hardware/Software Co-Design (CODES'96), Pittsburgh(PA), 18–25.

Chun Jason Xue, Edwin Hsing-Mean Sha, Zili Shao and Meikang Qiu. 2008. Effective Loop Partitioning and Scheduling under Memory and Register Dual Constraints. In Proceedings of the conference on Design, automation and test in Europe (DATE'08). Munich(Germany), 1202–1207.

Qingfeng Zhuge, Zili Shao, Bin Xiao and Edwin Hsing-Mean Sha. 2003. Design space minimization with timing and code size optimization for embedded DSP. In Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS'03). Newport Beach(Canada), 144-149.

Nelson Luiz Passos and Edwin Hsing-Mean Sha. 1998. Scheduling of Uniform Multi-Dimensional Systems under Resource Constraints. J. IEEE Trans. VLSI Syst. 6, 4, (Dec. 1998), 719-730.

Qingfeng Zhuge, Chun jason Xue, Meikang Qiu, Jingtong Hu, and Edwin Hsing-Mean Sha. 2008. Timing Optimization via Nest-Loop Pipelining Considering Code Size. J. Microproc. & Microsyst. 32, 7 (October 2008), 351-363.

Nelson Luiz Passos, Delvin C. Defoe, Reynold J. Bailey, Ranette Halverson and Richard P. Simpson. 2001. Theoretical Constraints on Multi-Dimensional Retiming Design Techniques. In Proc. Of Visual Information Processing X (Apr. 2001). SPIE, Orlando(FL), 238-245.

Timothy William O'neil, Sissades Tongsima and Edwin Hsing-Mean Sha. 1999. Extended retiming: Optimal scheduling via a graph-theoretical approach. In IEEE conference on the Acoustics, Speech, and Signal Processing (ICASSP'99), Phoenix(AZ), 2001–2004, vol. 4.

Charles E. Leiserson and James B. Saxe. 1991. Retiming synchronous circuitry. Algorithmica. 6, 1-6 (June 1991), 5-35.

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui. 2011. Timing and Code Size Optimization on Achieving Full Parallelism in Uniform Nested Loop. J. of comput. 3, 7, July 2011, 68-77.

Naresh Maheshwari and Sachin Sapatnekar. 1998. Efficient retiming of large circuits. J. IEEE Trans. VLSI. 6, 1 (Mar. 1998),74–83.

Hai Zhou. 2005. Deriving a new efficient algorithm for min-period retiming. In Proceedings of the 2005 Asia and South Pacific Design (ASP-DAC'05), pages 990–993, jan. 2005.

Jia Wang and Hai Zhou. 2008. An Efficient Incremental Algorithm for Min-Area Retiming. Proceedings of the 45th annual Design Automation Conference (DAC'08). Anaheim(Canada), 528 - 533.

Naresh Maheshwari and Sachin Sapatnekar. 1998. Efficient minarea retiming of large level-clocked circuits". In procceding of the conference on design, automation and test in Europe (DATE'98). Paris, 840-847.

Hongbo Rong, Zhizhong Tang, Ramaswamy Govindarajan, Alban Douillet and Guang Gao. 2007. Single-dimension software pipelining for multidimensional loops. J. ACM trans. arch. code opt. 4, 1 (March 2007), 163 – 174.

Uday Bondhugula, Muthu Baskaran, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, Ponuswamy Sadayappan. 2008. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. Lecture Notes in Computer Science, Vol. 4959. 132-146.

Antoine Morvan, Steven Derrien and Patrice Quinton. 2011. Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion. In International Conference on Field-Programmable Technology (FPT'11). New Delhi(India), 1–10.

Kieron Turkington, George A. Constantinides, Konstantinos Masselos and Peter Y. K. Cheung. Outer Loop Pipelining for Application Specific Datapaths in FPGAs. J. IEEE trans. VLSI SYST. 16, 10 (Oct. 2008), 1268-1280.

Kalyan Muthukumar and Gautam Doshi. 2001. Software pipelining of nested loops. Lecture Notes in Computer Science. 2027 (2001), 165–181.

Mohammed Fellahi, Albert Cohen and Sid Touati. 2007. Code-Size Conscious Pipelining of Imperfectly Nested Loops. In Proceedings of the 2007 workshop on memory performance: dealing with applications, systems and architecture (MEDEA'07). Brasov(Romania), 49-55.

Minhaj Ahmad Khan. 2011. Improving performance through deep value profiling and specialization with code transformation. J. Comp. Lang. Syst. Struct. 37, 4 (Oct. 2011), 193–203.

Mohammed Fellahi and Albert Cohen. 2009. Software Pipelining in Nested Loops with Prolog-Epilog Merging. Lecture Notes in Comp. Sc. 5409 (2009), 80-94.

Tobias Grosser, Albert Cohen , Paul H J Kelly , J. Ramanujam , Ponuswamy Sadayappan, and Sven Verdoolaege. 2013. Split tiling for GPUs: automatic parallelization using trapezoidal tiles. In proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units (GPGPU-6). Houston(Texas), 24-31.

Meilin Liu, Edwin Hsing-Mean Sha,Qingfeng Zhuge, Yi He and Meikang Qiu. 2011. Loop Distribution and Fusion with Timing and Code Size Optimization. J. Sign. Process. Syst. 62, 3 (March 2011), 325–340.

Apan Qasem and Ken Kennedy. 2008. Model-guided empirical tuning of loop fusion. Int. J. of High Perform. Syst. Arch. 1, 3(Dec. 2008), 183-198.

Duo Liu, Zili Shao, Meng Wang, Minyi Guo and Jingling Xue. 2009. Optimal Loop Parallelization for Maximizing Iteration-Level Parallelism. In Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems (CASES'09). Grenoble(France), 67-76.

Yi-Hsuan Lee , Cheng Chen. 2005. A two-level scheduling method: an effective parallelizing technique for uniform nested loops on a DSP multiprocessor. Journal of Systems and Software. volume 75, issue 1-2, page 155-170.

Chun jason Xue, Jingtong Hu, Zili Shao and Edwin Hsing-Mean Sha. 2010. Iterational Retiming with Partitioning: Loop Scheduling with Complete Memory Latency Hiding. J. ACM Trans. Emb. Comp. Syst. 9, 3 (Feb. 2010), No. 22.

Timothy William O'neil, and Edwin Hsing-Mean Sha. 2005. Combining extended retiming and unfolding for rate-optimal graph transformation. J. VLSI Sign. Proc. 39, 3 (March 2005), 273–293.

Louis-Noël Pouchet , Cédric Bastoul , Albert Cohen and John Cavazos. 2008. Iterative optimization in the polyhedral model: Part II, multidimensional time. In Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation (PLDI'08). Tucson(Arizona), 90-100.

Junjie Lai and André Seznec. 2011. TEG: GPU Performance Estimation Using a Timing Model. Research report N:7804, INRIA-IRISA.

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui. 2013. Execution Time and Code Size Optimization using Multidimensional Retiming and Loop Striping. 16th EUROMICRO Conference on Digital System Design (DSD'2013). Santander(Spain), 462-466.

Yaobin Wang; Zhiqin Liu; Huarong Chen; Xia Luo; Guotang Bi; Hong An. 2014. Exploring speculative procedure and loop level parallelism in SPLASH2. Int. J. of High Performance Systems Architecture. Vol. 5 No. 2. pp. 84-92.

Edson P. Ferlin, Heitor S. Lopes, Carlos R. Erig Lima, Mauricio Perretto. PRADA: a high-performance reconfigurable parallel architecture based on the dataflow model. Int. J. of High Performance Systems Architecture, 2011 Vol.3, No.1, pp.41 – 55.