WCET Nested-Loop Minimization in Terms of Instruction-Level-Parallelism

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui

▶ **To cite this version:**

Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui. WCET Nested-Loop Minimization in Terms of Instruction-Level-Parallelism. International Conference on High Performance Computing & Simulation (HPCS), Jul 2015, Amsterdam, Netherlands. hal-01796771

HAL Id: hal-01796771
https://hal.science/hal-01796771v1

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.

WCET Nested-Loop Minimization in Terms of Instruction-Level-Parallelism

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: {yaroub.elloumi, 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
Email: medhedi.bedoui@fmm.rnu.tn

***Abstract*—Several high-performance applications integrate loop bodies which represent the most critical sections. This aspect brings two challenges. First, the Worst Case Execution Time (WCET) must be determined in order to define the nested loop timing behaviour. The second challenge consists in raising the parallelism-level to enhance the performance. In particular, the Multidimensional Retiming (MR) is an important optimization approach that offers several instruction-level-parallelism solutions. Despite the fact that full parallelism allows achieving the optimal WCET, it leads to a high growth in processing cores, which is inadequate to embedded real-time implementations**.

The main idea of this paper consists in driving the parallelism-level rise in terms of WCET development. First, the MR parameters that correspond to the nested loops are extracted. Thereafter, the WCET is formulated in term of the parallelism level rise. Then, an optimization heuristic is proposed which identifies the parallelism level that allows respecting the WCET constraint. Our experiments indicate the WCET prediction is accurate within an error rate of 8.54%. Secondly, the optimization heuristic implementations show an average improvement on number of cores of 27.18% compared to full parallel ones.

Keywords— HW/SW codesign, design space exploration, optimization, WCET, nested loops, parallelism.

I. INTRODUCTION

The loop bodies compose a large group of embedded real-time applications. They imply a higher rise in the execution time, which generally results in overtaking the real-time properties. Therefore, the performance is handled through all the design levels in terms of analysis and optimization, to ensure a safe final implementation.

Several WCET estimation approaches are proposed, which are generally classified in two kinds [5]. The first one consists in running test codes and estimating the WCET, based on the measured performances. However, those approaches are inefficient in the case of complex codes. The other approaches are based on a WCET static analysis. They model the implementation code in order to identify the data flow and the control flow parameters and use them on an equation system. The WCET static analyses are integrated on a general workflow that intercepts both code and micro-architecture parameters to provide a safe WCET [1,2]. Several workflows are implemented in efficient software tools and compilers for real-time applications [4, 5, 19].

Actually, the real-time application complexity is increased due to an intensive utilization of nested loops. Moreover, parallel architecture implies different parallel scheduling, hence different timing behaviours. In this context, many works have focused on analysing the nested loop structures in order to purpose adequate equation systems for WCET estimation [6, 7, 8], which are in the scope of our work. The main idea is to explore the nested loop model to define the loop iteration numbers, the loop dependencies and the critical task inter-loops, etc. Some works have focused on specific loop bodies such as the one containing unfeasible paths like "if …else" instruction and "do … while" loop [2, 6], which are not considered in the paper.

Elsewhere, several parallelism approaches have led to enhance the nested loop performances. They generally represent the data flow and the control flow in a formal model such as the polyhedral model [9, 10] and the Multidimensional Data Flow Graph (MDFG) [11, 12]. Thereafter, they transform the model to parallelize the processing. The MDFG is an acyclic data flow graph that ensures an adequate representation of nested loops. It allows modelling the loop-carried dependencies across several loops. It is distinguished by a detailed instruction granularity compared to the polyhedral model. A lot of optimization approaches are proposed to parallelize the MDFG. The Multidimensional Retiming (MR) is a theoretical approach that guarantees parallelizing instructions inside nested loops. All MR techniques aim to iteratively increasing the instruction-level-parallelism until achieving a full parallel implementation, hence the minimal WCET [12, 13, 14]. However, the higher the parallelism level is, the more intensively the processing core number grows. Moreover, admitting that the application complexity keeps growing, the parallelism transformation implies a data overhead which requires a large cache memory size. As a result, even the optimal WCET is achieved, the MR provides implementations with important material resources.

In fact, the MR approach offers a significant solution space with a corresponding different parallelism level. Consequently, it is sufficient to apply a partial parallelism that enables attaining the WCET constraint, instead of applying a

full parallelism. However, it is inefficient to extract each solution and then define its WCET, due to the nested loop complexity. The works described in [17, 18] partially explore the solution space to provide a safe implementation, but they do not give optimal solutions. Few works have been interested in analyzing the WCET development in terms of parallelism techniques. As described in [15], the work combines the theory of both WCET model and loop unrolling to enhance the performance. Furthermore, an instruction-level-parallelism controlled by the WCET development is suggested for super-scalar processors in [16]. However, both works were proposed for specific nested loops without modeling general loop-carried dependencies.

In this paper, we aim to drive the instruction-level-parallelism rising of nested loops by the WCET development. First of all, The WCET equation system is formulated in terms of MR parameters. Then, the parallelism level is increased iteratively until respecting the WCET constraint.

The rest of the paper is organized as follows. In section II, we present the basic concepts of modelling nested loops with MDFG, parallelizing the loop body with MR and predicting nested loop WCET. In section III, we present the theory of the WCET estimation and the optimization heuristic that drives MR in terms of WCET. The experimental results are given in section IV, followed by the concluding remarks in section V.

II. BASIC CONCEPTS

A. 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. Indeed, 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 [25] 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. 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. Therefore, 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". The critical paths p_{cr} 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 an Iteration Time $IT(G)$, where $IT(G) \geq t(p_{cr})$. For example, the critical paths is structured as $p: A1 \rightarrow A2 \rightarrow D1 \rightarrow D2$. Assuming that $t(A1) = t(A2) = 1\ time\ units$ $(t.u.)$ and $t(D1) = t(D2) = 2\ t.u.$, the Jacobi algorithm IT is equal to $6\ t.u.$. The theoretical schedule is represented in Fig. 2 where the nodes belonging to the same iteration are modelled by the same pattern which can be executed using one core.

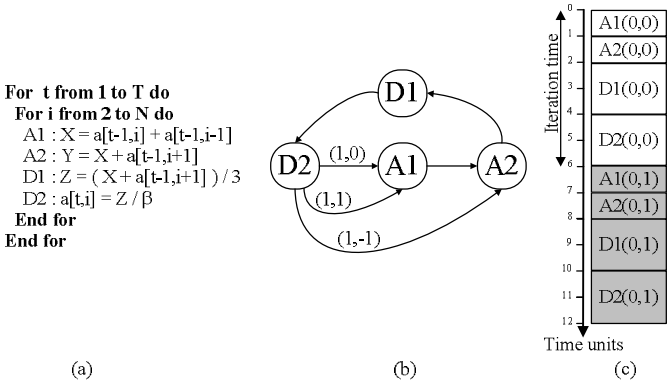

Fig.1. The Jacobi algorithm : (a) code, (b) MDFG, (c) static schedule

B. Multidimensional retiming

The MR is an instruction-level-parallelism approach of nested loops. It reduces the IT by decreasing nodes in critical paths. With this objective, it shifts nodes from their original iteration in order to execute them in parallel with other nodes. For a retimed node, modifying its belonging iteration consists in modifying their edge delays with respect to a MR function. For the MR function $r(u) = (r_1, \dots, r_n)$, the execution of the node u in the iteration i is moved to the iteration $i - r_k$ [12]. Thus, the MR is modelled as a graphical transformation that modifies the MDFG delays, while preserving the functional behaviour of the initial MDFG [12,13]. All MR techniques aim to achieve the IT_{min}. The incremental and chained MR techniques [12] apply the MR successively to each critical path nodes until having a full parallelism; i.e., all instructions in the same iterations are executed in parallel. However, the provided implementation requires processing cores as more as the nodes are.

Other MR techniques lead to schedule an MDFG with a IT_{min}, without getting a full parallelism [12, 14, 27]. Retiming the whole path is carried out, instead of a node only, in order

to minimize the processing cores when enhancing the performance. The delayed MR technique [14, 27] proposes a theoretical approach to select and retime paths. It defines two terms to reflect the timing and data dependency characteristics of each path in the MDFG: the $D(u,v)$ which represents the minimum delay between the 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 $D(u,v) = (0,\dots,0)$; else, $D(u,v) \neq (0,\dots,0)$), and the $T(u,v)$ which defines the maximum execution time among the zero delay path connecting u and v, as described in Equation (2).

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

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

Their values are ranged respectively in two matrices called D and T with $V \times V$ size, such as V in the node set. Each $p: u \to v$ path is indexed by the cell with the line u and the column v. Taking the Jacobi algorithm as an example, the MDFG in Fig. 1(b) is composed by four nodes, hence the 4×4 matrix dimension whose D and T matrices are respectively illustrated in Table I and Table II.

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

Thus, achieving the IT_{min} is synonymous to providing an MDFG whose $D(u,v) = 0$ and $T(u,v) > IT_{min}$. Therefore, the delayed MR technique explores both matrices to define the sub-critical paths with respect to the previous condition. For the Jacobi algorithm, the delayed MR selects the MR function $r = (0,1)$ to software pipeline the $P1: A1 \to A2$ path, which the retimed Jacobi algorithm is shown in Figure 2(a). Each i^{th} occurrence of $P1$ path is shifted up and executed in the previous iteration of the innermost loop, where $1 \leq i \leq n$. The first $P1$ 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 $P1$ paths, belonging to $(i,0)$ iteration, 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. This transformation permits executing any P1 path in the innermost loop in parallel with other nodes, as shown in the static schedule of Fig. 2(c). Accordingly, the IT is reduced from 6 to 4 $.u.$. In addition, the innermost loop is iterated $(N-1)$ times, though requiring two cores to execute the implementation.

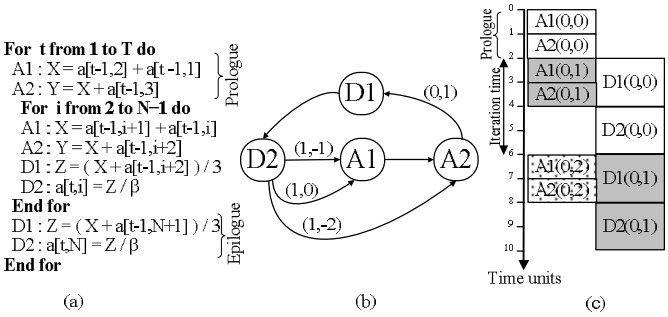

Fig.2. The Jacobi algorithm after r(A1 → A2) = (0, 1) : (a) code, (b) MDFG, (c) static schedule

the second MR function is applied to the $D1$ node. The final MDFG is scheduled with the $IT_{min} = 2\ t.u.$ using three processing cores as shown in the Fig.3.

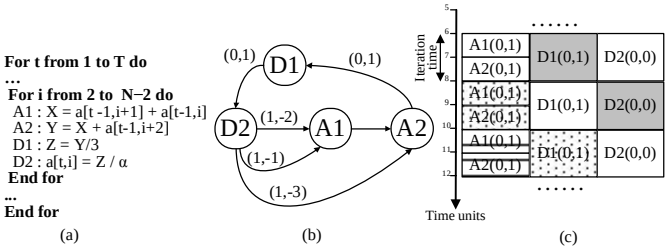

Fig. 3. The Jacobi algorithm provided by the delayed MR : (a) code, (b) MDFG, (c) static schedule

C. Nested loop WCET

The WCET analysis involves defining all parameters to quantify the execution time upper bound. For parallel architecture, the scheduling ensures distributing the processing into several cores, hence its running times. In addition, the growth in the grid dimension and the extension of the memory hierarchy implies an additional timing component that corresponds to the access memories. Therefore, several works [4, 26] split the $WCET$ into a Worse Case Running Time ($WCRT$) and a Worse Case Stall Time ($WCST$) as indicated in Equation 3.

$$WCET = WCRT + WCST \quad (3)$$

The $WCRT$ static analysis explores the software parameters of the loop nests. The equation system is always formulated using the computation time t_i of all i tasks and their instance numbers c_i [1, 4, 5, 21, 22, 23]. Each task is defined by exploring the control flow in order to define the critical path instructions. Therefore, its execution time t_i is determined by associating a timing parameter for each computation instruction. The c_i instance number is computed based on the iteration loop bounds that are identified from the code. In the case of parallel processing, the $WCRT$ consists in selecting the task executed in parallel to identify the ones having the maximal computation time [3, 20, 24]. Thus, the whole $WCRT$ value is generally the maximal value of multiplying t_i and c_i parameters, as indicated in Equation 4. In the case of nested loops, Equation 4 is enabling its application by considering each loop as a task in its outermost one.

$$WCRT = max \sum_i t_i \times c_i \quad (4)$$

The $WCST$ represents the necessary time to schedule data to parallel cores, which is in terms of the thread number. Accordingly, the more the threads are, the higher the stall time is. Moreover, nested loops iterate the execution of the parallel processing which leads to a similar stall time rise, resulting in computing the $WCST$ as indicated in Equation 5 [4, 20].

$$WCST = N \times (T - 1) \times C \quad (5)$$

Where N is the iteration number, T is the thread number and C is timing parameter that is experimentally defined as in [4, 20].

III. WCET DEVELOPEMENT IN TERMS OF MULTIDIMENSIONAL RETIMING

A. *Approach principle*

The main idea of this paper consists in driving the instruction-parallelism-level rise in terms of WCET development. For this purpose, our approach requires an efficient WCET estimation.

As indicated in the last paragraph, the WCRT is expressed in terms of computation times and instance numbers of tasks. For the MDFG, those tasks corresponding to the instructions belong either to the innermost loop or the prologue/epilogue codes. Instance numbers are to be computed directly from their outermost loop bounds, irrespective of their number. As indicated in section II, to obtain iteration time, the identification of the critical paths is vital. For the WCST, it depends of the thread number that ensures implementing any code block, which is similar to the parallelism level. In this objective, the MDFG is to be swept in order to compute the maximal node count executed in parallel.

Elsewhere, The MR iteratively increases the parallelism level by software pipelining nodes. Accordingly, each MR function consists in decreasing the loop bounds, modifying the data paths either inside or both side loops, and increasing the parallelism level. Moreover, those transformations rise with respect to the applied MR function number. In this objective, the approach formulates the loop bound, the iteration time and the parallelism level developments into equations in terms of MR function and its application numbers. Thereafter, those equations are employed to estimate the WCET nested loops.

Based on the MR modification and equations 4 and 5, the MR leads to reduce the WCET despite increasing the parallelism level, hence the core rise. Thus, the approach proceeds iteratively to estimate the WCET and raises the parallelism level using the MR until achieving the WCET constraint or the IT_{min}.

Within this framework, figure 4 illustrates a step-by-step approach. It starts by providing an optimal MR function r and defining the IT_{min} [12, 13, 14]. Then, it sweeps the MDFG to identify the maximal number of MR that can be applied and their corresponding data paths that will be shifted, as described in section B. In section C, those data paths are explored to compute the iteration times IT_R and the parallelism levels PL_R, which correspond to each MR order. The development of the iteration loop bounds LB is formulated in terms of MR retiming function in section D. As a consequence, the WCET prediction of the whole nested loops is then detailed in section E. Finally, an optimization heuristic is proposed in section F, which drives the instruction-level-parallelism in terms of WCET development.

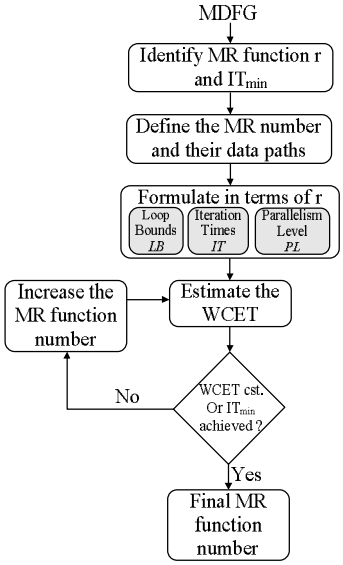

Fig. 4. The approach flow graph

B. *Multidimensional retiming analysis*

Each MR consists in reducing the critical path sizes by shifting nodes from the innermost loop and adding them in both sides of loop nests. Both actions affect the timing behaviour of the nested loops. To evaluate this modification, it is necessary to identify the possible MR modifications and the data-paths that are shifted. Those sub-paths can be identified by their latest nodes R with respect to the data dependency direction, where and $R \subseteq V$. Accordingly, this section objective is defining a set V_R of vector (i, u) where u is the node to retime and i is its MR order. In fact, those nodes are defined based on the timing and data dependency characteristics of the MDFG which are modelled on the D and T matrices, as described in section II. Thus, both matrices are swept to define the maximal number of MR functions, the retimed nodes and their MR function order. The D and T matrices are incrementally swept in the direction of the data dependencies. The node set S, having all incoming edges with non-zero-delays, are extracted in order to identify the last nodes R of the critical paths. So, each node is added to the set R if the computation time path is under IT_{min} and $D = (0, \dots, 0)$[13, 14]. Accordingly, the R list is filtered to remove redundancy and save the last ones. The first MR function adds delays to the R node outcoming edges. Hence, the R nodes are saved in the V_R list with their MR function order $(k = 1)$. These steps are repeated by sweeping from the R successors and incrementing the K level, as described in algorithm 2.

For example, the initial MDFG of the Jacobi algorithm, whose the critical path is $p_{ch}: A1 \rightarrow A2 \rightarrow D1 \rightarrow D2$, is shown in Fig.1(b). As indicated in algorithm 1, sweeping leads to select the $A2$ node to be retimed first. The second iteration in algorithm 1 implies the selection of the $D1$ node to be the second one to retime. As a result, $V_R = \{(A2,1), (D1,2)\}$.

ALGORITHM 1. MR analisys

Inputs: the D and T matrices, the minimal iteration time IT_{min}
Outputs: The list V_R, the MR number k
*/*identify the last nodes S of critical paths*/*
For e∈E where e:u→v **do**
  If d(e)≠(0,…,0) **then**
    Add v to S list
  End if
End for
k ← 0
Repeat
  k←k+1
  */*explore the successors R of the S nodes */*
  For each node v∈S **do**
    For each node x∈V **do**
      If D(v,x)=(0,…,0) and T(v,x) ≤ IT_{min} and x∉R **then**
        Add x to R list
      End if
    End for
  End for
  / identify the R nodes to retime*/*
  For each node p∈R **do**
    For each node q∈R **do**
      If D(p,q)=(0,…,0) **then**
        Delete p from R
      End if
    End for
  End for
  */*save the nodes to retime on V_R and their MR order k*/*
  For each node x∈R **do**
    add (x,k) to V_R
  End for
  S←R
Until all nodes are tested

C. *Innermost IT and parallelism level in terms of MR*

WCET prediction requires defining the innermost iteration time and the parallelism level for the initial MDFG and for each eventual MR function. Therefore, the parallelism levels are modeled as a set PL_R of vector $(i, PL_{loop}, PL_{pro}, PL_{epi})$ where i is the MR order, and PL_{loop}, PL_{pro} and PL_{epi} are the parallelism levels respectively of the innermost loop, the prologue and the epilogue. Similarly, the innermost ITs are modeled as a set IT_R of vector (i, IT_i). Based on the MR principles, the instructions added to the prologue are executed under IT_{min}. Therefore, the prologue and epilogue computation time are estimated in terms of IT_{min} and the current MR order.

The IT is equal to the computation time of the critical path, as $IT = max\{t(p), d(p) = 0\}$. For the initial MDFG, the critical path is defined by sweeping it from the nodes V_D, having all incoming edges with non-zero-delays, until the nodes V_A, having all outcoming edges with non-zero-delays. Each MR modifies the critical-path in a way that it is defined from the retimed nodes V_R, extracted by algorithm 1, to the V_A nodes. Similarly, the parallelism level of the initial MDFG corresponds to the maximal nodes that are executed in parallel, which are determined basing on their computation times. Taking as an example the Jacobi algorithm shown in Fig1, the MDFG is swept to identify the $p: A1 \rightarrow A2 \rightarrow D1 \rightarrow D2$ critical path. Knowing that the first MR is to be applied to the $A2$ node, sweeping is done from the $A2$ successor nodes to the V_A nodes. Consequently, the $D1 \rightarrow D2$ path is swept twice. The longer the critical path is, the more repeatedly the nodes are swept. For this purpose, the main idea is to compute the execution time of critical path once, for all MR functions that can be applied. In this purpose, the MDFG should be swept in the opposition direction of data dependencies and incrementally defining the critical-path computation-time in each node, where the result is stored in the V_{CP} list. To identify the execution time critical path after each MR function, the node having the same level is selected from the V_R, to extract the one having the maximal-execution-time critical path. Elsewhere, for each MR function, the parallelism level p of retimed sub-paths is computed. Thereafter, p is added to the previous PL_{loop} and PL_{pro}, and $(p_{init\_loop} - p)$ is added to the previous PL_{epi} where p_{init_loop} is the initial innermost loop parallelism level, as described in algorithm 2.

ALGORITHM 2. IT and PL in terms of MR

Input: a realizable MDFG G =(V,E,d,t), the V_R list
Output: the innermost iteration time list IT_R and the parallelism level list PL_R in terms of MR
*/*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 L list
  End if
End for
/ Save computation time V_{CP} of L nodes*/*
For each u∈L **do**
  $V_{CP}(u) \leftarrow t(u)$
End for
*/*Compute the path computation time V_{CP} starting by each node*/*
While Exists node not tested **do**
  For each node u∈L **do**
    Extract the V list of u predecessor nodes
    Add to N
    For each node v∈V **do**
      If $V_{CP}(v) \leq V_{CP}(u)+t(v)$ **then**
        $V_{CP}(v) \leftarrow V_{CP}(u)+t(v)$
      End if
    End for
  End for
  L←N
End While
/ define the IT_R set */*
Add (0, max (V_{CP})) to IT_R set
For each node u∈V_R **do**
  If $V_R(u)$ = i **and** $V_{CP}(u) - t(u) \geq IT_i$ **then**
    Add (i , ($V_{CP}(u) - t(u)$)) to IT_R set
  End if
End for
/ define the PL_R set */*
Define Pl_{init_loop}
Add (0, Pl_{init_loop}, 0, 0) to PL_R set
For each MR order i **do**
  Compute the parallelism level p of the retimed sub-paths
  Extract the PL element (i-1, x, y ,z)
  Add (i, (x+p), (y+p), (z+(Pl_{init_loop} -p))) to PL_R set
End for

In the case of the Jacobi algorithm MDFG shown in Fig.1(b), the $D2$ node is the last one for all zero-delay paths, which are saved in the L list. The MDFG is swept in the opposite direction to compute all the path execution times, which are finished by $D2$, whose values are stored in the V_{CP} list, where $V_{CP} = \{(A1,6), (A2,5), (D1,4), (D2,2)\}$. The following loop in algorithm.2 ensures defining the innermost iteration times after applying respectively the first and second MR functions, where $IT_R = \{(0,6), (1,4), (2,2)\}$. The last loop ensures defining the set$PL_R = \{(0,1,0,0), (1,2,1,1), (2,3,2,2)\}$.

D. Loop bounds analysis

The MDFG allows modelling l nested loops, whatever the loop iteration bounds are. Therefore, all iteration loop bounds are modelled as $LB = (n_1, n_2 \dots, n_k)$, where n_i is the loop bound of the i loop and $1 \leq k \leq l$. As indicated in section II, the MR leads to reduce the loop bounds with respect to the MR function r in a way that if $r = (r_1, r_2 \dots, r_k)$, the n_i loop bound is reduced by $|r_i|$. This decline is ensured either the r_i value, which is positive or negative. Thus, the nested loop bounds after the first r MR function is as shown in Equation 6.

$$LB_{(1)} = (n_1 - |r_1|, n_2 - |r_2|, \dots, n_k - |r_k|) \quad (6)$$

Elsewhere, the r function is enabled to be applied several times, as much as the full parallelism is required. Therefore, the more the MR is applied, the loop bounds decrease with respect to the r_i values. The LB loop bound is formulated in terms of the i MR function rank, as described in Equation 7.

$$LB_{(i)} = (n_1 - i \times |r_1|, n_2 - i \times |r_2|, \dots, n_k - i \times |r_k|) \quad (7)$$

After applying the MR function $r = (1, -1)$, the outermost and innermost loop bounds are respectively decreased by $|1|$ and $|-1|$. The same growth is implied after each MR function, as shown in Fig.3(b). Accordingly, the final $LB_{(r,i)}$ is formulated as described in Equation 8.

$$LB_{(i)} = (n_1 - i, n_2 - i) \quad (8)$$

E. WCET prediction

The WCET is predicted for all the nested loops in terms parallelism level that corresponds to the i order of the MR function. Since the prologue and epilogue are added in both sides of each retimed loop, the nested loop *WCET* is incrementally computed from the innermost loop $WCET_{(1,i)}$ to the outermost one $WCET_{(l,i)}$. For the innermost loop, the $WCET_{(1,i)}$ represents the addition of the $WCRT_{(1,i)}$ and the $WCST_{(1,i)}$ as described in Equation 3. The first term is equal to the iteration time IT_i multiplied by its loop bound $LB_{(1,i)}$, which corresponds to the ith MR function. The second term is the multiplication of the loop bound $LB_{(1,i)}$, the parallelism level $PL_{(1,i)}$ and the C constant. If $r_1 \neq 0$ and $1 < l \leq k$, a prologue is added in both sides of the l loop. As indicated in [13], the instructions shifted outside the loop are executed in IT_{min}. The greater $|r_k|$ is, the more the prologue execution time increases. Moreover, the prologue and epilogue WCSTs are computed in terms of PL_{pro}, PL_{epi} and $LB_{(l,i)}$. Accordingly, the prologue and epilogue WCET is computed as described in Equation 9.

$$WCET_{pro-epi_{(l,i)}} = (|r_l| \times IT_{min}) + (LB_{(l,i)} \times C \times (PL_{pro_{(l,i)}} + PL_{epi_{(l,i)}})) \quad (9)$$

Therefore, each l loop $WCET_{(l,i)}$, where $1 < l \leq k$, is formulated in terms of the $WCET_{pro-epi_{(l,i)}}$ and the $WCET_{(l-1,i)}$. Hence, the whole loop nested loop WCET is defined as shown in Equation 10.

$$WCET_{(l,i)} = \begin{cases} LB_{(l,i)} \times (IT_i + PL_{(l,i)} \times C), \\ \qquad if \quad l = 1 \\ LB_{(l,i)} \times \left(WCET_{(l-1,i)} + WCET_{pro-epi_{(l,i)}}\right), \\ \qquad if\ 1 < l \leq k \end{cases} \quad (10)$$

F. Optimization heuristic

Our approach aims to provide the MR function number that allows achieving the execution time constraint. In this context, it determines the MR parameters. Thereafter, the WCET is formulated in terms of MR parameters. Thus, the approach starts by computing the IT_{min}, the MR function r and the D and T matrices, as indicated respectively in [12, 13]. Accordingly, the both suggested algorithms are called, where the first one generates the MR function number and the nodes to retime, while the second one provides the IT_R and PL_R sets. Hence, the approach iteratively computes the WCET in order to compare it to the constraint. The first iteration corresponds to the initial MDFG. The last one corresponds to the retimed MDFG with an increased MR function number. The whole steps of our proposed optimization heuristic are described in algorithm 3.

ALGORITHM 3. MR for WCET constraint

Inputs: a realizable MDFG G =(V,E,d,t), the execution time constraint T_C, the iteration bounds of the outermost loop and the innermost loop
Outputs: the nr number of MR function
/ Compute the execution time and the minimal cycle period */*
Identify the minimal iteration time IT_{min}
Compute D and T matrices [13]
Define the M_R function r [12]
Identify the V_R list and the MR number K (as described in algorithm 1)
Identify the IT_R and the PL_R set (as described in algorithm 2)
$n_r \leftarrow 0$
Repeat
Compute $WCET_{(1,i)}$ (as indicated in Equation 10)
  for lp from 1 to n **do**
    Compute $WCET_{(lp,i)}$ (as indicated in Equation 10)
  end for
$n_r \leftarrow n_r + 1$
While $T \leq T_C$ and $n_r < k$

IV. EXPERIMENTAL RESULTS

In our experiments, the first step consists of verifying the WCET estimation, by comparing its values to the ones provided by the SWEET software tool [5, 19]. The second step, the optimization heuristic is compared to the delayed MR technique by evaluating the core number development of the provided implementations in terms of WCET constraints. Our benchmarks include several 2D nested loops which are the Infinite Impulse Response Filter (IIRF) [13], the Jacobi Algorithm (JA) [25], the Walsh-Fourier Transform (WFT) [12], and the Wave Digital Filter (WDF) [14]. The core numbers are respectively shown in Fig.6.

All benchmarks are explored to define the MR function and the required function number to achieve the full parallelism, as indicated in algorithm 1. Then, the estimation model, described on section III, is applied to define the WCET in terms of MR function, whose values are cited in table III.

Table III. MR parameters and WCETs of applications

Benchmarks	MR function	MR number	Execution time				
			Without MR	MR function number			
				1	2	3	4
IIRF	(1, -1)	4	1000	890	720	560	316
JA	(0, 1)	2	540	400	240	--	--
WFT	(1, -1)	2	368	242	138	--	--
WDF	(0, 1)	2	1324	917	504	--	--

The SWEET software tool intercepts the nested loop code and the architectural timing parameter to provide the upper bound WCET. The verification main idea compares the SWEET provided values to those computed by the proposed equation system. This step is done for all benchmarks exposed in Table III, which values are shown in Fig. 5. As a consequence, the proposed WCET estimation offers an efficient prediction with an error rate equal to 8.54 % compared to the SWEET tool.

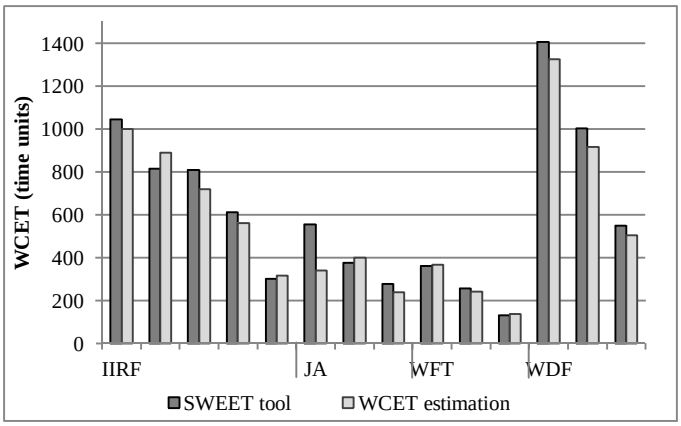

Fig. 5. Benchmark WCETs in terms of SWEET tool and WCET equation

Each application is submitted to five execution time constraints, as indicated in the first column of Table IV. Therefore, the optimization approach selects the MR parallelism and estimates its execution time that respects the target constraint, which are listed in the third column of Table IV. To get each constraint, the optimization approach provides implementation with a different parallelism level, which is in relation to the core number. The full parallel implementations are only achieved in the case of minor execution time constraints.

Table IV. WCETs and core numbers in terms of WCET Constraints

Benchmarks	WCET constraint	WCET	Core number
IIRF	400	316	16
	650	560	15
	800	720	14
	950	890	12
	1100	1000	8
JA	300	240	3
	400	400	2
	500	400	2
	600	540	1
	700	540	1
WFT	200	138	4
	250	242	3
	300	242	3
	350	242	2
	400	368	2
WDF	600	504	4
	800	504	4
	1000	917	3
	1200	917	3
	1400	1324	2

After analyzing the optimization approach results, we compare its implementations to those provided by the delayed MR retiming in terms of core number, as shown in Fig.6. As a consequence, the optimization approach presents a benefit in terms of core numbers regarding the delayed MR technique for an average improvement of 27.18%.

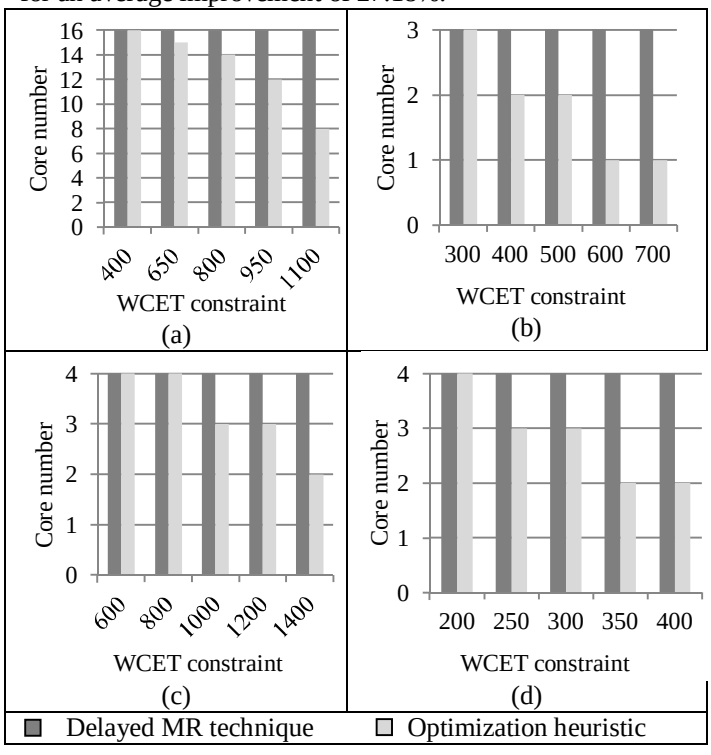

Fig. 6. The core number development in term of delayed MR technique and the optimization heuristic : (a) IIRF, (b) JA, (c) WFT and (d) WDF

V. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed a nested loop WCET static analysis and an optimization heuristic that allows increasing the instruction-level-parallelism in order to achieve a WCET constraint. The main idea consists in providing an optimal parallelism level instead of a full parallel implementation. The experimental results show a noticeable efficiency of the WCET prediction and an important optimization of the heuristic in terms of processing cores.

In our future works, we aim to extend the approach to optimize the cache memory and overhead ratio, which allow enhancing the WCET estimation. Moreover, this work is enabled to be extended to other parallelism techniques such as the loop tilling and loop stripping, hence the whole parallelism solution space, which consequently leads to optimal implementations.

VI. REFERENCES

[1] Pascal Raymond, Claire Maiza, Catherine Parent-Vigouroux and Fabienne Carrier, "Timing Analysis Enhancement for Synchronous Program", journal of Real-Time Systems, March 2015, Volume 51, Issue 2, pp 192-220.

[2] Julien Henry, Mihail Asavoae, David Monniaux, Claire Maîza, "How to Compute Worst-Case Execution Time by Optimization Modulo Theory and a Clever Encoding of Program Semantics", in the ACM SIGPLAN Conference on Languages, Compilers and Tools for Embedded Systems (LCTES 2014), Pages 43-52, ISBN: 978-1-4503-2877-7.

[3] Stefan Bygde, Andreas Ermedahl and Bjorn Lisper, "An Efficient Algorithm for Parametric WCET Calculation", Journal of Systems Architecture, Volume 57, Issue 6, June 2011, Pages 614–624.

[4] Haluk Ozaktas, Christine Rochange and Pascal Sainrat, "Minimizing the Cost of Synchronisations in the WCET of Real-Time Parallel Programs", Proceedings of the 17th International Workshop on Software and Compilers for Embedded System (SCOPES 2014) , pages 98-107, table of contents ISBN: 978-1-4503-2941-5.

[5] Mark Bartlett, Iain Bate and Dimitar Kazakov, "Guaranteed Loop Bound Identification from Program Traces for WCET", 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009 (RTAS 2009), 13-16 April 2009, Page(s): 287 – 294, ISSN : 1545-3421.

[6] Marianne De Michiel, Armelle Bonenfan, Hugues Cassé, "Normalisation of Loops with Covariant Variables", Electronic Notes in Theoretical Computer Science, Volume 289, 6 December 2012, Pages 41–51.

[7] Duc-Hiep Chu, Joxan Jaffar, "Symbolic simulation on complicated loops for WCET Path Analysis", Proceedings of the International Conference on Embedded Software (EMSOFT), 2011, October 9–14, 2011, Taipei, Taiwan, Page(s): 319 – 328, Print ISBN: 978-1-4503-0714-7

[8] Jens Knoop, Laura Kovacs, and Jakob Zwirchmayr, "Symbolic Loop Bound Computation for WCET Analysis", Lecture Notes in Computer Science Perspectives of Systems Informatics, Volume 7162, 2012, pp 227-242.

[9] M. W. Benabderrahmane, L. N. Pouchet, A. Cohen, C. Bastoul, "The Polyhedral Model Is More Widely Applicable Than You Think", Compiler Construction Lecture Notes in Computer Science Volume 6011, 2010, pp 283-303.

[10] A. Morvan, S. Derrien, P. Quinton, "Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion", in International Conference on Field-Programmable Technology (FPT), Page(s):1 – 10, New Delhi (india), 12-14 Dec. 2011.

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

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

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

[14] 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.

[15] Paul Lokuciejewski, Peter Marwedel, "Combining Worst-Case Timing Models, Loop Unrolling, and Static Loop Analysis for WCET Minimization", 21st Euromicro Conference on Real-Time Systems, 2009.

[16] Jonathan Barre, Cédric Landet, Christine Rochange, Pascal Sainrat, "Modeling Instruction-Level Parallelism for WCET Evaluation", Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06), 2006.

[17] Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui, "Execution Time and Code Size Optimization using Multidimensional Retiming and Loop Striping", 16th Euromicro Conference on Digital System Design, September 2013, Santander (Spain), pages 462-466.

[18] L. Kaouane, M. Akil, T. Grandpierre, "A methodology to implement real-time applications on reconfigurable circuits", The Journal of Supercomputing, December 2004, Volume 30, Issue 3, pp 283-301.

[19] Jan Gustafsson, Andreas Ermedahl, Christer Sandb erg, and Bjorn Lisp er, "Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis using Abstract Execution", Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International, Dec. 2006, pages : 57 – 66.

[20] Sadjadi, S.M. ; Florida Int. Univ. (FIU), Miami, FL ; Shimizu, S. ; Figueroa, J. ; Rangaswami, R., "A Modeling Approach for Estimating Execution Time of Long-Running Scientific Applications, Parallel and Distributed Processing", 2008. IPDPS 2008. IEEE International Symposium on, 14-18 April 2008, pages:1-8, Miami, FL.

[21] Jukka Mäki-Turja, Mikael Sjödin, "Response-Time Analysis for Transactions with Execution-Time Dependencies", in 19th International Conference on Real-Time and Network Systems (RTNS'11), Sep. 2011, pages: 139- 145.

[22] Pascal Raymond, Claire Maiza, Catherine Parent-Vigouroux and Fabienne Carrier, "Timing Analysis Enhancement for Synchronous Program", RTNS '13 Proceedings of the 21st International conference on Real-Time Networks and Systems RTNS'2013, Pages 141-150.

[23] Peter Altenbernd , Andreas Ermedahl, Bjorn Lisper, Jan Gustafsson , "Automatic Generation of Timing Models for Timing Analysis of High-Level Code", in 19th International Conference on Real-Time and Network Systems (RTNS'11), Sep. 2011, page 55-64.

[24] C. Ballabriga, H. Cassé,"Improving the WCET computation time by IPET using control flow graph partitioning", 8th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, p. 19-27, jully 2008.

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

[26] Radu Prodan, Thomas Fahringer, "Overhead Analysis of Scientific Workflows in Grid Environments", IEEE Transactions on parallel and distributed systems, VOL. 19, NO. 3, MARCH 2008.

[27] Y. Elloumi, M.Akil, and M.H. Bedoui, "Execution time optimization using delayed multidimensional retiming", International Journal of High Performance Systems Architecture (IJHPSA), ISSN: 1751-6528 , in press.