HAL
open science

Execution Time and Code Size Optimization using Multidimensional Retiming and Loop Striping

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui

To cite this version:

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui. Execution Time and Code Size Optimization using Multidimensional Retiming and Loop Striping. EUROMICRO Conference on Digital System Design, Sep 2013, Santander, Spain. hal-01800763

HAL Id: hal-01800763

https://hal.science/hal-01800763

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 and Code Size Optimization using Multidimensional Retiming and Loop Striping

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*— Nested loops present the most critical sections in several embedded real-time applications. To achieve a higher performance, the design process employs an optimization technique in order to increase parallelism. However, the nested loop codes rise greatly in terms of parallelism level. Due to tight execution time constraints, each optimization technique produces implementations with an important code size. This criterion presents a limiting factor to implement the provided results in embedded real-time systems.**

In this paper, we propose a novel optimization approach that combines the delayed multidimensional retiming and loop striping techniques. It explores the solution space, which is composed by all parallelism cases proposed by both techniques, in order to provide the implementation that achieves the execution time constraint while uses a lower code size. In this context, we present the theory of combining both techniques. Then, we propose efficient algorithms that ensure selecting a set of parallelism transformations, based on their execution time and code size evolutions. The experimental results show that our optimization approach provides optimal solutions compared to those provided by applying only one technique. It achieves average improvements on the code size of 35.21% compared to the delayed multidimensional retiming and 16.38% compared to the loop striping.

***Keywords*— *design space exploration, optimization, nested loops, parallelism*.**

I. INTRODUCTION

A large group of embedded real-time applications integrate loop bodies. Their algorithmic structure implies a higher growth in the execution time. To provide an implementation that respects the timing constraints, the design process always consists of employing approaches which present a graph transformation flow. It ensures modeling applications through a data flow graph [3, 4, 12], and then modifying the initial graph to a one respecting the targeted constraints. The transformation steps are based on applying an optimization technique such as unrolling [15], extended retiming [14], etc. Several design methodologies combine optimization techniques [1, 2, 3, 4] which allow exploring a larger solution space and hence providing an optimal one compared to those provided by a single technique. The actual embedded real-time applications integrate increasingly the iterative and recursive processing such as 4D reconstruction, high-definition television, medical imaging, and remote sensing. In fact, the data flow graph mentioned above does not allow modeling both the iterative and recursive aspect of the loop bodies. They are always limited to model a single loop [3, 4], or are disable to model recursive data dependencies [12]. Thus, the enhancement of the previous approaches is very limited in the case of nested loops.

Within this framework, the acyclic data flow graph is extended to a Multidimensional Data Flow Graph (MDFG) that ensures an adequate representation of the nested iterative and recursive aspects [7]. Therefore, many optimization techniques are proposed in order to explore the MDFG potentialities. These techniques can be divided into two types: The first one ensures parallelizing iteration executions such as the loop striping [5, 6]. It requires duplicate instructions as well as the iterations are striped. The second type ensures parallelizing the execution of instructions belonging to the same iteration, whose all techniques are based on the multidimensional retiming principles [7, 8, 10, 11, 18]. Its optimization process implies adding instructions on both sides of the loop structures. We specify that the delayed multidimensional retiming technique [11, 13] allows providing optimal solutions in terms of execution time and code size then those provided by the previous techniques [7, 8, 18].

The loop striping and delayed multidimensional retiming techniques proceed to raise the parallelism level in order to decrease the execution time. Each technique aims at reducing one parameter of the execution time, where the delayed multidimensional retiming reduces the cycle period, while the loop striping minimizes the cycle number. Consequently, each technique is forced to increase parallelism level in order to decrease its aimed parameter. However, the more the parallelism level is high, the more the code size grows dramatically. Moreover, admitting that the application complexity keeps rising, the parallelism transformation implies a high code rise in the nested loops, due to data overlapping. As a result, even through the execution time constraint is achieved, each technique provides implementations with an important code size.

In this paper, we present a novel optimization approach that allows using both the delayed multidimensional retiming and the loop striping, in order to achieve the execution time constraint of the MDFG while using a lower code size. It

ensures exploring a large solution space by combining the optimization processes of both techniques. This work consists in predicting the execution time and the code size evolution in terms of each optimization transformation. The optimization approach explores the predicted values in order to select the set of transformations that ensure respecting the execution time while using a lower code size.

The rest of the paper is organized as follows. In section 2, we present the MDFG and we describe the delayed multidimensional retiming and the loop striping formalisms. In section 3, we present the principles and the basic concepts of our approach. The experimental results are presented in section 4, followed by the concluding remarks in section 5.

II. BASIC CONCEPTS

A. Multidimensional Data Flow Graph

The MDFG is a graphical representation allowing modeling n nested loops whose n number is called the graph dimension. It is formulated as $G = (V, E, d, t)$, where V represents the set of computation nodes, $E \subseteq V \times V$ represents the set of edges, $d(e_i)$ is a function representing a multidimensional delay between two nodes, and $t(v_j)$ is the computation time of the node v_j. An iteration corresponds to running all nodes in the MDFG once. An $d(e)$ edge delay is modeled by a vector with an n index such 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 an example, the wave digital filter algorithm in Fig. 1(b), composed by two nested loops, is modeled as the two-dimensional Data Flow Graph (2DFG) in Fig. 1(a). Each $e : v_i \rightarrow v_j$ edge is assigned by a $d(e) = (d.x, d.y)$ delay, where the "$d.x$" and "$d.y$" terms are in relation with the outermost loop and the innermost one, respectively. Taking as an example the instruction number 4 of the algorithm shown in Fig. 1(b), the $A(i,j)$ and $B(i,j)$ values are computed in the same iteration whether for the innermost or the outermost loop. For this purpose, the $e1: A \rightarrow B$ edge is labeled by the delay $d(e_1) = (0,0)$. The $d(e_4: B \rightarrow D) = (1,-1)$ means that if B is executed in the iteration i of the outermost loop, then D is executed in the iteration $i + 1$. Similarly to the innermost loop, D is executed in the previous iteration of the B execution.

All iterations of the algorithm in Fig. 1(b) are modeled into an acyclic graph, called the Cell Dependency Graph (CDG) shown in Fig. 2 (a). It is a cartesian space, whose the horizontal and the vertical axes show respectively the outermost and the innermost iteration indexes. Each iteration is modeled as a circle that is ranged in terms of its index values. Due to the space constraint, we are restricted to present only (6*6) iterations. The CDG illustrates explicitly the data dependencies between the iterations; e.i., The discontinuous arrows present the occurrences of the e_5 edge of the MDFG in Fig 1.(a), while the continuous ones present those of the e_4 edge. The CDG is used to identify an execution order of iterations, called the schedule vector s that verifies $s \times d(e) \geq 0$ for each $e \in E$ [7]. An MDFG $G = (V, E, d, t)$ is called realizable if there exists a schedule vector s with respect to G, and no cycle exists in its CDG. For instance, the iterations of the CDG in Fig. 2 (a) can be executed following the schedule vector $s = (1,0)$.

A path represents a node succession from v_i to v_j, noted as $p: v_i \xrightarrow{e_m} \dots \xrightarrow{e_n} v_j$. The delay vector of a path p is equal to $d(p) = \sum_{k=m}^{k=n} d(e_k)$. Accordingly, the computation time is equal to $t(p) = \sum_{k=i}^{k=j} t(v_k)$. A p_{cr} is defined as critical path if it has the maximal computation time among paths having zero delays $p_{cr} = max\{t(p), d(p) = 0\}$. An MDFG scheduling requires defining the critical path for the reason that its computation time presents the MDFG cycle period $C(G)$. In the case of the MDFG in Fig. 1(b), and assuming that $t(A) = t(D) = 1$ and $t(B) = t(C) = 2$, we identify two critical paths that are $p_1: D \rightarrow A \rightarrow B$ and $p_2: D \rightarrow A \rightarrow C$. Thus, the cycle period is equal to $C(G) = 4$. Fig. 2(b) shows the static scheduling of the algorithm in Fig. 1(a). Those nodes belonging to the same iteration, which are executed in a single cycle period, are modeled by the same pattern. In the case where $m = 20$ and $n = 20$, the wave digital filter requires 441 cycle periods and thus 1764 time units to be executed.

B. Delayed Multidimensional Retiming

The multidimensional retiming approach aims at reducing the cycle period. It proceeds to reduce the critical path size by redistributing the nodes in iteration, while preserving the loop structures of the original MDFG. As a result, it implies growing the parallelism in the nested loops. Several multidimensional retiming techniques [7, 8, 18] proceed to apply successively the multidimensional retiming

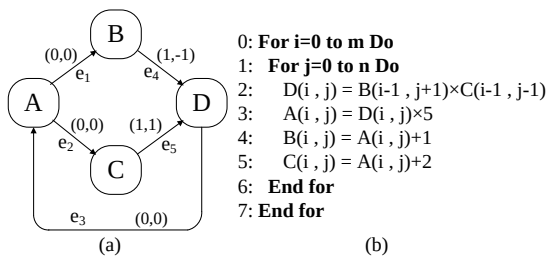

Fig. 1. (a) The MDFG; (b) the algorithm of the wave digital filter

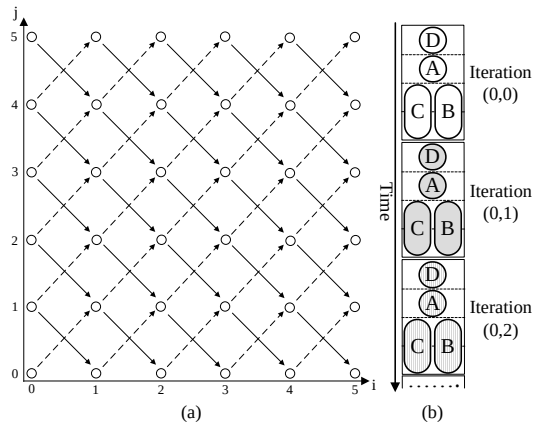

Fig. 2. (a) The cell dependency graph; (b) the static schedule of the wave digital filter

transformation until achieving a full parallelism. They provide an MDFG without any zero delay edge. However, the full parallelized MDFG implementation is characterized by an important code size. The work in [11,13] describes a technique called the "delayed multidimensional retiming" which allows achieving the minimal cycle period without achieving the full parallelism. It proceeds to retime the whole paths instead of retiming each node in the MDFG. Therefore, it provides implementations with the minimal execution time and code size compared to those provided by previous techniques.

The delayed multidimensional retiming technique applies a graphical transformation modeled as $R(p: u \rightarrow v) = (r_1, \dots, r_n)$, where p is a zero delay path $d(p) = (0, \dots, 0)$, $u, v \in V$ and n is the loop dimension. It consists in subtracting the $(r_1, \dots, r_n)$ delay from all the p incoming edges and adding them to all p outcoming ones. Each p path is moved from its original iteration as follows: For each r_k index, such as $1 \leq k \leq n$, the execution of the p path in the iteration i is moved to the iteration $i - r_k$. We show in Fig. 3(a) the MDFG after applying the multidimensional retiming $R(p: D \rightarrow A) = (0,1)$. It consists in modifying the MDFG in Fig. 1(a) by subtracting $(0,1)$ delay from e_4 and e_5, hence adding it to e_1 and e_2. It implies that the p path is executed one iteration before its originally one. In this context, the first occurrence of the p path should be executed before the first iteration. It involves adding to the retimed algorithm the p instructions upstream the innermost loop which is called the prologue. Likewise, the complementary instructions should be added downstream the same loop, which is called the epilogue. The code provided after the retiming is as shown in Fig. 3(b).

The inter-iteration data-dependencies of the retimed MDFG are showed in the CDG in Fig. 4(a), where the e_1 and e_2 edge are modeled by dotted arrows, and the e_4 and e_5 edges preserve the same patterns of the CDG in Fig. 2(a). This multidimensional retiming transformation provides a realizable MDFG whose the CDG can be executed following the scheduling vector $s = (3,1)$. Based on the algorithm in Fig. 3(b), there is not any data dependency between the p nodes and the other ones belonging to the same iteration. Consequently, it allows executing the p path in parallel with the B and C nodes, as shown in the static scheduling in Fig. 4(b). Thus, it allows reducing the cycle period $C(G_R)$ from 4 to 2 time units. In the case where $m = 20$ and $n = 20$, the delayed multidimensional retiming allows reducing the execution time from 1764 to 924 time units, despite increasing

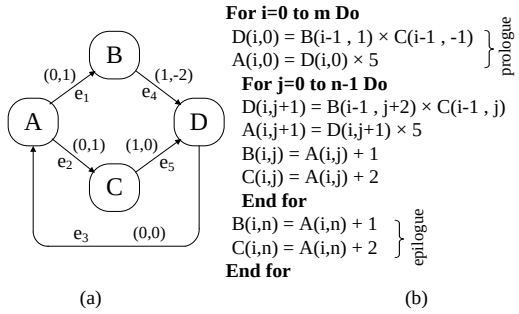

Fig. 3. (a) The MDFG; (b) the algorithm of the wave digital filter after multidimensional retiming

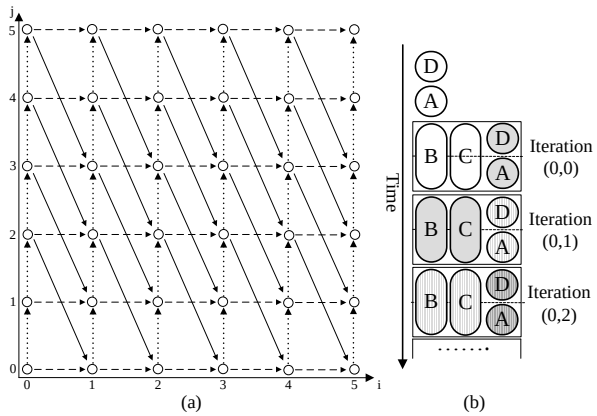

Fig. 4. (a) The cell dependency graph; (b) the static schedule of the wave digital filter after multidimensional retiming

the code size by 4 instructions.

C. Loop Striping

The loop striping aims at reducing the cycle number necessary to execute the nested loop. It proceeds to execute in parallel several iterations of the MDFG without modifying the original data dependency. This transformation requires that the striped iterations have not any data dependencies between them. The loop striping modifies the MDFG with respect to both parameters which are the factor f and the offset g. The first one presents the number of iterations that will be placed in the same strip. The second one defines the direction of the striping iteration; i.e., the iteration $(1,0)$ and iteration $(0, g)$ are placed in the same strip if f is equal to 2. The choice of the offset g is done in a way that there is not any data dependency between the striped iterations [6]. This criterion is the major difference between the unrolling and the loop striping. Taking the MDFG in Fig. 1(a) as an example, the loop striping technique allows striping the iterations such that $f = 2$ and $g = 2$, called $LS(f = 2, g = 2)$, whose strips are modeled by fat eclipses in the CDG of Fig. 5(a).

This technique implies collecting the striped iteration instructions in the innermost loop. The index evolution of the outermost loop is incremented by a step equal to the f factor. Moreover, in the case of a non-zero g offset value, the $(0, x)$ iteration, where $0 < x < f$, does not belong to any strip. Nevertheless, they must be executed before running the striped iterations. For this purpose, a loop structure is added to the algorithm upstream the innermost loop, which is called the prologue. Similarly, a second loop is added in upstream, which is called the epilogue. For example, the $LS(f = 2, g = 2)$ loop striping applied to the wave digital filter provides the algorithm shown in Fig. 6(b). The innermost loop instructions are duplicated compared to the original algorithm. The $(0,0)$ and $(0,1)$ iterations in the original algorithm are executed in the prologue, while the $(n, i + 1)$ and $(n - 1, i + 1)$ ones are executed in the epilogue.

Due to the duplication of instructions in each iteration unit, the provided MDFG integrates two occurrences of each node belonging to the original MDFG in Fig. 1(a), which is shown in Fig. 6(a). The loop striping preserves the data dependency

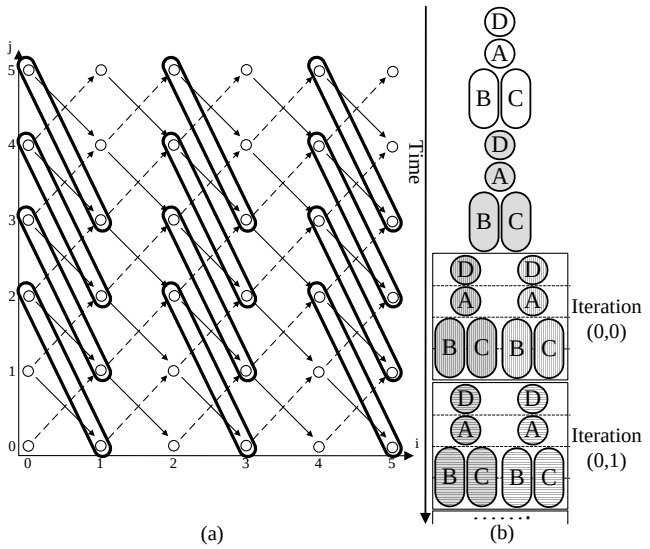

Fig. 5. (a) The cell dependency graph; (b) the static schedule of the wave digital filter after loop striping

inside each iteration. Therefore, it implies keeping zero-delays in the $D \rightarrow A$, $A \rightarrow B$ and $A \rightarrow C$ edges, such as the original graph. The non-zero delays are modified with respect to the CDG in Fig. 5(a). The static scheduling of the striped algorithm is shown in Fig. 5(b) which the iteration instructions in the original MDFG are modeled with the same pattern. In the case where $m = 20$ and $n = 20$, the loop striping allows reducing the cycle number from 441 to 291, hence reducing the execution time from 1764 to 1164 time units, in spite of raising the code size by 10 instructions.

III. MDFG Optimization Approach

A. Principles

Our work targets the design of embedded real-time applications, which require respecting an execution time constraint. Taking as an example the MDFG in Fig. 1(a), we aim to execute it in 550 time units. The delayed multidimensional retiming provides the final MDFG that requires 924 time units to be executed, as shown in the MDFG in Fig. 3(a). This technique is unable to respect the execution time constraint whatever the multidimensional retiming function R is. The loop striping allows increasing the f factor in order to reduce the execution time of the whole application. Respecting the g offset that is equal to 2, this technique should collect 5 iterations in the same strips in order to achieve the execution time constraint. However, the innermost loop is to include 20 instructions that are executed in parallel. In addition, the iterations which are not stripped are added to the prologue and epilogue in the final code. As a result, even through the loop striping respects the execution time constraint, it increasing the code size more than 8 times.

As we all know, the execution time is defined as the result of multiplying the cycle period and the cycle number parameters. Therefore, decreasing the execution time is similar to minimize either one of the parameters or both of them. Taking the retimed MDFG in Fig. 3(b) as an example, we deduce that both iterations (1,0) and (0,3) have not any data dependency between them. This condition is verified for both iterations: $(i+1, j)$ and $(i, j+3)$, for any $0 \leq i \leq m-3$ and $0 \leq \mathrm{j} \leq \mathrm{n}-1$. We proceed to apply the loop striping $LS(f = 2, g = 3)$ to the retimed MDFG whose the iterations are collected in strips as shown in the CDG in Fig. 7(a). This transformation implies duplicating the innermost loop instructions once, and adding both prologue and epilogue where each of them runs 3 innermost iterations. The provided MDFG shown in Fig. 8(a) is still realizable, whose execution can be done following the scheduling vector $s = (4,1)$. The correspondent algorithm is structured by 32 instructions as shown in Fig. 8(b). Based on the static schedule shown in Fig. 7(b), the cycle period has already fallen to $C(G) = 2$. Thus, the algorithm in Fig. 8(b) requires 250 cycle number, hence having 500 time units to the whole execution. So, we deduce that applying the delayed multidimensional retiming and the loop striping to the wave digital filter allows achieving the execution time constraint that the delayed multidimensional retiming has been unable to do. Moreover, it permits getting the aimed constraint by using a minimal code size then the one

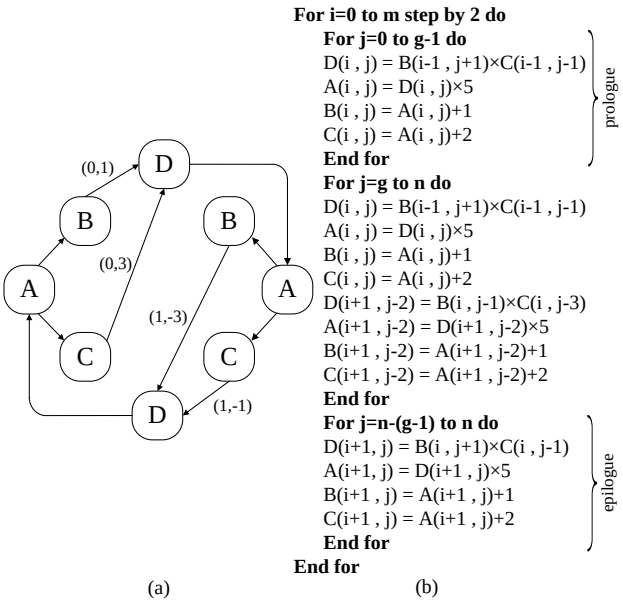

Fig. 6. (a) The MDFG; (b) the algorithm of the wave digital filter after loop striping

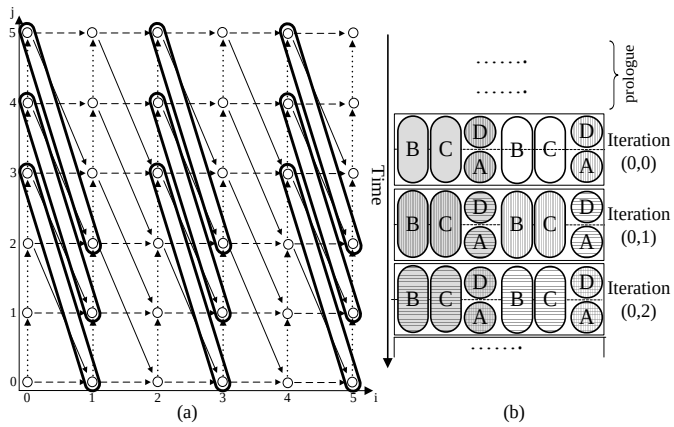

Fig. 7. (a) The cell dependency graph; (b) the static schedule of the wave digital filter after multidimensional retiming and loop striping

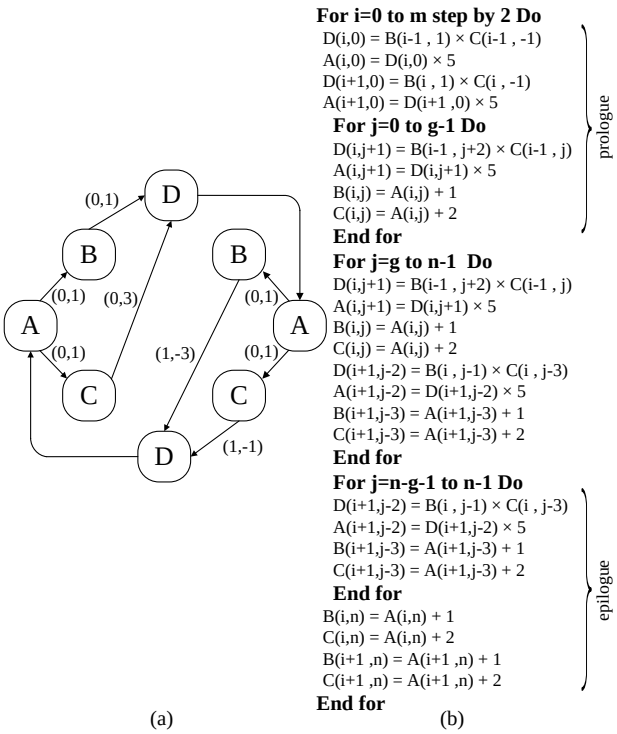

Fig. 8. (a) The MDFG; (b) the algorithm of the wave digital filter after multidimensional retiming and loop striping

provided by the loop striping with an enhancement of 41.06%.

In this context, we propose an optimization approach that allows using both the delayed multidimensional retiming and the loop striping, in order to achieve the execution time constraint while using a lower code size. The both techniques imply reducing the execution time in spite of rising the code size. None of the techniques is considered optimal compared to the other. For this purpose, our optimization approach ensuring exploring all the solution space offered by these two techniques in order to provide the one having the minimal code size among those explored. We note that the only work that combines iterational parallelism and instructional parallelism into an MDFG aims to only minimizing the iteration time average, without taking into account the timing impact of the added code [9].

We describe in the following sections the theory of choosing and applying the optimization technique in terms of the aimed constraint. Even if the formalisms correspond to a 2DFG, they can be directly extended to the general case.

B. Technique Order

We define in this section the theory of applying both the delayed multidimensional retiming and loop stripping techniques to the same MDFG. Let us focus on the MDFG structure before and after each optimization technique. In fact, the theory of both techniques consists in optimizing uniform nested loops [6, 7]. The loop striping implies duplicating the iterations, hence modifying the loop structure of the MDFG. We introduce in the following theorem the striped MDFG structure.

Theorem 3.1 *Given an MDFG $G = (V, E, d, t)$ and g is a striping offset of G. If $g \neq 0$ then the striped MDFG is a non-uniform nested loop.*

Proof. *$g \neq 0$ means that (1,0) and (0,g) iterations compose the first strip. The iterations (i, j) where $i < 1$ and $j < g - 1$ should be executed before the stripped iterations. Thus, $n \geq 1$ which implies adding a loop upstream the innermost one in order to execute the (i,j) iterations. Therefore, the stripped MDFG is a non-uniform nested loop.*

According to the example in Fig. 6(b), the loop striping implies duplicating 3 times the innermost loop. So, for a striped MDFG, a multidimensional retiming is limited to optimize only one of the innermost loops, which minimizes the multidimensional retiming improvement.

In contrast, a multidimensional retiming transformation provides an MDFG preserving the loop structure, whatever le retiming function is. Figure 3 shows that the retimed algorithm contains the same loops even though the retiming overheads the instructions in the two loop directions. We study in theorem 3.2 the ability of applying the loop striping to a retimed MDFG.

Theorem 3.2 *Given an MDFG $G = (V, E, d, t)$, a legal multidimensional retiming $R = (i, j)$ of G and the retimed MDFG G_R by R, and if g is a striping offset of G then there exists a striping offset g' of G_R.*

Proof. *g is a striping offset of G which means that there exists $b = (x, 1)$ and that $b \times d > 0$ for each $d \in E$. This condition proves that b is a scheduling vector of G, and therefore b and R are orthogonal [7]. R is a legal multidimensional retiming of G, which means that there exists a scheduling vector $s = (x, y)$ and that $d_R \times s > 0$ for each $d_R \in E_R$, where $G_R = (V, E, d, t)$, which verifies the existence of a striping offset g' of G_R.*

This theorem proves that we can apply the loop striping to a retimed MDFG independently from the optimization parameters of both techniques. Based on theorem 3.1 and theorem 3.2, the solutions can be provided either by applying the loop striping, the multidimensional retiming or the multidimensional retiming followed by the loop striping. In fact, the delayed multidimensional retiming proceeds to applying several multidimensional retiming transformations to the same MDFG. Similarly, the final MDFG of the delayed multidimensional retiming technique can always be striped, as introduced in the following lemma.

Lemma 3.1 *Given an MDFG $G = (V, E, d, t)$, a legal multidimensional retiming $R = (i, j)$ of G and the MDFG G_{DR} after the delayed multidimensional retiming technique using R, and if g is a striping offset of G, then there exists a striping offset g_{DR} of G_{DR}.*

Proof. *The delayed multidimensional retiming applies n times the a multidimensional retiming function R to the same MDFG until achieving the minimal cycle period. Thus, the lemma is checked immediately by repeating n times the verification of theorem 3.2.*

Our work aims at achieving an execution time constraint. When an algorithm requires appealing both techniques, we have to apply the delayed multidimensional retiming first. The loop striping will be the last graphical transformation in order to achieve the targeted constraint.

The lemma 3.1 offers several cases of multidimensional retiming, from applying the first transformation to applying all ones purposed by the delayed technique [11], before the loop striping. Thus, the optimization approach selects the optimization parameters of the multidimensional retiming R and the loop striping LS. Then, it chooses the most profitable one as described in paragraph C, in order to apply it to the original MDFG. This provided MDFG presents the subject of the next test. This step is done iteratively until achieving the execution time constraint T_C as modeled in the flow of Fig. 9.

C. Technique Choice

This work aims to provide an implementation that respects an execution time constraint while using a minimal code size. Thus, the selection of a parallelism is based on its contribution in terms of execution time and code size. Therefore, after selecting the parameters of each technique which are the function R of the retiming technique, and the factor f and the offset g of the loop striping one, our optimization approach proceeds to predict the execution time enhancement and the code size rise of each technique result in terms of the selected parameters, which the prediction steps are detailed in the paragraph D. Next, it chooses an optimization technique which is based on the predicted values.

On the one hand, choosing the optimization transformation that provides the maximal execution time may lead to an implementation with a high code size. As cited in section 3.1, the loop striping achieves the execution time constraint while the multidimensional retiming is unable to do that. Yet, the result purposed by the loop striping requires a greater code size than those provided by applying the multidimensional retiming as a first optimization, as shown in Fig. 8(b). On the other hand, the approach cannot choose the transformation that requires the minimal code. It can have a negligible contribution to the execution time which implies a compensating effect. That is why, the choice should be based on both execution time and code size contributions. In this context, we purpose to generate a cost function as done in [12], as indicated in (1).

$$f_{\text{cost}} = \Delta C / \Delta T \tag{1}$$

where ΔC represents the code size rising and ΔT represents the execution time reduction. A cost function is computed for each eventual optimization among the multidimensional retiming and loop striping. Then, our approach chooses the optimization presenting the small cost, hence applying it to the current MDFG.

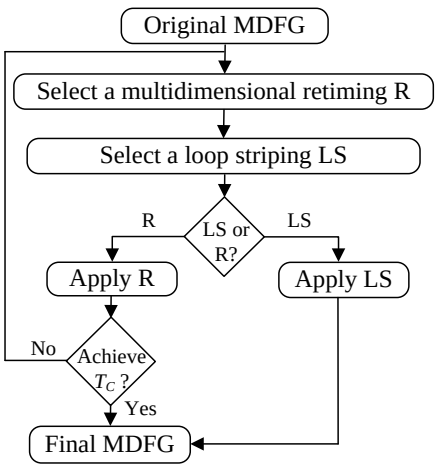

Fig. 9. The optimization approach flow

The whole steps of our proposed optimization approach are described in algorithm 1. It starts by predicting the execution time T of the MDFG. Based on [16, 17], this term consists of estimating two parameters whose are the computation time and the communication time. The first one is predicted by multiplying the cycle period and cycle number of the whole application. The second one presents the time spent on data scheduling and memory access. It depends of the processor technology and the memory architecture. In this work, we restrict on predicting only the computation time. Therefore, our approach computes the cycle period Clk, the cycle number N_{Clk} and the minimal cycle period Clk_{min} that can be achieved [11]. Then, it runs an iterative structure that selects a loop striping $LS = (f, g)$ and a multidimensional retiming $R = (x, y)$. Afterward, it computes both technique costs and chooses that presents the minimal ones. If the minimal cycle period is achieved, the approach applies directly the selected LS loop striping. The approach stops running when the execution time constraint T_C is achieved.

D. Cost Function Computing

We describe in this section our process to predict the optimization costs. The first term, which is the ΔT execution time reduction, presents the difference between the execution times before and after the optimization transformation. Admitting that both two values are used into the approach progress, each one of them must be predicted for each eventual optimization. For the second term, which is the ΔC code rising, the optimization approach runs without requiring the code size values in any step of the optimization process. However, the added prologue and epilogue after a multidimensional retiming can be computed in terms of the retiming function R. Similarly, we can define the iterations

Algorithm 1. The optimization approach

Inputs: MDFG $G = (V, E, d, t)$, execution time constraint T_C, the iteration bounds of the outermost loop m and the innermost loop n
Outputs : the optimized MDFG

```
Begin
/* Compute the execution time T and the minimal cycle period Clk_min */
Compute the cycle period Clk = max{t(p), d(p) = 0} where p is a path of G
N_Clk ← m × n
T ← Clk × N_Clk
Compute the minimal cycle period Clk_min = max{t(v), v ∈ V}
/* Choose the optimization technique */
While (T > T_C) do
    Select a multidimensional retiming function R = (x, y)
    Compute the retiming cost f_R (as described in algorithm 2)
    Select a loop striping LS = (f, g)
    Compute the loop striping cost f_ls (as described in algorithm 3)
    If (Clk > Clk_min) and (f_R < f_LS) then
        Apply the multidimensional retiming R
        T ← T_R
    Else
        Apply the loop striping LS
        T ← T_LS
    End if
End While
End
```

that are not striped in terms of the f factor and the g offset. Thus, we proceed to predict only the added code size ΔC using the optimization parameters of each technique.

In the case of the multidimensional retiming, the optimization modifies the cycle period Clk_R, the cycle number N_{Clk_R} and the code size, where all must be predicted. If the retiming function affects only the innermost loop, the prologue and epilogue will be added only upstream and downstream this last loop, which consists in adding the innermost loop instructions C_{in} duplicated by the second index $|y|$ of the function R. Contrarily, the prologue and the epilogue are added to each loop structure. The multidimensional retiming cost is predicted as described in algorithm 2.

In the case of the loop striping, the optimization consists in duplicating f times the innermost iteration instructions and adding the non-striped iteration as prologue and epilogue. Thus, this technique modifies only the cycle number $N_{Clk_{LS}}$ and the code size. We start by defining the number of iterations $iter$ that are run in each part of prologue and epilogue. Then, we compute the cycle number $N_{Clk_{LS}}$ while taking into account the new iteration bound values of both loops. The ΔC code rise is the addition of those duplicated in the innermost loop and those added in the prologue and epilogue. The cost of a loop striping is computed as described in algorithm 3.

Our optimization approach always provides an implementation with a minimal code size whose efficiency is proved in theorem 3.3.

THEOREM 3.3. *Given an MDFG $G = (V, E, d, t)$, the innermost iteration instructions number C_{in} and the maximal iterations bounds I, the optimization approach provides the final MDFG in at most $O(V^2 + E + I \times C_{in})$ times.*

Proof. *The delayed multidimensional retiming requires $O(E + V^2)$ times. Thereafter, the loop striping requires $O(E)$ to select the f factor and the g offset parameters and $O(E + f \times C_{in})$ to generate the final code [6]. Moreover, the cost functions is run on $O(V)$ in the case of the multidimensional retiming and $O(C_{in})$ for the loop striping which are associated to each optimization selection. Admitting that the code generation after the loop striping is executed once and $f < I$, our optimization approach requires only $O(V^2 + E + I \times C_{in})$ times to be performed.*

Algorithm 2. Cost function of the delayed multidimensional retiming

Inputs: MDFG $G = (V, E, d, t)$, retiming function $R = (x, y)$, the current cycle number N_{Clk}
Outputs : multidimensional retiming cost f_R
Begin
Define the paths to retime [11]
/ Compute the cycle period of the multidimensional retiming*/*
Extract the critical paths P_{ch} after retiming
Compute the cycle period after retiming $Clk_R = max\{t(p), d(p) = 0\}$ where $p \in P_{ch}$
Compute the code size of the innermost loop C_{in}
/ Compute the cycle number and the added code size after the multidimensional retiming */*
If $((x = 0)\ and\ (y \neq 0))$ **then**
$N_{Clk_R} \leftarrow (N_{Clk} \times (n + |y|))\ /\ n$
$\Delta C \leftarrow C_{in} \times |y|$
Else
$N_{Clk_R} \leftarrow 2 \times |x| + (N_{Clk} \times (m + |x|) \times (n + |y|))/(m \times n)$
$\Delta C \leftarrow C_{in} \times (|x| + |y|)$
End if
/ Compute the cost function of the multidimensional retiming */*
$f_R \leftarrow \Delta C/(T - (Clk_R \times N_{Clk_R}))$
End

Algorithm 3. Cost function of the loop striping

Inputs: MDFG $G = (V, E, d, t)$, loop striping vector (f, g), the current cycle period Clk
Outputs : loop striping cost f_{LS}
Begin
/ Compute the cycle number after the loop striping */*
$iter \leftarrow 0$
For i from 0 to $(f - 2)$ **do**
  For j from 0 to $(g - 1)$ **do**
  $iter \leftarrow iter + 1$
  End for
End for
$N_{Clk_{LS}} \leftarrow \left\lceil \frac{m}{f} \right\rceil \times (2 \times iter + (n - g))$
/ Compute the added code size after the loop striping */*
Compute the instruction code size of the innermost loop C_{in}
$\Delta C \leftarrow C_{in} \times iter + C_{in} \times f$
/ Compute the cost function of the loop striping */*
$f_{LS} \leftarrow \frac{\Delta C}{(T - (Clk \times N_{Clk_{LS}}))}$
End

IV. EXPERIMENTAL RESULTS

To validate our approach contribution, we have compared its provided results to those provided by the loop striping or the delayed multidimensional retiming. This step consists in applying the three optimization processes to the same set of MDFGs. The applications chosen in our experiments are the Wave Digital Filter (WDF), the Infinite Impulse Response Filter (IIRF) and the Walsh-Fourier Transform (WFT), whose MDFGs are composed by the adder and multiplier nodes. Recognizing that the multidimensional retiming [11, 13] depends on the execution-time nodes, we proceed to model each MDFG with different node execution times, as shown in the second column of TABLE.I, where the adder and multiplier execution times are respectively indicated by t_A and t_M. To verify the solution space exploration, we require different execution time constraints T_c for each graph which are indicated in the third column. Then, we apply the three optimization techniques with the aim of achieving each required constraint. The execution time T and the code size C values of each technique are shown in the three last columns of TABLE.I. We indicate that the bold values mean that the used technique does not allow achieving the execution time constraint.

The "improve" row presents the benefit in terms of the code size of the results provided by our techniques compared to those provided by the multidimensional retiming and the loop striping, which accounts for an average improvement of 35.21% compared to the multidimensional retiming technique and 16.38% compared to the loop striping technique.

TABLE I. THE EXECUTION TIME AND CODE SIZE VALUES IN TERMS OF OPTIMIZATION TECHNIQUES

MDFG	(t_A, t_M)	T_C	Multidim. retiming		Loop striping		Optim. approach	
			T	C	T	C	T	C
WDF	(1,1)	800	504	16	693	12	639	12
		550	504	16	525	28	504	16
		450	**504**	**504**	411	36	314	28
	(1,2)	1500	1449	12	1155	12	1155	12
		1200	1008	16	1155	12	1155	12
		700	**1008**	**16**	685	36	628	28
	(2,1)	1100	924	12	1004	24	924	12
		700	**924**	**12**	624	32	534	24
		550	**924**	**12**	548	36	534	24
IIRF	(1,1)	750	720	130	525	70	525	70
		450	316	402	395	102	436	92
		380	316	402	315	134	284	108
	(1,2)	1200	844	52	1050	70	844	52
		900	720	130	790	102	872	92
		650	516	149	615	134	568	108
	(2,1)	1200	1157	130	1050	70	1050	70
		900	632	402	790	102	872	92
		650	632	402	615	134	568	108
WFT	(1,1)	300	242	26	165	24	165	24
		150	138	77	147	32	120	28
		110	**138**	**77**	**147**	**32**	110	42
	(1,2)	400	202	16	220	24	202	16
		300	202	16	196	32	202	16
		200	**202**	**16**	196	32	120	28
	(2,1)	400	357	26	275	24	275	24
		300	276	77	275	24	275	24
		250	**276**	**77**	245	32	240	28
Improve (%)			35.21		16.38			

V. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed an optimization approach that explores the solution space offered by the multidimensional retiming and the loop striping techniques, in order to select an implementation that achieves the execution time constraint while using a lower code size. The improvement averages in the experimental results prove that our approach provides optimal solutions compared to each technique used separately. The approach efficiency proved in theorem 3.3 allows implementing our approach in a software tool dedicated to design embedded real-time applications.

In our future works, we aim to concretize our approach in a software tool that allows the designer to model a MDFG and to fixe an execution time constraint in order to provide automatically the optimized implementation. Moreover, we aim for developing the execution time and code size prediction by taking into account the technological parameters such as the memory access and the data routing, etc. Also, we will be interested in enhancing the function cost by offering the ability of weighting between the execution time and the code size.

REFERENCES

[1] Q. Zhuge, Z. Shao, B. Xiao, and E.H.-M. Sha, “Design space minimization with timing and code size optimization for embedded DSP”, CODES+ISSS, California (USA), pp. 144-149, October 2003.

[2] Q. Zhuge, B. Xiao, Z. Shao, and E.H.-M. Sha, C. Chantrapornchai, “Optimal code size reduction for software-pipelined and unfolded loops”, ISSS, pp. 144-149, 2002.

[3] T.W. O’neil, and E.H.-M. Sha, “Combining extended retiming and unfolding for rate-optimal graph transformation”, J. of VLSI Sign. Process., vol. 39, iss. 3, pp: 273–293, March 2005.

[4] Q. Zhuge, C. Xue, Z. Shao, M. Liu, M. Qiu, and E.H.-M. Sha, “Design optimization and space minimization considering timing and code size via retiming and unfolding”, J. Microproc. Microsyst., vol. 30,pp. 173–183, 2006.

[5] C. Xue, Z. Shao, M. Liu, and E.H.-M. Sha, “Iterational retiming: maximize iteration-level parallelism for nested loops”, CODES+ISSS, pp. 309-314, 2005.

[6] C. Xue, E.H.-M. Sha, “Maximize parallelism minimize overhead for nested loops via loop Striping”, J. of VLSI Sig. Proc., vol. 47, pp. 153–167, December 2006.

[7] 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, pp. 1150-1163, November 1996.

[8] M. Sheliga, N. L. Passos, and E.H.-M. Sha. “Fully parallel hardware/software codesign for multidimensional DSP applications”, CODES, Pennsylvania (USA), pp. 18-20, March 1996.

[9] C. J. Xue, Z. Shao, M. liu, M. K. Qiu, E.H.-M. Sha, “Optimizing parallelism for nested loops with iterational and instructional retiming”, J. Embed. Comput., vol. 3, iss.1, pp. 29-37, January 2009.

[10] Y. Elloumi, M.Akil, and M.H. Bedoui, “Timing and code size optimization on achieving full parallelism in uniform nested loop”, J. Comp., vol. 3, iss. 7, Jully 2011.

[11] Y. Elloumi, M.Akil, and M.H. Bedoui, “Execution time optimization using delayed multidimensional retiming”, IEEE/ACM DSRT, Dublin (Ireland),pp. 177-184, October 2012.

[12] L. Kaouane, M. Akil,T. Grandpierre, and Y. Sorel, “ A methodology to implement real-time applications onto reconfigurable circuits”, J. Supercomp., vol. 30, iss. 3, , pp. 283-301, December 2004.

[13] Y. Elloumi, M.Akil, and M.H. Bedoui, “Achieving Minimal Cycle Period with Delayed Multidimensional Retiming”, J. App. Soft Comp., unpublished.

[14] T. O'Neil, S. Tongsima, and E.H.-M. Sha, “Extended retiming: Optimal scheduling via a graph-theoretical approach,” ICASSP, vol. 4, pp. 2001–2004, March 1999.

[15] K.K. Parhi, and D.G. Messerschmitt, “Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding”, IEEE Transact. Comp., vol. 40, iss. 2, pp. 178-195, February 1991.

[16] O. Lobachev, M. Guthe, and R. Loogen, “Estimating parallel performance”, J. Par. Dist. Comp., January 2013, (in press).

[17] G. Romanazzi, P.K. Jimack, and C.E. Goodyer, “Reliable performance prediction for multigrid software on distributed memory systems”, J. Adv. Engin. Softw., vol. 42, pp. 247–258, May 2011.

[18] 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, pp. 351-363, october 2008.