Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 980762, 20 pages
doi:10.1155/2010/980762

Research Article

New Three-Level Resource Management Enhancing Quality of Offline Hardware Task Placement on FPGA

Ikbel Belaid,[1, 2] Fabrice Muller,[1, 2] and Maher Benjemaa[1, 2]

[1] *University of Nice Sophia-Antipolis/LEAT-CNRS, 250 rue Albert Einstein, bât 4. 06560, Sophia Antipolis - Cedex, France*
[2] *Research Unit ReDCAD, National Engineering School of Sfax, B.P. 1173-3038 Sfax, Tunisia*

Correspondence should be addressed to Ikbel Belaid, ikbel.belaid@unice.fr

Received 14 November 2009; Revised 6 March 2010; Accepted 24 April 2010

Academic Editor: Christophe Bobda

Currently, reconfigurable hardware devices feature a high density of heterogeneous resources to enable multitasking and offer flexibility in application needs. These concepts raise the need for efficient management of hardware tasks and hardware resources. The scheduling of hardware tasks is highly dependent on placement. Placement focuses on allocation of hardware resources required by the scheduled hardware tasks. In this paper, we propose novel three-level resource management that investigates enhancement of placement quality by reducing task rejection, configuration overheads, and by optimizing resource utilization. Improving placement quality will produce significant enhancement of performance for scheduling and overall execution time of the application in FPGA. Hence, the placement problem is formulated into a constrained optimization problem and resolved with powerful solvers using the Branch and Bound method. The obtained results of an application of heterogeneous hardware tasks show an average resource utilization of 36% of the available resources on the reconfigurable region and an overall overhead of 11% of total application running time, and we have eliminated the issue of task rejection. Compared to static implementation, the gain in resource utilization within the reconfigurable region achieves up to 43%.

1. Introduction

Scheduling and placement are strongly linked. The scheduler decides which of the ready tasks should be executed next and calls the placer to find a feasible location. The scheduler decision should be taken in accordance with the ability of placer to allocate free resources required by the elected task. Field-Programmable Gate Array (FPGA) is the most widely used reconfigurable hardware device. Today's FPGA devices provide several million reconfigurable heterogeneous resources. The development of dynamic partial reconfiguration in the FPGAs allows reconfiguring only the necessary part of the FPGA when required without interfering with any other parts running on the same FPGA. While this technique can increase device utilization and performance of scheduling and application, it also leads to high configuration overhead, fragmentation, and complex allocation situations of hardware tasks [1]. Frequently, the existing methods of placement face these issues. Consequently, the quality of placement and performance of the scheduling degrades while the overall response time increases. So, there exists a serious need to define an efficient method that helps manage the area of resources.

In general, the placement of hardware tasks consists of two main functions: (i) *partitioning,* which handles the free space in the device and identifies any potential sites enabling execution of hardware tasks, and (ii) *fitting,* which selects the feasible placement solution. In this paper, under FOSFOR project [2], we address the aspect of placement and introduce new three-level offline resource management that challenges all the above mentioned issues. The main concern of our method is enhancing placement quality by targeting the optimized use of FPGA's resources and taking into account the physical and functional features of hardware tasks (FOSFOR (Flexible Operating System For Reconfigurable platform) is French national program (ANR) targeting the most evolved technologies. Its main objective is to design a real time operating system distributed on hardware and software execution units which offers required flexibility to application tasks through the mechanisms of dynamic reconfiguration and homogeneous Hw/Sw OS services.). During the conception of three-level resource management, we rely on the physical architecture of target technology and on the advantages of dynamic partial reconfiguration. The contribution of this paper is the development of

(i) offline flow for hardware task classification which is an enhancement of our previously proposed work in [3] and which creates task classes,

(ii) a formulation of the task placement as a constrained optimization problem and its resolution by powerful solvers using Branch and Bound method by considering two independent sublevels: the first sublevel ensures the fitting of obtained task classes on physical blocs partitioned on target technology which improves resource efficiency up to 36% and the second sublevel performs the mapping of tasks to obtained classes by optimizing the overall overhead up to 11% of total running time and by minimizing the number of task relocations.

The remainder of this paper is organized as follows. Section 2 reviews related work of placement. Section 3 details our three-level resource management. Section 4 focuses on the formulation of hardware task placement as a constrained optimization problem and its resolution by the Branch and Bound method. The obtained results are depicted in Section 5. In Section 6, we summarize our work, make some concluding remarks, and present future works.

2. Related Work

Current strategies dealing with task placement are divided into two categories: offline placement and online placement.

2.1. Online Methods for Hardware Task Placement. The main reference is [4], Bazargan et al. suggest online scenario for hardware task placement. In fast on-line placement, Bazargan et al. introduce two partitioning techniques; the first technique denoted *Keeping All Maximal Empty Rectangles (KAMER)* searches all the Maximal Empty Rectangles (MER) after each task insertion/deletion operation. The MERs are defined as the empty rectangles which are not contained in another empty rectangle and are not necessarily disjoined. The second technique of partitioning called *Keeping Nonoverlapping Empty Rectangles* keeps all the nonoverlapping holes and is evoked after each split/merge operation. For both previous techniques of partitioning, the fitting in the on-line placement is conducted by the *First Fit*, *Best Fit (BF)* and *Bottom Left* bin-packing algorithms [5].

Fast two-dimensional on-line placement is presented in [6] which is an extension of Bazargan's partitioning; the *Keeping Nonoverlapping Empty Rectangles*. In [6], Walder et al. propose placement methods that rely on efficient algorithms of partitioning enhancing the quality of Bazargan's partitioning by 70% and on a hash matrix data structure that finds a feasible placement in constant time. Based on a nonpreemptive system without precedence constraint, the Walder's partitioner delays split decision instead of using the Bazargan's heuristics for decision of split. Reference [6] details four enhancements of Bazargan's partitioner. The main partitioner is On-The-Fly (OTF) partitioner which consists of resizing empty rectangles only if the newly arrived task overlaps them. After each task insertion or deletion, the Walder's partitioner updates the data on the hash matrix. If a new arrived task fits into more than one empty rectangle determined by the hash matrix, a fitting strategy is used to choose a rectangle [6]. Reference [6] implements four types of fitting: *BF*, *Worst Fit*, *Best Fit with Exact Fit,* and *Worst Fit with Exact Fit*.

Ahmadinia et al. present in [7] a new method of on-line placement by managing the occupied space instead of free space because of the difficulty of managing empty space and the huge increase of empty rectangles. In [7], at the arrival of a new task, the space manager starts by delimiting the *Impossible Placement Region (IPR)* relative to placed modules and to device. Thereafter, the *Nearest Possible Position* fitter selects the optimal point that gives the optimal communication cost, which is not included in the *IPR*.

In [8], Marconi et al. extend Bazargan's placement by means of an *Intelligent Merging* (*IM*) algorithm. IM dynamically combines three techniques of managing free resources: *Merging Only if Needed*, *Partially Merging,* and *Direct Combine*. IM accelerates Bazargan's partitioner by 3 and improves placement quality by increasing the rate of accepted tasks.

Handa et al. introduce the staircase method in [9]. The staircase method handles free space during the first subfunction of the on-line placement. This method is considered efficient as it tries to cover the faults of the *KAMER* method, especially for task rejection.

Some approximate metaheuristics are adopted to resolve the hardware task placement such as [10] that employs an on-line task rearrangement by using genetic algorithm approach. When a newly arrived task could not be placed immediately, the proposed approach tries to rearrange a subset of tasks executing on the FPGA to allow the processing of the pending task sooner. The approach is based on the First-Fit strategy and genetic algorithm. By allowing the rotation of tasks and by using input buffer to save the data of suspended tasks, the approach combines two genetic algorithms to resolve two subproblems. The first subproblem identifies a feasible rearrangement and the second one consists on scheduling the moves of executing tasks to attain the feasible rearrangement.

Reference [11] copes with task placement problem and adopts interconnection-based FPGA as support for run-time reallocation of hardware tasks. It applies matador task concurrency management methodology for scheduling hardware tasks on identical tiles by minimizing run-time reconfiguration. This goal is reached by two new techniques implied in the scheduler named configuration reuse and configuration prefetch. Reduction in configuration overhead decreases significantly the execution time and energy consumption.

2.2. Offline Methods for Hardware Task Placement. In the offline scenario for hardware task placement, [4] defines 3D templates depicting the tasks in time and space dimensions and uses slow heuristics: simulated annealing and greedy research, and KAMER-BF to perform a high-quality placement in terms of resource utilization and task rejection.

Reference [12] models the problem of resource allocation as a 0-1 integer linear programming problem which aims necessarily at minimizing the resources area which is reconfigured at runtime. Reference [12] considers an application with known sequential execution trace. Hence, the huge configuration latency is tackled by reducing the overlapped areas between tasks.

By considering the placement of hardware tasks as rectangular items on hardware device as rectangular unit, several approaches for resolving the two-dimensional packing problem are proposed. For example, in [13], the offline approximate heuristics: Next-Fit Decreasing Height, First-Fit Decreasing Height, and Best-Fit Decreasing height are presented as strip-packing approaches based on packing items by levels.

In addition, Lodi et al. in [14] propose different offline approaches to resolve hardware task placement as 2D bin-packing problem. For instance, the Floor-Ceiling algorithm that considers alternate directions for packing tasks, either from left to right when their bottom edges touch the level floor or from right to left; when their top edges are on the top of the level floor. The Knapsack packing algorithm is also proposed in [15] which initializes each level by the tallest unpacked item and completes it by packing tasks as the associated Knapsack problem that maximizes the total area within level. For both latter algorithms, the second phase of 2D bin packing is achieved by Best-Fit Decreasing algorithm.

Baker et al. define in [16] the Bottom-left (BL) offline algorithm. BL packs each hardware task in the bottom left position.

Martello and Vigo propose in [17] an enumerative offline approach to exact solution for 2D bin-packing. Their algorithm is based on a two-level branching scheme: the outer branch-decision tree that assigns tasks to the bins without specifying their position and the inner branch-decision tree that enumerates all possible patterns.

By optimizing the total execution time and the resource utilization, the method of placement in [18] consists of two phases: the first phase is the recursive bi-partitioning by means of slicing tree that defines the relative position of each hardware task towards the other hardware task placement and finds the appropriate room in the reconfigurable device for each hardware task according to task's resources and intertask communication. The second phase uses the obtained room topology to achieve the sizing that computes the possible sizes for each room.

Reference [19] minimizes task rejection and presents offline algorithms for 3D floorplanning of hardware tasks on reconfigurable functional unit such as: KAMER-BF Decreasing, Simulated Annealing, Low-temperature Annealing, and Zero-temperature Annealing. The 3D placement models tasks as 3D boxes having a base corresponding to the spatial dimensions of tasks and a height corresponding to their time-span.

In [20], as bin-packing problem, an offline approach is proposed by Fekete et al. through a graph-theoretical characterization of the packing of a set of items into a single bin. Tasks are presented as three-dimensional boxes and the feasible packing is decided by the orthogonal packing problem within a given container. Their approach considers packing classes, precedence constraints, and the edge orientation to solve the packing problem. Similarly, in [21], Teich et al. definesthe task placement as more-dimensional packing problem. Tasks are modeled as 3D polytopes with two spatial dimensions and the time of computation. Based on packing classes as well as on a fixed scheduling, they search a feasible placement on a fixed-size chip to accommodate the set of tasks. The resolution is performed by Branch and Bound technique to optimality of dynamic hardware reconfiguration.

The major shortcoming of all the above-proposed methods of placement is that they are applicable only in homogeneous devices. In fact, these methods assume that the relocation of tasks is allowed and enable the allocation of resources whenever sufficient free space is available. Furthermore, the placement disregards the routing constraints as it does not address the issue of intertask communication and I/O routing. Moreover, tasks are nonpreemptive and almost-identical. Unfortunately, the algorithms for 2D packing focus only on the objective of minimizing resource waste and do not satisfy all other goals. All the existing strategies of placement provide a nonguarantee system as they suffer from task rejection and fragmentation. We believe the issue of task rejection is caused by the constructive way in which the placement is performed throughout all existent strategies of placement. The issue of fragmentation may lead to undesirable situations where a new task cannot be placed although there would be sufficient free space.

As we have full knowledge about the set of hardware tasks and the features of the reconfigurable device, in this paper, we present a realistic three-level resource management solution as a new strategy to perform offline placement of hardware tasks in FPGA. This new strategy aims at enhancing placement quality by trimming the previously mentioned issues. Our proposed method is technology-dependent and in accordance with generic placement as the second level partitions the available resources in FPGA according to the task classes provided by the first level. Nevertheless, the third level ensures the subfunction of fitting. The task model is preemptive and preemption points are predefined. Our resource management allows the relocation of tasks and results in strict positions for each hardware task by respecting its preemption points and types of resources.

3. Three-Level Resource Management

We use Xilinx's Virtex FPGA as a reference for the hardware reconfigurable device to lead our hardware resource management study. We offer a definition of a few terms which are used throughout the paper: *NT* is the number of tasks, *NR* the number of Reconfigurable Physical Blocs, *NZ* the number of Reconfigurable Zones, and *NP* the number of resource types in the chosen technology. In the beginning, we should start by introducing the hardware task models. We have defined three models.

(i) *The functional model*: this contains the functional features of hardware tasks T_i as the worst case execution time (C_i), the period (P_i), and preemption points l ($Preemp_{i,l}$). The number of preemption points of T_i is denoted by $NbrPreemp_i$. This number also includes the first point of execution of T_i. Preemption points are specified by the designer.

(ii) *The behavioral model*: This includes the finite state machine controlling each task and which handles a set of *NbrReg* registers of 32 bits to conduct the context switch. The behavioral model defines the functional overhead ($Context_i$) that is needed to preempt or resume the execution of tasks. This functional overhead is fixed for all the hardware tasks as they have similar register banks with *NbrReg* registers. The functional overhead is computed as two times (save and load) the access to a bus having 32 bits of width and functioning at 80 MHz. In addition, we have considered the worst case, when tasks need the *NbrReg* 32bit-registers to perform context switch. Hence, this functional overhead represents sequential access of *NbrReg* registers associated for a given task to save and load its context through a 32 bit-bus ($Context_i$ = 2x $NbrReg$/80 MHz).
In our preemptive modeling, we do not use the classical method of readback and load bitstream since it takes a significant latency, complicates the preemption, and requires a large space memory as a new readback bitstream must be saved at each preemption. Thus, we resort to save the state of finite state machine with an acceptable amount of data by keeping always the same bitstream for each task. Preemption points of hardware tasks are fixed in a way to reduce the data dependency that could exist between two states. In fact, we must avoid keeping a preemption point between two states processing the same data because we need to save these data into an external memory which might increase the overhead at run-time. Otherwise, it is recommended to put a preemption point when the task is in a blocked state waiting for receiving external resource to allow the ready tasks to be executed in the RZ. As tasks are periodic, a preemption point could be inserted after the last state before restarting the task to avoid any data dependency. In the finite state machine, the longest execution time between two states must be considered in order to deduct the worst case execution time.

(iii) *The RB-model*: tasks are presented as a set of reconfigurable resources called Reconfigurable Blocs (RB). The RBs are closely shaped to the reconfiguration granularity in the chosen technology. The determination of the RB-model of hardware tasks is well-detailed in our work in [3]. Each type of RB is characterized by specified cost $RBCost_k$ which is defined according to three parameters: the number of the RB type in the device, its power consumption and the importance of its functionality. The more consistent these parameters are, the higher the cost of RB type. The RB-model of each hardware task is described by (1).

$$T_{i-}RB = \{\alpha_{i,k} RB_k\}, \quad \alpha_{i,k} \text{ is natural}, \quad 1 \le i \le NT, \; 1 \le k \le NP. \tag{1}$$

The functional model and the RB-model are the basic models for resource management. However, the behavioral model is employed during scheduling. The FPGA provides reconfigurable resources organized according to column-based technology [22]. The management of hardware resources on FPGA consists of three levels.

3.1. Level 1: Offline Flow of Hardware Task Classification.

Level 1 takes a set of tasks as input and provides the types and instances of Reconfigurable Zones (RZ). The RZs are abstractions of task classes and are defined according to the types of resources needed by the tasks. As the RZs model the classes of hardware tasks, they are described by their RB-model given by

$$RZ_{i-}RB = \{\beta_{i,k} \, RB_k\}, \quad \beta_{i,k} \text{ is natural}, \quad 1 \le i \le NZ, \; 1 \le k \le NP. \tag{2}$$

Level 1 consists of three steps:

Search of RZ types: Step 1 gathers the tasks sharing the same types of RBs under the same type of RZ. Step 1 is essentially based on RB-model of hardware tasks and is achieved by Algorithm 1 of complexity in the worst case O ($NT * NP * NZ$).

Step 1 scans the RB-model of each hardware task and checks whether there exists in the list of RZ types *List-RZ* an already inserted type of RZ that closely matches the required types of RBs in the task (line 6). In this case, step 1 updates the number of RBs within this type of RZ by the maximum between the number of RBs in the task and that in the RZ (line 9). If the required types of RBs in the task do not match any type of RZ included in the *List-RZ*, the algorithm of the search of RZ types decides the creation of a new type of RZ as required by the task (line 13) and inserts it in *List-RZ* (line 14). At the end of step 1, we obtain the possible types of RZs. The number of RZ types is limited by the number of tasks. As shown in Figure 1, step 1 groups T_1 and T_3 in the same type of RZ (RZ_1) as both need RB_1 and RB_2 and adjusts the number of each RB type within RZ_1 by the maximum number of RBs between $T_1$and T_3. Similarly, RZ_2 is created by T_2 and T_4, and T_5 defines the third type of RZ (RZ_3).

Classification of hardware tasks: Step 2 starts by computing cost D between hardware tasks and RZ types resulting from step 1. Based on RB-models of hardware tasks (T_i) and RZs (RZ_j), cost D is computed as follows according to two cases.

We define by

$$d_{i,j,k} = \alpha_{i,k} - \beta_{j,k}, \quad 1 \le i \le NT, \quad 1 \le j \le NZ, \; 1 \le k \le NP. \tag{3}$$

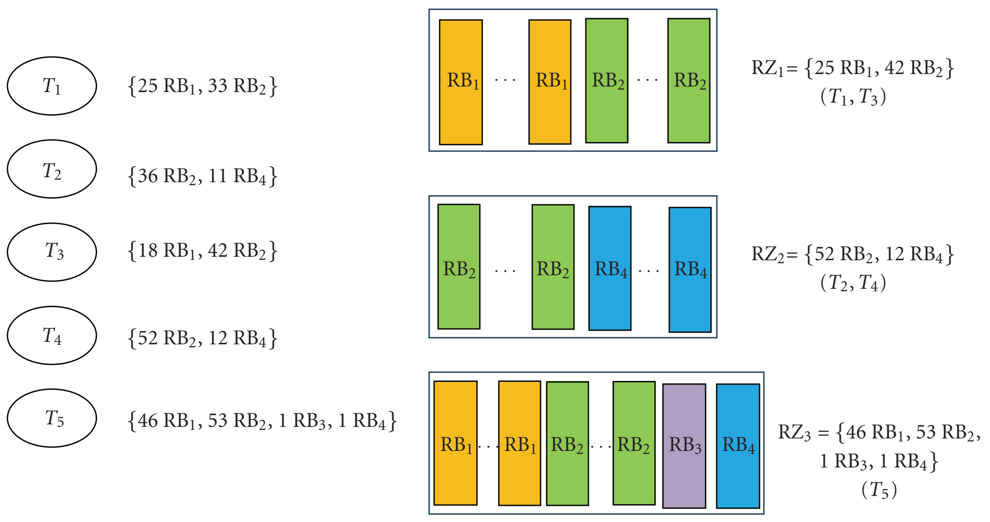
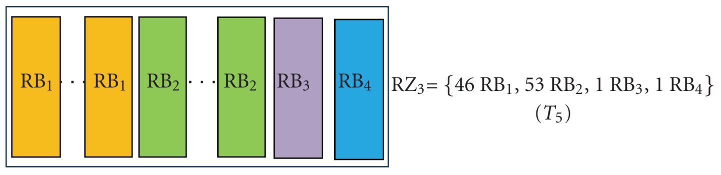

T_1
{25 RB$_1$, 33 RB$_2$}
T_2
{36 RB$_2$, 11 RB$_4$}
T_3
{18 RB$_1$, 42 RB$_2$}
T_4
{52 RB$_2$, 12 RB$_4$}
T_5
{46 RB$_1$, 53 RB$_2$, 1 RB$_3$, 1 RB$_4$}
RB$_1$ ··· RB$_1$ RB$_2$ ··· RB$_2$
RZ$_1$= {25 RB$_1$, 42 RB$_2$} (T_1, T_3)
RB$_2$ ··· RB$_2$ RB$_4$ ··· RB$_4$
RZ$_2$= {52 RB$_2$, 12 RB$_4$} (T_2, T_4)
RB$_1$ ··· RB$_1$ RB$_2$ ··· RB$_2$ RB$_3$ RB$_4$
RZ$_3$ = {46 RB$_1$, 53 RB$_2$, 1 RB$_3$, 1 RB$_4$} (T_5)

Figure 1: Example of RZ types search.

```
RZ-type = 0 // RZ types
List-RZ // list of RZ types
n // natural
for all tasks $T_i$ do
    // $T_{i\_}RB = X_{i,k}\, RE_k$
    if ((RZ-type $\neq 0$) and ($\exists$ n, $1 \le n \le$ RZ-type)/for all k(($X_{i,k} \neq 0$ and $Z_{n,k} \neq 0$) or ($X_{i,k} = 0$
    and $Z_{n,k} = 0$))) then
        // this test checks whether the task matches with an RZ type that already exists in list-RZ
        for all k do
            $Z_{n,k} = \max(X_{i,k}, Z_{n,k})$ // update RB number of $RZ_n$
        end for
    else
        Increment RZ-type
        $RZ_{RZ\text{-}type}$ = Create new RZ($X_{i,k}$) // new type of RZ, $RZ_{RZ\text{-}type} = \{X_{i,k} RE_k\}$
        Insert(list-RZ, $RZ_{RZ\text{-}type}$)
    end if
End for
```

Algorithm 1: Search of RZ types.


RB$_1$ ··· RB$_1$ RB$_2$ ··· RB$_2$ RB$_3$ RB$_4$
RZ$_3$= {46 RB$_1$, 53 RB$_2$, 1 RB$_3$, 1 RB$_4$} (T_5)

RBCost$_1$= 20, RBCost$_2$= 80, RBCost$_3$= 192, RBCost$_4$= 340

D (T_1, RZ$_3$) = $20 \times |25 - 46| + 80 \times |33 - 53| + 192 \times |0 - 1| + 340 \times |0 - 1|$
D ($T2$, RZ$_3$) = ∞ (lack of 10 RB$_4$ in RZ$_3$ for T_2)
D (T_3, RZ$_3$) = $20 \times |18 - 46| + 80 \times |42 - 53| + 192 \times |0 - 1| + 340 \times |0 - 1|$
D (T_4, RZ$_3$) = ∞ (Lack of 11 RB$_4$ in RZ$_3$ for T_4)
D (T_5, RZ$_3$) = 0

Figure 2: Example of computing cost D with RZ_3.

Case 1. $for\ all\ k, d_{i,j,k} \leq 0, RZ_j$ contains a sufficient number of each type of RB (RB_k) required by T_i. In this case, cost D is equal to the sum of differences in the number of each RB type between T_i and RZ_j weighted by $\mathrm{RBCost}_{k.}$(see (4))

$$D\left(\mathrm{T}_i, RZ_j\right) = \sum_{1\leq k\leq NP} \mathrm{RBCost}_k \times \left| d_{i,j,k} \right|. \tag{4}$$

Case 2. $\exists\ k, d_{i,j,k} > 0$, the number of RBs required by T_i exceeds the number of RBs in the RZ_j or T_i needs RB_k which is not included in RZ_j. In this case, the cost D between T_i and RZ_j is infinite (see (5))

$$D\left(T_i, RZ_j\right) = \infty \tag{5}$$

Figure 2 illustrates the computing of costs D between the five tasks and RZ_3 described in Figure 1.

As shown in Table 1, step 2 assigns each task to the RZ giving the lowest cost D as described by the third column. For example, T_1 is assigned to RZ_1 since D (T_1, RZ_2) and D (T_1, RZ_3) are superior to D (T_1, RZ_1). Then, by using (6), step 2 computes the workload of each RZ according to this assignment and by using the functional models of hardware tasks

$$\text{Load of } RZ_j = \sum_{i \text{ in } RZ_j} \left(C_i/P_i + Nbr\mathrm{Preemp}_i \times \mathrm{Overhead}_{j,i}/P_i\right),$$
$$\mathrm{Overhead}_{j,i} = \mathrm{Config}_j + \mathrm{Context}_i. \tag{6}$$

The overhead $Overhead_{j,i}$ is the sum of $Config_j$ corresponding to each RZ_j and $Context_i$ (save and load) common for all tasks T_i. $Config_j$ corresponds to the configuration overhead to place RZ_j on the target technology. We compute this configuration overhead by making the floorplan of each RZ_j on the chosen device and by conducting the whole partial reconfiguration flow up to the creation of partial bitstream. According to the configuration frequency and the configuration port, the $Config_j$ is determined by

$$\mathrm{Config}_j = \frac{\text{size of bit stream}}{(\text{Configuration frequency} \times \text{configuration port width})}. \tag{7}$$

For example, the workload of RZ_1 resulting from T_1 and T_3 is 66%. The last column in Table 1 gives costs D of the other tasks not assigned to the RZ.

We notice an overload in RZ_2 caused by the workloads of execution of T_2 and T_4 as well as by their overheads. This overload is resolved during Step 3.

Decision of increasing the number of RZs: Step 3 takes place only when an overload within some RZs is detected in Step 2 and is achieved by Algorithm 2. Step 3 aims to lighten the overload in RZs by conducting the migration of task execution sections to nonoverloaded RZs before resorting to the solution of increasing the number of overloaded RZs. Hence, for each task, we search all the possible combinations of task execution sections. For each overloaded RZ, Algorithm 2 searches all the nonoverloaded RZs that could accept at least one of its assigned tasks; that is, $D \neq \infty$. Then, Algorithm 2 checks task by task the possibility of migration of an execution section or a combination of execution sections of the current task in order to reduce the overload of its RZ by respecting the workload of the nonoverloaded receiving RZ. In the worst case, the complexity of Algorithm 2 is O (M *N *NTM *TS), where M denotes the number of overloaded RZs, N is the number of nonoverloaded RZs, NTM is the maximum number of tasks assigned to an overloaded RZ and TS the maximum number of execution section combinations for a task assigned to an overloaded RZ.

Step 3 groups the workloads of overloaded RZs in *L1* (line 7) and the workloads of nonoverloaded RZs in *L2* (line 7). Step 3 goes throughout the RZs in *L1* to resolve their overloads independently (line 11). Step 3 uses nonoverloaded RZs in *L2* to lighten the workloads of RZs in *L1* (line 15). This step searches the nonoverloaded RZ in *L2* that gives finite cost D with at least one task assigned to the overloaded RZ during Step 2 (line 19). Once Step 3 finds the set of tasks that could be executed in the nonoverloaded RZ, it balances the workloads between both RZs by respecting the tasks' preemption points (line 22–line 37). If the overload persists in the RZ of *L1*, the algorithm decides adding other instances of this RZ up to workload of RZ?(line 42). When the processed nonoverloaded RZ_n do not affect the added number of overloaded RZ_m, Step 3 reinitializes their workloads to their values before dealing with the overload of RZ_m(line 43).

Without any loss of generality, our proposed strategy of resource management includes the main functions of generic placement: partitioning and fitting, which are fulfilled by the two following levels.

3.2. Level 2: Partitioning of Reconfigurable Physical Blocs on the Target Technology. Level 2 takes the types of RZs provided by level 1 as inputs and searches all the possible locations for them on the target device. These locations, called Reconfigurable Physical Blocs (RPB), are partitioned on the specified Reconfigurable Regions (RR) delimited in the target device. The RPBs are depicted by their RB-model as presented in

$$RPB_{i-}RB = \{\gamma_{i,k}\ RB_k\}, \quad \gamma_{i,k} \text{ is natural},$$
$$1 \leq i \leq NR, 1 \leq k \leq NP. \tag{8}$$

The RPBs must contain all the types of RBs required by the RZ type. The number of RBs in RPBs is greater than or equal to the number of RBs in RZs. Figure 3 shows an example of RPBs partitioned in RR_1 which are associated to an RZ requiring two RB_1 and one RB_3. The RPBs are presented by the five dotted rectangles.

3.3. Level 3: Two-level Fitting. Level 3 consists of two independent sublevels. The first sublevel ensures the fitting of

Table 1: Example of final results of Step 2.

RZ types	RZ resources	Assigned tasks T_i (D) and RZ workloads	Costs D $T_i(D)$
RZ_1	$\{25\ RB_1,\ 42\ RB_2\}$	T_1 (720), T_3 (140) **66%**	$T_2(\infty), T_4(\infty), T_5(\infty)$
RZ_2	$\{52\ RB_2,\ 12\ RB_4\}$	T_2 (1620), T_4 (0) **137%**	$T_1(\infty), T_3(\infty), T_5(\infty)$
RZ_3	$\{46\ RB_1, 53\ RB_2, 1\ RB_3, 1\ RB_4\}$	T_5 (0) **35%**	$T_1(2552), T_2(\infty)\ T_3(1972), T_4(\infty)$

```
$Load_m$: the load (%) of overloaded $RZ_m$
$Load_n$: the load (%) of nonoverloaded $RZ_n$
$Load_{n,i}$: the load (%) of nonoverloaded $RZ_n$ after adding a section of execution of $T_i$
$Section_i$: the list of possible execution sections of task $T_i$ determined by its preemption points
$Exe_i$: execution section of $T_i$
$p, q, r, j, i, l$: naturals
$L1$ = {loads of overloaded $RZ_j$}; $L2$ = {loads of nonoverloaded $RZ_j$}
$L3$: list of tasks
Sort $L1$ in descending order
Sort $L2$ in ascending order, in case of equality. Sort $L2$ in ascending order according to confi-guration overhead
for $p = 1$ to sizeof($L1$) do
  $RZ_m = L1(p)$
  $Load_m$ = load($RZ_m$)
  $q = 1$
  while $q \leq$ sizeof($L2$) and $Load_m > 100$ do
    $RZ_n = L2(q)$
    $Load_n$ = load ($RZ_n$)
    //Search $Ti$ from $RZ_m$ to migrate to $RZ_n$
    if$\exists$ $\{T_i\}$ assigned to $RZ_m/D(T_i, RZ_n) \neq \infty$ then
      Sort $\{T_i\}$ in ascending order according to $D(T_i, RZ_n)$ in $L3$
      $r = 1$
      while ($r \leq$ size of ($L3$)) and ($Load_m > 100$) do
        $\tau_i = L3(r)$
        $l = 1$
        // checking the possibility of relocation of the sections of $T_i$ by respecting the load of $RZ_n$
        while $l \leq$ size of ($Section_i$) and $Load_m > 100$ do
          Select the first execution section $Exe_i$ and discard it from $Section_i$
          $Load_{n,i} = Load_m + Exe_i/P_i + Overhead_{n,i}/P_i$
          if $Load_{n,i} \leq 100$ then
            // Migration of $Exe_i$ from $RZ_m$ to $RZ_n$ is accepted
            $Load_m = Load_m$-$Exe_i/P_i$-$Overhead_{m,i}/P_i$ // Removing $Exe_i$ from $RZ_m$
            $Load_n = Load_{n,i}$// Migration of $Exe_i$ to $RZ_n$
          end if
          $l$++
        end while
        $r$++
      end while
    end if
    $q$++
  end while
  if $Load_m > 100$ then
    New $RZ_m^*(\lceil Load_m/100 \rceil - 1)$// Adding new $RZ_m$
    Reinitialize the load of $\{RZ_n\}$ when it does not affect the number of added $RZ_m$
  end if
end for
```

Algorithm 2: Decision of increasing the number of RZs.

RZs on the most suitable nonoverlapped RPBs in terms of resource efficiency. The second sublevel performs the mapping of tasks to RZs according to their preemption points by respecting the workload of each RZ and guaranteeing the total execution of each task. Such mapping essentially promotes the solution giving the lowest overhead and lowest cost D. The second fitting sublevel provides an execution unit for each task; consequently, there is no longer the issue of task rejection. The mapping of tasks to RZs is strongly based on dynamic partial reconfiguration. This latter concept enables

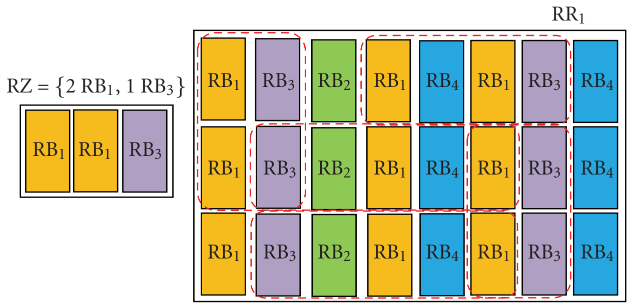


FIGURE 3: Example of partitioning RR_1 into RPBs.

multitasking as well as execution of several hardware tasks on the same RZs. In fact, dynamic partial reconfiguration allows the reconfiguration of subareas on the FPGA during runtime without affecting other running tasks.

4. Resolution of the Hardware Task Placement Problem

Previously, in our work presented in [23], we proposed a straightforward method to resolve partitioning and fitting. Nevertheless, with the rapid growth of resources in recent technologies and with the increasing complexity of applications, this exhaustive search is not efficient. In fact, the problem of partitioning/fitting is NP-complete, the search space is immense and the temporal complexity of the execution of our proposed algorithm in [23] is exponential. In this paper, we formulate the partitioning/fitting problem as a constrained optimization problem. Our work is based on the smart nonexhaustive complete method called Branch and Bound [24] which employs efficient techniques for scanning search space and extracting the optimal solution.

4.1. Formulation of Hardware Task Placement as a Constrained Optimization Problem. The problem of partitioning/fitting is modeled as a constrained combinatory optimization problem as it uses discrete solution set, chooses the best solution out of all possible combinations and aims the optimization of multicriteria function. Partitioning/fitting problem is mixed integer problem as it uses some natural and binary variables. This problem is described by the quadruplet (Constants, Variables, Constraints, and Objective Function).

4.1.1. Constants. *NT:* number of tasks constituting the application, *NZ*: number of RZs resulting from level 1 (Offline flow of hardware task classification), *NP*: number of RB types existing in the target technology, D (T_i, RZ_j): the cost D between T_i and RZ_j, $RBCost_k$: the cost of each RB type.

Device Features. Device_Width: the width of the device, *Device_Height*: the height of the device, *Device_RB*: the RB-model of the device.

Task features. C_i: the Worst Case Execution Time (WCET) of T_i, P_i: the period of T_i, $Preemp_{i,l}$: the preemption point l of T_i, $NbrPreemp_i$: the number of preemption points in T_i, $Context_i$: the functional overhead for preempting and resuming T_i, $Overhead_{j,i}$: the full overhead for execution of task T_i on RZ_j, T_i_RB: the RB-model for T_i.

RZ features. RZ_j_RB: the RB-model for RZ_j, $Config_j$: the configuration overhead for RZ_j in target technology.

4.1.2. Variables

RPB_j Features for Each RZ_j. X_j: the abscissa of the upper left vertex of RPB_j, Y_j: the ordinate of the upper left vertex of RPB_j, $WRPB_j$: the abscissa of the upper right vertex of RPB_j, $HRPB_j$: the ordinate of the bottom left vertex of RPB_j, RPB_j_RB: the RB-model of the RPB_j constructed by the above coordinates.

The Task Preemption Points. Under the task functional model, it is assumed that the preemption points*l*of each task T_i are known. $PreempUnicity_{j,i,l}$: Boolean variable controls whether the mapping of $Preemp_{i,l}$ of T_i is performed on RZ_j. It is equal to 1 when $Preemp_{i,l}$ is mapped to RZ_j. $PreempTask_{j,i,l}$: each preemption point assigned to RZ_j is taken from the predefined preemption points of each task(see(9)).

$$\text{PreempTask}_{j,i,l} = \text{Preemp}_{i,l} \times \text{PreempUnicity}_{j,i,l}$$
$$\forall\ 1 \le i \le NT, \quad 1 \le l \le N\text{brPreemp}_i, 1 \le j \le \text{NZ}. \tag{9}$$

$SumPreemp_{j,i}$: The sum of preemption points of T_i performed within $RZ_j$is expressed by

$$\text{SumPreemp}_{j,i} = \sum_{\substack{1 \le l \le Nbr\text{Preemp}_i \\ \text{PreempUnicity}_{j,i,l} \ne 0}} 1$$
$$\forall\ 1 \le i \le NT, \quad 1 \le j \le NZ. \tag{10}$$

$OccupationRate_{j,i}$: The full occupation rate of T_i in RZ_j resulting from the mapping of preemption points of T_i to RZs. The occupation rate is computed as expressed in

$$\text{OccupationRate}_{j,i} = \begin{cases} \displaystyle\sum_{\substack{1 \le l < Nbr\text{Preemp}_i \\ \text{PreempUnicity}_{j,i,l} \ne 0}} \left(\text{Preemp}_{i,l+1} - \text{Preemp}_{i,l}\right) + \left(C_i - \text{Preemp}_{i,Nbr\text{Preemp}_i}\right), & \text{if PreempUnicity}_{j,i,Nbr\text{Preemp}_i} \ne 0 \\ \displaystyle\sum_{\substack{1 \le l < Nbr\text{Preemp}_i \\ \text{PreempUnicity}_{j,i,l} \ne 0}} \left(\text{Preemp}_{i,l+1} - \text{Preemp}_{i,l}\right), & \text{else.} \end{cases} \tag{11}$$

AverageLoad: The average of RZ workloads obtained after task mapping is calculated by

$$\text{AverageLoad} = \left(\sum_{\substack{1\le i\le NT\\1\le j\le NZ}} \left(\text{OccupationRate}_{j,i}/P_i + \text{SumPreemp}_{j,i} \times \text{Overhead}_{j,i}/P_i\right)\right)/NZ. \tag{12}$$

4.1.3. Constraints

Heterogeneity Constraint. The RZs must be fitted on RPBs containing a sufficient number of their required types of RBs. This constraint must be respected during partitioning and fitting of RZs. The heterogeneity constraint is formulated by

$$\beta_{j,k} \le \sum_{\substack{X_j\le m\le WRPB_j\\Y_j\le n\le HRPB_j}} \sum_{\text{device\_}RB[m][n]=RB_k} 1, \quad \forall\ 1\le j\le NZ, 1\le k\le NP \tag{13}$$

$$RZ_{j_{RB}} = \left\{\beta_{j,k}\ RB_k\right\}, \quad 1\le j\le NZ,\ 1\le k\le NP.$$

Nonoverlapping between RPBs. As expressed by (14), this constraint restricts the fitting of RZs on nonoverlapped RPBs

$$X_q > WRPB_j \text{ or } X_j > WRPB_q \text{ or } Y_q > HRPB_j \text{ or } Y_j > HRPB_q \quad \forall\ j\ne q, 1\le j,q\le NZ. \tag{14}$$

Nonoverload in RZs. As mentioned in (15), the non-overload in RZs must be respected during mapping of tasks

$$\sum_{1\le i\le NT} \left(\text{OccupationRate}_{j,i}/P_i + \text{SumPreemp}_{j,i} \times \text{Overhead}_{j,i}/P_i\right) \le 100\%, \forall\ RZ_j. \tag{15}$$

Infeasibility of Mapping for Preemption Points. This constraint prohibits the mapping of preemption points of tasks to RZs giving infinite cost D (see (16)).

$$\text{PreempUnicity}_{j,i,l} = 0 \text{ when } D\left(T_i, RZ_j\right) = \infty, \quad \forall\ 1\le i\le NT, \quad 1\le l\le Nbr\text{Preemp}_i, 1\le j\le NZ. \tag{16}$$

Uniqueness of Preemption Points. This constraint claims that each preemption point l of T_i must exist on unique RZ_j and guarantees the achievement of task execution as well as the elimination of task rejection (see (17))

$$\sum_{1\le j\le NZ} \text{PreempUnicity}_{j,i,l} = 1, \quad \forall\ 1\le i\le NT, \quad 1\le l\le Nbr\text{Preemp}_i. \tag{17}$$

Domains of RPB Coordinates. Equation (18) defines the allowed domain of values that can be assigned to RPB coordinates during partitioning

$$1\le X_j, WRPB_j \le \text{Device\_Width}, \quad 1\le Y_j, HRPB_j \le \text{Device\_Height}. \tag{18}$$

4.1.4. Minimization Objective Function (F). As with all optimization problems, we have defined a minimization objective function F that helps in selecting the optimal solution. As described by (19), F promotes a solution giving the best values for the two objective subfunctions *MappingFunction* and *PlaceFunction*

$$F = \text{MappingFunction} + \text{PlaceFunction}. \tag{19}$$

PlaceFunction focuses on the subproblem of fitting RZs on the most suitable RPBs after partitioning the target device by respecting the predefined constraints. In (20), *PlaceFunction* evaluates the efficiency of resources after fitting RZs on the selected RPBs. *PlaceFunction* promotes the fitting of RZs on the RPBs that strictly contain the number and type of RBs required by RZs

$$\text{PlaceFunction} = \sum_{\substack{1\le j\le NZ\\1\le k\le NP}} RB\text{Cost}_k \times \left(\gamma_{j,k} - \beta_{j,k}\right),$$
$$RPB_j\text{\_}RB = \left\{\gamma_{j,k}\ RB_k\right\}, RZ_j\text{\_}RB = \left\{\beta_{j,k}\ RB_k\right\}, \quad 1\le j\le NZ, 1\le k\le NP. \tag{20}$$

MappingFunction focuses on the subproblem of fitting hardware tasks on RZs by respecting the predefined preemption points. *MappingFunction* targets the full exploitation of RZs and their workloads balance. It also aims at mapping tasks to the RZs providing the lowest cost D to optimize resource utilization. Moreover, *MappingFunction* promotes the solution that minimizes the overall overhead and number of task relocations. Thus, *MappingFunction* is expressed by four subfunctions targeting these goals

$$\text{MappingFunction} = Map1 + Map2 + Map3 + Map4. \tag{21}$$

Map1 focuses on the RZ workloads by means of (22). Its first expression evaluates whether the RZs are fully exploited. While minimizing this expression, the RZ workloads approach 100% of their exploitation. Its second expression computes the variance of the RZ workloads towards the obtained average workload. Minimization of this second expression ensures load balancing between RZs

$$Map1 = \sum_{1\le j\le NZ}\left(100\% - \left(\sum_{1\le i\le NT}\mathfrak{A}\right)\right) + \sum_{1\le j\le NZ}\frac{\left(\left(\sum_{1\le i\le NT}\mathfrak{A}\right) - \text{Average Load}\right)^2}{NZ}. \tag{22}$$

Where

$$\mathfrak{A} = \frac{\text{OccupationRate}_{j,i}}{P_i} + \frac{\text{SumPreemp}_{j,i}\times\text{Overhead}_{j,i}}{P_i} \tag{23}$$

Map2 computes the overhead in (24) resulting from the mapping of task preemption points to the RZs. *Map2* takes into account all the possible preemption points, even when two successive preemption points of one task are mapped to the same RZ, for obtaining the worst case overhead. In fact, the scheduler could preempt a task on these successive preemption points in the same RZ in favor of a higher priority task. Minimizing *Map2* promotes the solutions that map the tasks to the RZs providing the lowest configuration overhead

$$Map2 = \sum_{\substack{1\le i\le NT\\1\le j\le NZ}}\text{SumPreemp}_{j,i}\times\text{Overhead}_{j,i}. \tag{24}$$

By minimizing *Map3* in (25), we promote a solution that maps tasks by a high occupation rate to the RZs giving the lowest cost D. The benefit of this minimization is optimizing the use of available resources in the technology. Indeed, these D costs between tasks and RZs reveal the rate of resource waste. Moreover, these D costs consider the weight of each resource in terms of three parameters which are: its frequency on the technology, the importance of its functionality, and its power consumption. Our objective is to minimize the utilization of these costly resources whenever possible. Hence, the more we promote mapping of tasks with high occupation rates to the RZs giving the lowest cost D, the more we optimize resource utilization

$$Map3 = \sum_{\substack{1\le i\le NT\\1\le j\le NZ}} D\left(T_i, RZ_j\right)^2\times\text{OccupationRate}_{j,i}{}^2/4, - D\left(T_i, RZ_j\right)\times\text{OccupationRate}_{j,i}. \tag{25}$$

Map4 computes in (26) the total number of task relocations obtained after such mapping. Although the migration of tasks between RZs solves the conflicts between tasks on the same RZ, minimizing the number of migrations optimizes the number of preemptions and overhead

$$Map4 = \sum_{\substack{1\le i\le NT\\1\le j\le NZ}}\ \sum_{\substack{1\le l<Nbr\text{Preemp}_i\\ \text{PreempUnicity}_{j,i,l}\neq 0,\ \text{PreempUnicity}_{j,i,l+1}=0}} 1 \tag{26}$$

4.2. Branch and Bound Method for Solving the Hardware Task Placement. Our work is based on the global optimization method called Branch and Bound during partitioning of resource space and fitting RZs and tasks. The method of branch and Bound consists in enumerating all the possible solutions in an intelligent way by relying on the features of the specified problem. This technique succeeds in eliminating all the partial solutions that do not lead to the optimal solution. The Branch and Bound method relies on a predefined bound function which is our objective function F to make boundary on solutions to be excluded or to be kept as potential solutions. The performance of the Branch and Bound method depends on the quality of this function. Branch and Bound is adapted to mixed integer linear and non-linear programming.

Initially only one subset of solutions exists namely, the root subset that contains all the solutions for the problem. The unexplored subsets are represented as nodes in a dynamically generated search tree. Each iteration on Branch and Bound processes one such node. The iteration has three components: the selection of the node to process, branching and the bound function calculation of the ramified nodes. For each node, the bound function calculation considers the best fitting for the remaining RZs or tasks without constraints. Whereas, only nodes respecting predefined constraints are only ramified after node selection. Our strategy for selecting the next node is based on the value of the bound function F of the node. We start by the node giving the best bound function F at the current level and we use the strategy of *Depth First Search* (*DFS*). This strategy starts by exploring, at each level, the node giving the best F until obtaining the complete solution which gives the last best F. Then, the process is repeated for the next best node on the last visited level if its F does not exceed the last best F. In this case, we branch this node and compute the bound functions of its ramified nodes and compare them to the last best F. If F of these partial solutions is greater than or equal to the last best F, they are rejected. If not, they are kept and the best partial solution is selected to be processed by Branch and Bound. Once a new complete solution is founded, the method checks whether its F optimizes the last best F. When an optimization is detected, the last best F is updated. The process is repeated for all the next best nodes in each level as described above. Algorithm 3 describes the Branch and Bound method applied on our problem.

Partitioning resource space and fitting RZs can be resolved independently from task mapping. In the subproblem of partitioning resource space and fitting RZs, *PlaceFunction* is employed as the bound function F in Algorithm 3. While in the task mapping subproblem, F corresponds to *MappingFunction*. To fit RZs on their suitable RPBs, the two subproblems of partitioning the RPBs on the target device and fitting RZs on the selected RPBs are performed in parallel. In fact, the resolution starts by partitioning the RPBs corresponding to the RZ root (for example RZ_1) with respect to the heterogeneity constraint. This partitioning is done randomly by assigning all potential values to the RPB coordinates and the RZ root is fitted onto the RPB that gives the best *PlaceFunction*. After selection

```
c: narural // the counter on branched nodes
b: natural // the number of branched nodes provided by the current node
Best $F = +00$,
Live = $\{(\text{Node in root level}, F(\text{node in root level}))\}$// the set of nodes to be processed
while Live $\neq \oslash$ do
    Select the node $N$ from Live to be processed which gives the best $F$ by using DFS
    Live = Live \ $\{(N, F(N))\}$
    if $F(N) = F(X)$ for a feasible complete solution $X$ and $F(X) <$ Best $F$ then
        Best $F = F(X)$
        Solution =$X$ // A complete solution is founded
    else if $F(N) \geq$ Best $F$ then
        Discard $N$ from processing // partial candidate is rejected
    else
        // partial candidate is kept
        Branch on $N$ generating $N1, \ldots, Nb$ by respecting the problem constraints
        for $c = 1$ to $b$ do
            Bound $Nc$: compute $F(Nc)$ // bound calculation for the node $Nc$
            Live = Live U $\{(Nc, F(Nc)\}$
        end for
    end if
end while
Optimal Solution = Solution, Optimal Value = Best $F$
```

Algorithm 3: Branch and Bound for hardware task placement [24].

of the best RPB for the RZ root that is, the selected node, this node is branched into all the possible RPBs for RZ_2 by respecting the predefined constraints and the *PlaceFunction* of each ramified node is computed. The first selected node for the RZ_2 level must satisfy the heterogeneity constraint as well as the nonoverlapping constraint. The subset of solutions ramified from this first selected node in RZ_2 level is explored by searching the RPBs of RZ_3. The resolution is carried out similarly until the achievement of exploration branched from the first selected node which gives the last best *PlaceFunction*. The process is repeated for the next best nodes and recursively, the remainder RZs are fitted on RPBs as described for the RZ root by respecting the predefined constraints. During exploration, if a partial solution gives *PlaceFunction* worse than the last best *PlaceFunction*, this partial solution is rejected and its branching is stopped. If not, this partial partitioning/fitting of RZs is kept as a potential partial solution. During this repetitive process, all possible combinations of RPBs respecting the constraints are tested, when the process is achieved, the optimal solution is extracted.

Figure 4 illustrates the running of the Branch and Bound method on the partitioning/fitting of four RZs. The nodes with an X mark present the subsets of solutions that do not contain the optimal solution and with a $\surd$ mark present potential solutions. In the *RZroot level* and *RZ_2 level*, the best node is selected, branched on partial solutions and the *PlaceFunction* of its ramified nodes are computed. As can be noticed, there are two processed best fittings in the *RZ_3 level*. The method starts by the first best node that is, *PlaceFunction3p* and processes this selected node. It is branched into two nodes by the RPBs of RZ_4. After computing the *PlaceFunction* for these ramified nodes, the RPB_1-RZ_4 is selected and a complete solution is obtained. The last best *PlaceFunction* is *PlaceFunction41*. RPB_2-RZ_4 is rejected as it does not optimize *PlaceFunction41*. The next best node in the last visited level not yet processed is RPB_1-RZ_3. The node is kept and branched into two RPBs on the *RZ_4 level*. *PlaceFunction43* optimizes the last best *PlaceFunction* and the solution is complete, thus the optimal solution is obtained by *PlaceFunction43*. The next iteration processes the next best node in the last visited level; partial solution RPB_2-RZ_3. This partial solution is rejected as *PlaceFunction32* exceeds the last best *PlaceFunction*. The other nodes are also not processed as their partial bound function exceeds the last best *PlaceFunction*. Finally, the optimal solution of partitioning/fitting of RZs is obtained by fitting *RZroot* (RZ_1) on RPB_i-*RZroot*, RZ_2 on RPB_2-RZ_2, RZ_3 on RPB_1-RZ_3, and RZ_4 on RPB_3-RZ_4.

The resolution of mapping starts by randomly assigning the preemption points of task root (T_1) to RZs giving finite cost D and computing the bound function of each node. The selected node is the node that gives the most optimal mapping of preemption points according to *Mapping Function.* This node is ramified for new processing for the next task. This ramification must respect the total execution of tasks as well as the non-overload on RZs. Recursively, as task root, the remainder tasks are mapped by satisfying the two previous constraints and computing the *Mapping Function* of their ramified nodes. When a complete solution is obtained, it represents the last best *Mapping Function*. Similarly to partitioning/fitting of RZs, the mapping is resolved in a recursive manner by computing the *Mapping Function* of each partial mapping branched from the selected best nodes and keeping only the ones optimizing the last best *Mapping Function*. The mapping resolution is finished when all the

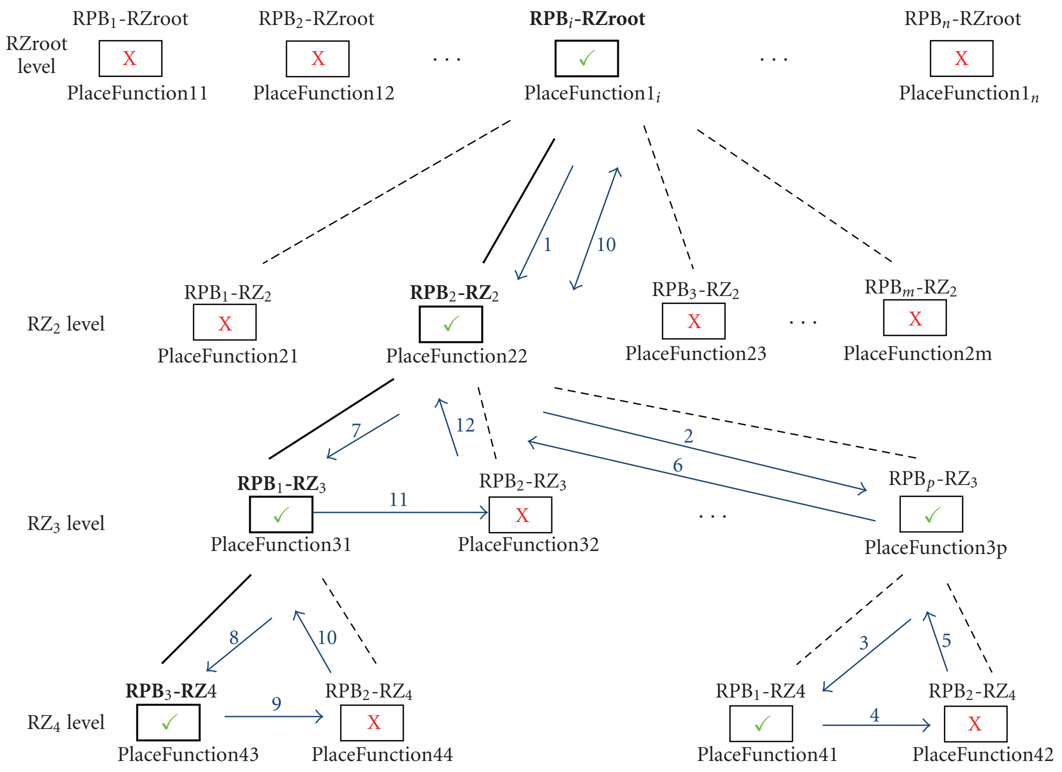


Figure 4: Branch and Bound for partitioning/fitting of RZs.

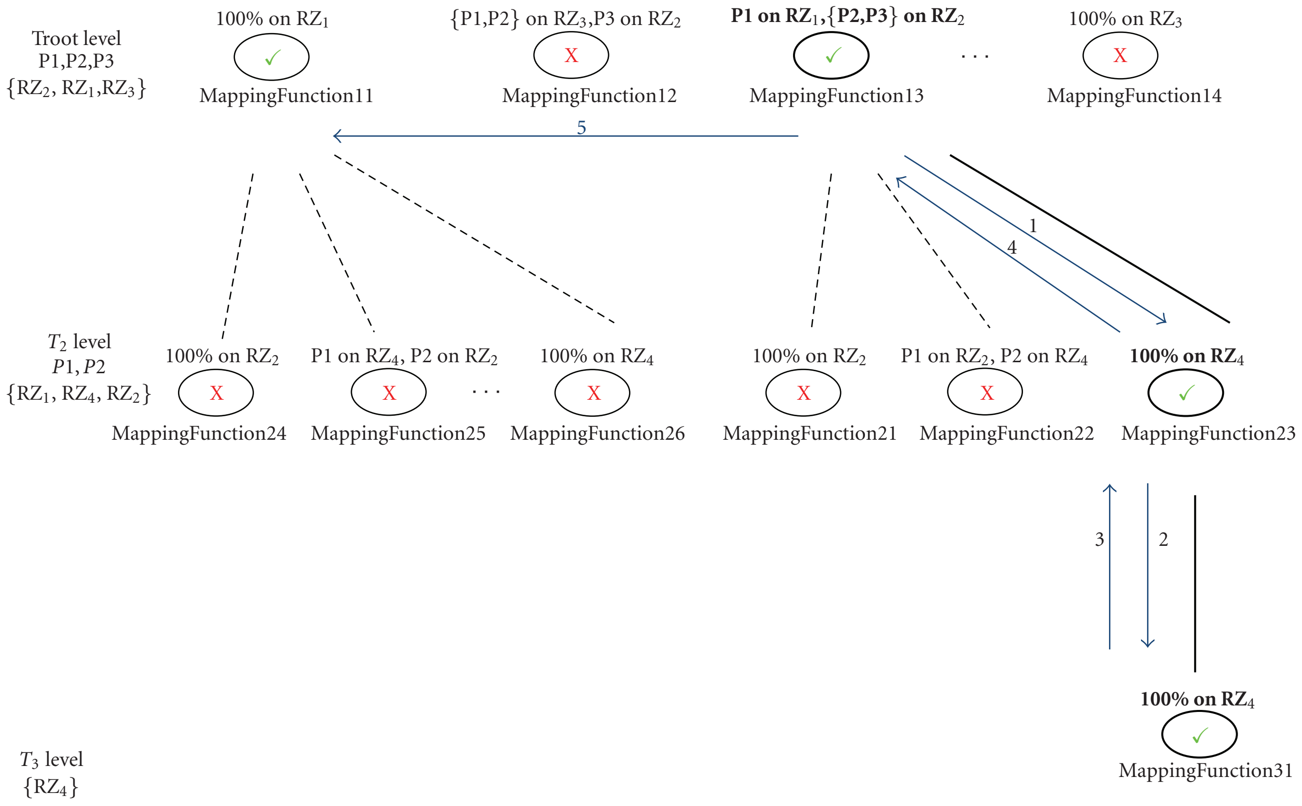


Figure 5: Branch and bound for fitting tasks.

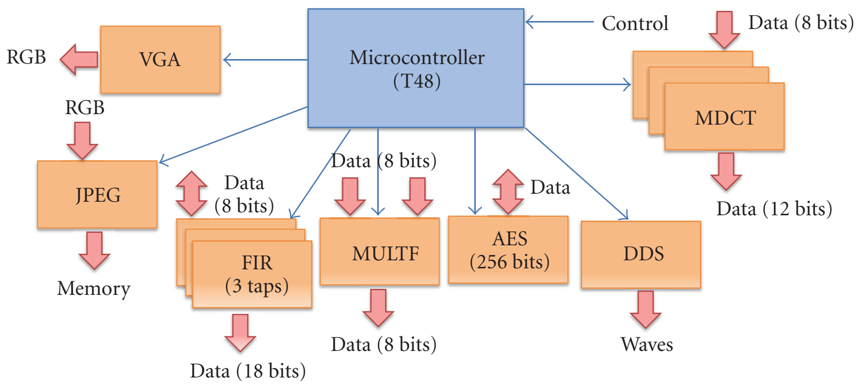


FIGURE 6: Hardware tasks of the application.

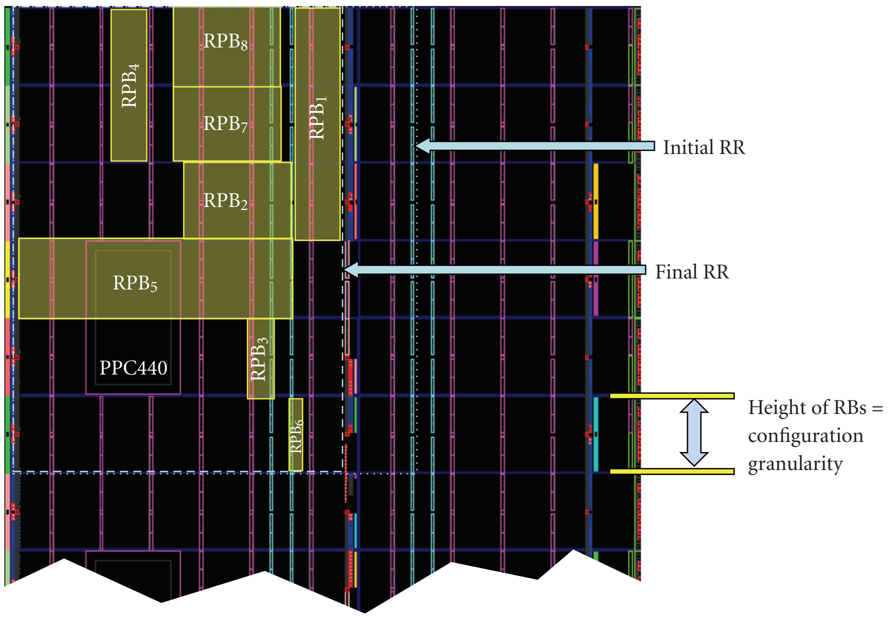


FIGURE 7: Floorplanning of RPBs on Virtex 5 FX200.

potential mappings of preemption points for all tasks to RZs respecting mapping constraints are checked.

Figure 5 describes the progress of mapping three tasks to four RZs. The set of RZs giving finite cost D is provided for each task. In *Troot level*, the resolution selects the first best node that is, *Mapping Function 13*, after branching it and after computing the *Mapping Function* of the partial solutions, the next iteration is released on T_2 *level*. In T_2 *level*, *MappingFunction23* is the best partial solution, its node is ramified on T_3 *level* and the last best *Mapping Function* is obtained by *MappingFunction31*. The process is repeated for the next best nodes in the last visited level. The nodes *MappingFunction21 and MappingFunction22* are not processed since they exceed the last best *Mapping Function*. The method processes the next best node in the last visited level which is *MappingFunction11*. The branching of this node does not optimize the last best *MappingFunction*. Thus, its ramified nodes are not explored in T_2 *level*. Finally, the optimal solution of mapping hardware tasks to RZs is obtained by fitting the first preemption point P1 of T_1 on RZ_1 and its remaining preemption points P2 and P3 on RZ_2 and by fitting all the preemption points of T_2 and T_3 on RZ_4.

As well-known, in the worst case, the complexity of the classical Branch and Bound method is exponential [25]. In fact, in the worst case, the complexity of the first subproblem: partitioning/fitting RZs is equal to O $((4{*}B{+}8{*}B^2{+}(NZ\ \ {+}1){*}B^2)^{NZ-1}{+}4{*}B)$ where B denotes a possible combination of RPB coordinates for a given RZ as presented by a node in Figure 4. Thus, B is a possible value assignment for the decision variables X_j, $WRPB_j$, Y_j, $HRPB_j$. The values domain of B is equal to $|\text{domain}_{Xj}|^*|\text{domain}_{WRPBj}|^*|\text{domain}_{Yj}|^*|\text{domain\_}HRPBj|$, which is equal to $(Device\_Width)^{2*}$ $(Device\_Height)^2$. $4{*}B$ depicts the selection of a node, its removal from the set live, and both following test: the test of final and optimal solution (line 8) and the test of failed solution (line 11). $8{*}B^2$

Table 2: Hardware task features.

Instances		RBs	WCET(μs)	Period (μs)	Configuration overhead (μs)	Preemptio points (μs)
MDCT	$\{T_1, T_9, T_{10}, T_{11}, T_{12}\}$	$\{2\ RB_1, 12\ RB_2, 3\ RB_3, 0\ RB_4\}$	40552	416666	1856	10000, 20000, 30000
AES	$\{T_2\}$	$\{4\ RB_1, 7\ RB_2, 1\ RB_3, 1\ RB_4\}$	51540	100000	2185	30000, 40000
DDS	$\{T_3\}$	$\{0\ RB_1, 1\ RB_2, 1\ RB_3, 1\ RB_4\}$	5000	12000	432	none
T48	$\{T_4\}$	$\{5\ RB_1, 4\ RB_2, 0\ RB_3, 0\ RB_4\}$	20000	50000	605	800, 10600, 16300
JPEG	$\{T_5\}$	$\{8\ RB_1, 12\ RB2, 0\ RB_3, 2\ RB_4\}$	350000	416666	2421	200000, 300000
MULTF	$\{T_6\}$	$\{1\ RB_1, 1\ RB_2, 0\ RB_3, 1\ RB_4\}$	5600	10000	491	1400, 2350, 4020, 5170
FIR	$\{T_7, T_{13}, T_{14}\}$	$\{0\ RB_1, 1\ RB_2, 0\ RB_3, 1\ RB_4\}$	300	2000	112	120, 210, 255
VGA	$\{T_8\}$	$\{2\ RB_1, 4\ RB_2, 1\ RB_3, 0\ RB_4\}$	5000	10000	681	1650, 2700

Table 3: The results obtained for Step 1 and Step 2 of Level 1.

	MDCT (T_1,T_9,T_{10},T_{11},T_{12})	AES (T_2)	DDS (T_3)	T48 (T_4)	JPEG (T_5)	MULTF (T_6)	FIR (T_7,T_{13},T_{14})	VGA (T_8)
$RZ_1\{2\ RB_1, 12\ RB_2, 3\ RB_3, 0\ RB_4\}$**57%**	**0**	∞	∞	∞	∞	∞	∞	1024
$RZ_2\{4\ RB_1, 7\ RB_2, 1\ RB_3, 1\ RB_4\}$**338%**	∞	**0**	560	∞	∞	**732**	752	**620**
$RZ_3\{0\ RB_1, 1\ RB_2, 1\ RB_3, 1\ RB_4\}$**45%**	∞	∞	**0**	∞	∞	∞	192	∞
$RZ_4\{5\ RB_1, 4\ RB_2, 0\ RB_3, 0\ RB_4\}$**44%**	∞	∞	∞	**0**	∞	∞	∞	∞
$RZ_5\{8\ RB_1, 12\ RB_2, 0\ RB_3, 2\ RB_4\}$**85%**	∞	∞	∞	1380	**0**	1360	1380	∞
$RZ_6\{0\ RB1, 1\ RB2, 0\ RB3, 1\ RB4\}$**112%**	∞	∞	∞	∞	∞	∞	**0**	∞

is the branching operation by respecting the constraints (line 15) and $(NZ+1)^*B^2$ is the bound computation for the branched nodes (line 17) by assigning each already processed RZ to its RPB and the remaining ones to the most suitable RPBs in terms of resource efficiency without constraints and the insertion of this branched nodes into the set live (line 18). Consequently, the complexity of the subproblem in the worst case is $\sim$ O $(((Device\_Width)^{4*}(Device\_Height)^4)^{NZ})$. In the worst case, the complexity of the second subproblem: task fitting is O $((4^*B+B^{2*}\ (2+NT+6^*NT\ ^*NZ))^{NT-1}+4^*B)$ where B denotes a possible fitting of preemption points on RZs RZ_j for a given task as associated to a node in Figure 5. Thus, B is a possible value assignment for the decision variables $PreempUnicity_{j,i,l}$. The values domain of B for a given task T_i is $|$domainPreempUnicity$j, i, l|$ which is equal to: $|\{0,1\}|^{\mathrm{MaxPreemp}^*NZ} = 2^{\mathrm{Max\,Preemp}^*NZ}$ and we consider the maximum number of preemption points is *MaxPreemp*. After the branching of the selected node, we compute the bound of its branched candidate by considering that the remaining tasks which are not yet processed are mapped with 100% to their optimal RZs in terms of distance and configuration overhead. Consequently, in the worst case, the complexity of the second sub-problem by using the Branch and Bound search is $\sim O(2^{\mathrm{MaxPreemp}^*NT^*NZ})$. Thus, in the worst case, the search by Branch and Bound algorithm grows exponentially with *NT* and *NZ*. This complexity is expected as the placement of hardware tasks on the heterogeneous reconfigurable devices is NP-complete problem. Currently, many enhancements on Branch and Bound algorithm are conducted as in [26] which could reduce the exponential complexity of some problems to logarithmic complexity. But despite this possible exponential complexity, compared to an exhaustive exact method, the Branch and Bound method immensely lightens the search space. Effectively, the search tree branches are discarded when the current bound function exceeds the last best bound function. The Branch and Bound ensures complete resolution of placement problem with a performance better than or equal to that of complete exhaustive method in terms of resolution speed and storage space. With this smart method, we ensure a full combination of RZ fittings as well as task mapping which solves the problem of task rejection confronted in the previous works of placement. Unlike the approached methods such as metaheuristics [27], this method is complete and affords an optimal solution. In fact, within these metaheuristics, the computation in each generation is rather complicated and time consuming and the number of generations must be increased to ensure a more favorable solution. The quality of the obtained solution in metaheuristics is conditioned by the best choice of the initial generation.

5. Application and Results

To investigate the influence of our proposed method for hardware task placement, we implemented an application composed of several hardware tasks taken from opencores [28]. We have chosen examples of hardware tasks that are frequently used in recent embedded systems performing video and audio applications. Our application features hardware tasks of varied sizes and of heterogeneous resources. The hardware tasks are guided by an application manager; the microcontroller. We do not consider the control overhead between the microcontroller and the other hardware tasks. During the design of this application, we synthesized the resources of each hardware task by the ISE 11.3 Xilinx tool. We defined the configuration overheads of the obtained RZs by performing the partial reconfiguration flow by means of the Planahead 11.3 Xilinx tool.

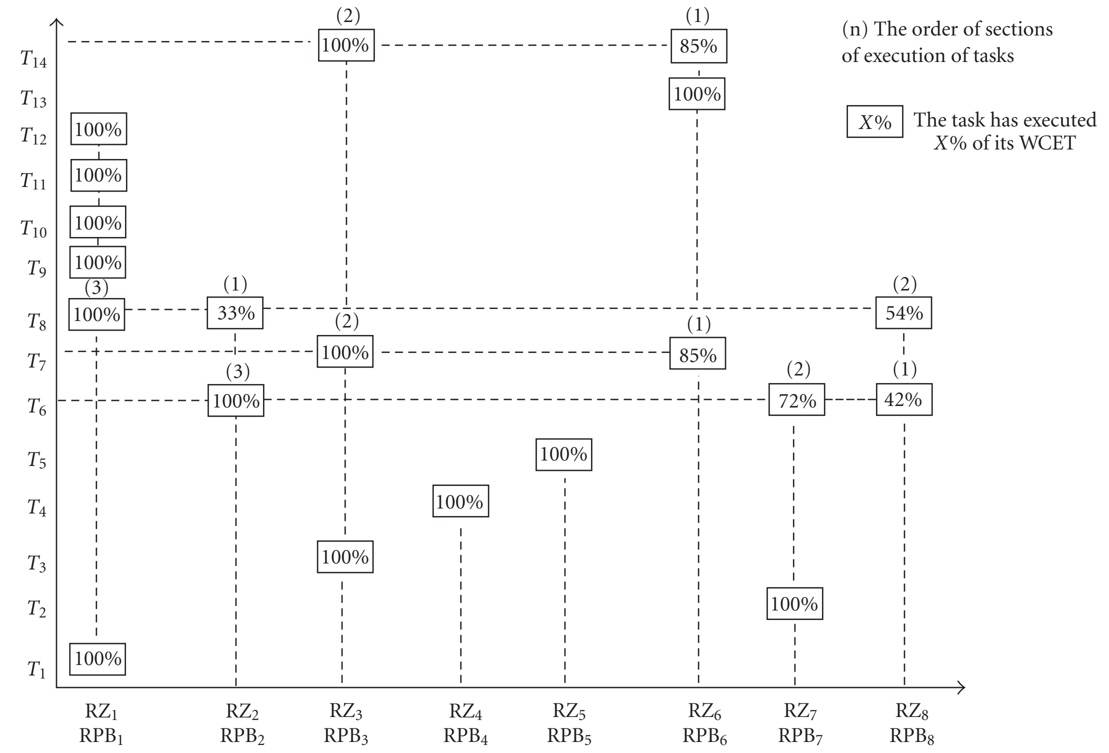


Figure 8: Resolution of fitting tasks on RZs.

Table 4: Coordinates of obtained RPBs.

	X_j	WRP_j	Y_j	$HRPB_j$
RPB_1 for RZ_1	40	45	1	3
RPB_2 for RZ_2	25	38	3	3
RPB_3 for RZ_3	33	36	5	5
RPB_4 for RZ_4	14	18	1	2
RPB_5 for RZ_5	1	39	4	4
RPB_6 for RZ_6	39	40	6	6
RPB_7 for RZ_7	24	37	2	2
RPB_8 for RZ_8	24	37	1	1

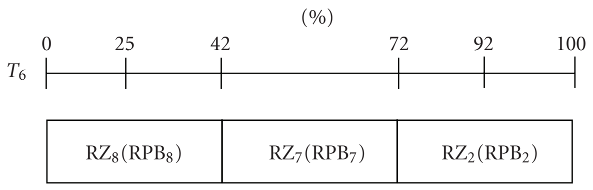


Figure 9: Mapping of preemption points of T_6.

5.1. Application. Figure 6 shows the hardware tasks included in the application. The core of the application is the microcontroller T48. The microcontroller guides the other hardware tasks either for speeding up the computing such as FIR and MULTF or for performing complicated processing such as JPEG compression. Consequently, there are three categories of hardware tasks.

(i) Application manager.

 (a) Microcontroller T48: Hardware task configuration and data flow synchronization.

(ii) Fine-grained hardware tasks (for speeding up microcontroller).

 (a) FIR: Performs 1000 FIR (Finite Impulse Response) filters. Each FIR features 3 taps, 8bit-input data and 48bit-coefficients.

 (b) MULTF: Performs 1000 floating point multiplications between two vectors of 8 bits. The exponent precision and mantissa precision are 3 and 4 respectively. These multiplications are pipelined with the latency of 8 clock cycles.

Table 5: Comparison of number of RBs between RZs and RPBs.

	RB_1	RB_2	RB_3	RB_4	$\triangle$
RPB_1	3	12	3	0	1 RB_1
RPB_2	4	7	2	1	1 RB_3
RPB_3	0	2	1	1	1 RB_2
RPB_4	6	4	0	0	1 RB_1
RPB_5	8	12	3	2	3 RB_3
RPB_6	0	1	0	1	0
RPB_7	4	7	2	1	1 RB_3
RPB_8	4	7	2	1	1 RB_3

Table 6: *Placement results comparison.*

Placement Methods	Task rejection	Resource utilization	Configuration overhead	Complexity
3-level offline placement –RR composed of 55 ∗ 6 of heterogeneous RBs –14 heterogeneous hardware tasks –Configuration frequency: 100 MHz	0	36%	11%	$O(((Device\_Width)^4 \times (Device\_Height)^4)^{NZ})$ $+O(2^{MaxPreemp \times NT \times NZ})$
Metaheuristics				
Simulated annealing [4] –Device cells: 100 × 100 –100 tasks –Task size: 17 × 17	13%	—	—	—
Genetic algorithm [10] –Device cells: 64 × 64 –Sets of 3,000 tasks –Task size: 32 × 32	45%	60%–80%	—	—
Heuristics				
KAMER-BF [4] –Device cells: 100 × 100 –2048-16384 tasks –Task size: 17 × 17	13%–18%	—	—	$O(n\log(n))n$: number of tasks on the device
Blind compaction [1] –Device cells: 64 × 64 –10.000 tasks –Task size: 32 × 32	—	60%	—	$O(n^2)n$: number of tasks on the device
IM [8] –Device cells: 100 × 100 –13 task sets each of 1000 tasks –Task size: 2 × 2 − 20 × 20	10%	—	—	—
Configuration Reuse + Configuration Prefetch [11] –Device cells: 4 tiles –JPEG decoder, MPEG encoder –Configuration frequency: 50 MHz	—	—	5% (JPEG), 18% (MPEG)	$O(NT\ x\ N) + O(N^2)NT$: number of tiles N: number of threads
LFD [32] –Virtex II Pro –2 JPEG + 1 MPEG –Configuration frequency: 100 MHz	—	—	8%	—

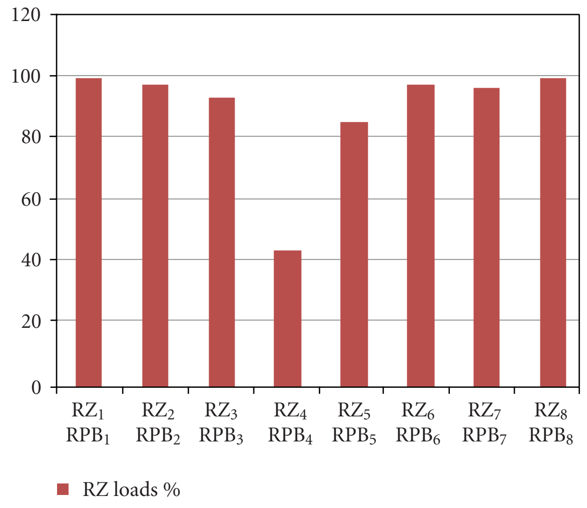


FIGURE 10: RZ workloads after task mapping.

(iii) Coarse-grained hardware tasks.

(a) MDCT: Computes Modified Discrete Cosine Transform.

(b) AES: (Advanced Encryption Standard): Performs encryption of blocs of 128 bits with 256bit-key.

(c) DDS: (Direct Digital Synthesizer): Creates sinusoidal waves programmable with frequency and phase on-time.

(d) JPEG: Performs hardware compression of 24 frames per second.

(e) VGA: Drives VGA monitors with an 800x600 resolution, it can display one picture on the screen either of chars or color waveforms or color grid.

The features of hardware tasks and their instances are presented in Table 2. These hardware tasks are characterized by considering the resource area in Virtex 5 technology [29]. Virtex 5 contains four main types of resources CLBL, CLBM, BRAM, and DSP. The RBs are vertical stacks composed of the same type of resources and match the reconfiguration granularity. Hence, RB_1 is 20 CLBMs, RB_2 is 20 CLBLs, RB_3 is 4 BRAMs, and RB_4 is 8 DSPs. We have assigned 20, 80, 192, and 340 as *RBCost,* respectively, for RB_1, RB_2, RB_3, and RB_4. Configuration overhead is determined as described in (7) by considering that each task defines an RZ. After synthesizing hardware tasks by the ISE tool, they are modeled under their RB-model reported in [3]. The partial reconfiguration flow dedicated by the Planahead tool enables the floorplanning of hardware tasks on the chosen device to create their bitstreams independently for estimating configuration overheads. The estimation of configuration overheads considers the best case fitting of each task, as we believe the subproblem of partitioning/fitting RZs is solved efficiently and independently from the subproblem of task mapping. We rely on parallel 8bit-width configuration ports and use 100 MHz as the configuration clock frequency. Preemption points are determined arbitrarily according to the granularity of hardware tasks and their WCET. For all tasks, we consider the first preemption point is equal to 0 μs.

5.2. Results. We applied the three levels of our resource management on the application and obtained the following results.

5.2.1. Level 1 Results. Step 1 creates 6 types of RZ depicted in Table 3. RZ_1 is created by MDCT and VGA tasks, RZ_2 by AES, RZ_3 by DDS, RZ_4 by T48, RZ_5 by JPEG, and MULTF and RZ_6 by FIRs. If the RBs of one type of RZ are not constructed from the same task (i.e. there exist some RBs created by other tasks), the configuration overhead corresponding to this RZ must be recomputed as described in (7) before performing Step 2. In our application, the RBs of all RZs are created by one task. Hence, the RZ configuration overhead is provided by the predefined features of hardware tasks in Table 2.

During Step 2, the D costs between tasks and RZs are computed; in Table 3, the column specific for each task and its instances shows the values of obtained D costs. Thereafter, step 2 calculates the workloads of obtained RZs by assigning to each RZ the tasks giving lowest cost D as mentioned by the bold numbers. WorkLoad values are presented in the first column of Table 3. For example, the workload of RZ_2 is computed by assigning the hardware tasks AES, MULTF, and VGA. We detected an overload in RZ_2 and RZ_6. The overloads on these RZs are the result of the assigned hardware task execution time as well as the overheads, especially the configuration overheads of the RZs. These overloads are dealt with in step 3.

To resolve these overloads, step 3 adds two other RZs having the same type of RZ_2 since the other RZs cannot totally resolve its overload. In fact, the only possible migration is performed by T_8 on its second 1650 μs preemption point which loads off RZ_2 up to 299%. Whereas, the overload of RZ_6 could be resolved by performing the migration of two tasks among T_7, T_{13}, and T_{14} on RZ_3 on their second preemption points that is, 120 μs, since RZ_3 is the least loaded. Consequently, the final number of RZs is equal to 8: RZ_1, $3 \times RZ_2$ (RZ_2, RZ_7, RZ_8), RZ_3, RZ_4, RZ_5, and RZ_6.

5.2.2. Level 2 and Level 3 Results. Among all the available solvers [30], our work is based on the AIMMS environment [31] relying on powerful solvers. AIMMS environment has independently resolved the two subproblems: partitioning/fitting of RZs and fitting of tasks on RZs based on the Branch and Bound method. For the subproblem of partitioning/fitting of RZs, we used the Mixed Integer Programming model and for task fitting we employed Mixed Integer Nonlinear Programming model. At the end of resolution on CPU of 2 GHz with 2 GB of RAM, each RZ is fitted on its most suitable RPB and each preemption point of each task is mapped to a unique RZ. The resolution respects the consistency of constraints and extracts the optimal solution.

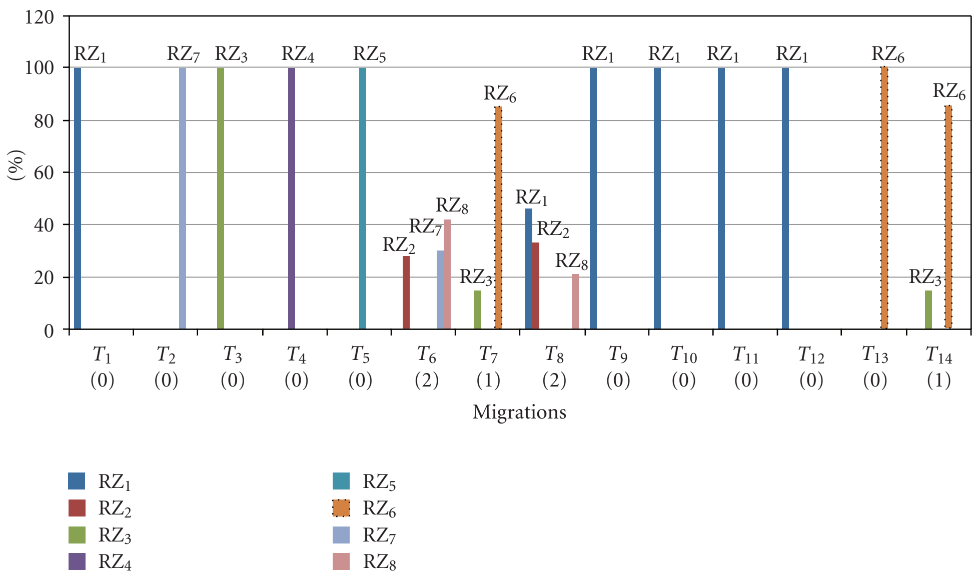


FIGURE 11: Task Occupation rates on RZs after task mapping.

The subproblem of RZ partitioning/fitting was resolved after 2 hours and 30 minutes. Table 4 shows the four coordinates of the most suitable RPBs for RZ fitting on the initial RR limited on the Virtex 5 FX200 device (82×12 RBs) as depicted in Figure 7. The initial RR is defined by the designer and by taking into account the set of resources dedicated only for the static part.

Table 5 shows comparison between the RBs of the obtained RPBs and their RZs. We observe high resource efficiency as the number of RBs within the RPBs is nearly equal to that of the RZs. Consequently, the internal fragmentation within the RPBs is considerably reduced. The differences of RBs are given by the last column ($\triangle$) of Table 5. For all RZs, except RZ_5 and RZ_6, their corresponding RPBs have one RB in excess. The RPB of RZ_5 contains 3 extra RB_3 and the RPB of RZ_6 strictly contains the required number of each RB type. The nonnull $\triangle$ is explained by the rectangular shape of the RPBs, thus the number of RBs included in the RPB could exceed the required RBs in the RZ. The nonnull $\triangle$ is also due to the heterogeneity on the device; the partitioning could book some RBs which are not used by the RZ.

Figure 7 depicts the floorplanning of RPBs conducted on the Virtex 5 FX200 device. The obtained results show an average of resources utilization of 36% of the available resources on the initial RR. This average is computed according to the number and the cost of each RB type. The optimization in utilization of resources minimizes the area of FPGA which is reconfigured at runtime. We have created static design by floorplanning each instance of each hardware task on its RPB without using the concept of dynamic partial reconfiguration. The obtained utilization of such static design resources is 63% of the available resources on the initial RR. Therefore, the gain of configuration overhead in a static design is paid by the resource waste, which is 43% compared to our obtained results employing dynamic partial reconfiguration. The RPBs are closely packed on the initial RR which avoids the resources waste and the external fragmentation in the device. For this reason, the initial RR could be resized in order to dedicate sufficient space for the remainder static part as depicted in Figure 7 by final RR.

Once the final RPB floorplanning is conducted, the final configuration overheads corresponding to RZs are determined. Thus, these new values are used to resolve the subproblem of task mapping to RZs.

The subproblem of task fitting on RZs was solved within 9 seconds. Figure 8 shows the results of preemption point mapping of hardware tasks in the application. T_7 and T_{14} start on RZ_6, they are preempted after 85% of their WCET and continue their execution on RZ_3. T_8 is mapped first to RZ_2, it is stopped on RZ_2 on its second preemption point that is, after 33% of execution and restarts on RZ_8 up to 54%. At this third preemption point, T_8 migrates to RZ_1 where it completes its execution. T_1, T_9, T_{10}, T_{11}, and T_{12} are totally mapped to RZ_1. T_2, T_3, T_4, T_5, and T_{13} are totally mapped, respectively to RZ_7, RZ_3, RZ_4, RZ_5, and RZ_6.

As shown in Figure 9, task T_6 starts its execution on RZ_8 then is preempted on its third preemption point that is, after 42% of execution. T_6 resumes its execution on RZ_7 till 72% of its execution. Hence, it is preempted again on RZ_7 on the fourth preemption point and restarts on RZ_2 where it is achieved.

This resolution of mapping hardware tasks to RZs discards the problem of task rejection as it guarantees an execution unit for each task to achieve its execution by respecting its predefined preemption points.

The bar chart on Figure 10 presents obtained RZ workloads after task mapping resolution. Except RZ_4 having a workload of 44%, the remainder RZs have balanced workloads which are closest to the average workload equal to 89%.

The bar chart in Figure 11 depicts the task occupation rates on the RZs as a result of mapping. This bar chart shows the number of migrations that must be performed to avoid the RZ overloads and ensure total execution of tasks. We obtained 6 migrations. T_8 realizes two migrations. In fact, T_8 is mapped to RZ_1, RZ_2, and RZ_8. The mapping of T_8 combines the two objectives expressed by *Map2,* which is the minimization of configuration overhead and *Map4* which optimizes utilization of costly resources. Similarly T_6 is mapped to RZ_2, RZ_7, and RZ_8. T_7 and T_{14} sustain both, migration from RZ_6 to RZ_3.

If we consider independent tasks, within each RZ respecting 100% as workload bound, execution sections mapped to this RZ could be scheduled by Earliest Deadline First (EDF) algorithm as they respect the monoprocessor sufficient schedulability condition ($\sum_{Ti\ mapped\ to\ RZ}(Exe_i/P_i) \leq 1$).

After RZ fitting on their RPBs and according to the obtained task mapping, the final RPB configuration overheads are determined. In the worst case, by counting all the possible preemption points, the total overhead including configuration overheads and functional overheads of tasks is 72959 μs. The overall overhead is 11% of total running time.

Table 6 gives some comparisons with previous work in hardware task placement in terms of task rejection, resource utilization, configuration overhead, and the complexity of the performed technique. To the best of our knowledge, there are several placement algorithms proposed for each goal. These algorithms could be classified as metaheuristics or online heuristics. Nevertheless, these algorithms target only one goal and do not take into account other goal satisfactions. In opposition to [11, 32], our multiobjective placement computes the configuration overhead in the worst case before scheduling (11%) and targets an application of 14 tasks which is not the case of [11, 32] that optimizes the configuration overhead only for two or three tasks during the scheduling (18%, 8%). Comparing to [10] (sets of 3,000 homogeneous tasks) and [1] (10.000 homogeneous tasks) applied in homogeneous devices; we have reduced efficiently the resource utilization (36%) for an application of 14 heterogeneous tasks and by taking into account the heterogeneity of recent reconfigurable devices. The heterogeneous resources in FPGA could fit the RZs on large RPBs giving a significant resource waste. Comparing to the offline simulated annealing in [4] performed for 100 tasks which produces 13% of task rejection, our three-level offline placement discards totally the task rejection for an application of 14 tasks.

6. Conclusion and Future Works

In this paper, we propose novel three-level resource management that investigates enhancing placement quality by reducing task rejection, resource waste, and configuration overheads. Our work is based on the optimization of resource utilization relying on the features of heterogeneous reconfigurable devices and on constrained optimization problems. We reported on resolution showing an improvement of placement quality compared to previous works. In fact, the overall overhead is 11% of total running time, the average resource utilization is 36% of the available resources on the reconfigurable region and we enhanced resource utilization by 43% compared to static design. In addition, the non-overload in some RZs improves the possibility of mapping other tasks by respecting their deadlines without performing another resource allocation. Unlike the works achieved in placement that deal with independent tasks, we eliminated the issue of task rejection as we pack tasks on RZs and employ dynamic partial reconfiguration.

Our results do not agree with other works on hardware task placement since experimental conditions are not the same. In fact, we used the recent FPGA technology that provides the highest configuration frequency and wide data ports that speed up configuration overhead. The improvement of resource utilization is explained by the optimality of our solution. The cancellation of task rejection is due to the employment of dynamic partial reconfiguration to map several tasks to the same RZ instead of dealing with each task placement independently. We have exploited powerful solvers that ensure the achievement of searching. Consequently, the most optimal solution provides the least rates of resources utilization and configuration overhead.

To conclude, it is recommended to take advantage of the obtained results for purposes of establishing the rules of hardware task scheduling for real-time applications. By attempting to follow the obtained partitioning/fitting, we guarantee the highest placement quality equal to or better than that obtained in offline three-level resource management.

Further work includes the defining of efficient on-line scheduling guided by obtained offline results. Moreover, we aim improving our offline task mapping by adding precedence constraints as well as deadline and periodicity constraints to achieve an offline mapping/scheduling of hardware tasks. Consequently, we ensure a full offline placement/scheduling of hardware tasks on hardware reconfigurable devices. The dependency between tasks should be investigated, especially in considering intertask communication with the overall overhead presented in this paper. Intertask communication will be an important criterion in deciding the most optimal RZ fitting. Intertask communication relies on the management of an efficient communication network such as FAT-Tree [33] as well as on the management of a shared memory.

Acknowledgments

This paper was supported by AIMMS technical support and Xilinx tools. It is sponsored by the national agency of research in France and the world-ranking "Secured Communicating Solutions" (SCS) cluster that pool together with industrial, research and higher education players which are involved in the microelectronics, telecommunications, software, and multimedia sectors.

References

[1] A. A. El Farag, H. M. El-Boghdadi, and S. I. Shaheen, "Improving utilization of reconfigurable resources using two dimensional compaction," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '07)*, pp. 135–140, Nice, France, April 2007.

[2] http://www.polytech.unice.fr/~fmuller/fosfor/,FOSFOR.2008.

[3] I. Belaid, F. Muller, and M. Benjemaa, "Off-line placement of hardware tasks on FPGA," in *Proceedings of the 19th International Conference on Field Programmable Logic and Application (FPL '09)*, pp. 591–595, Prague, Czech republic, September 2009.

[4] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test, Special Issue on Reconfigurable Computing*, vol. 17, no. 1, pp. 68–83, 2000.

[5] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson, *Approximation Algorithms for Bin Packing: A Survey*, Chapter 2, PWS Publishing Company, Boston, Mass, USA, 1997.

[6] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-hashing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)*, p. 178, Nice, France, April 2003.

[7] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A new approach for on-line placement on reconfigurable devices," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, vol. 4, p. 134, Santa Fe, NM, USA, April 2004.

[8] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "Intelligent merging online task placement algorithm for partial reconfigurable systems," in *Proceedings of the Design Automation Test Europe (DATE '08)*, pp. 1346–1351, Munich, Germany, March 2008.

[9] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement," in *Proceedings of the 41st Design Automation Conference (DAC '04)*, pp. 960–965, San Diego, Calif, USA, June 2004.

[10] H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt, "Task rearrangement on partially reconfigurable FPGAs with restricted buffer," in *Proceedings of the Field Programmable Logic and Applications*, vol. 1896, pp. 379–388, Vienna, Austria, August 2000.

[11] J. Resano, D. Mozos, D. Verkest, S. Vernalde, and F. Catthoor, "Run-time minimization of reconfiguration overhead in dynamically reconfigurable systems," in *Proceedings of the International Conference on Field Programmable Logic and Application*, vol. 2778 of *Lecture Notes in Computer Science*, pp. 585–594, Lisbon, Portugal, September 2003.

[12] E. M. Panainte, K. Bertels, and S. Vassiliadis, "FPGA-area allocation for partial run-time reconfiguration," in *Proceedings of the Design Automation Test Europe (DATE '05)*, pp. 100–105, Munich, Germany, March 2005.

[13] A. Lodi, S. Martello, and M. Monaci, "Two-dimensional packing problems: a survey," *European Journal of Operational Research*, vol. 141, no. 2, pp. 241–252, 2002.

[14] A. Lodi, S. Martello, and D. Vigo, "Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem," in *Proceedings of the Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization*, pp. 125–139, Sophia Antipolis, France, July 1997.

[15] A. Lodi, S. Martello, and D. Vigo, "Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems," *INFORMS Journal on Computing*, vol. 11, no. 4, pp. 345–357, 1999.

[16] B. S. Baker, E. G. Coffman Jr., and R. L. Rivest, "Orthogonal packings in two dimensions," *SIAM Journal on Computing*, pp. 846–855, 1980.

[17] S. Martello and D. Vigo, "Exact solution of the two-dimensional finite bin packing problem," *Management Science*, vol. 44, no. 3, pp. 388–399, 1998.

[18] K. Danne and S. Stuehmeier, "Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants," in *From Specification to Embedded Systems Application*, vol. 184 of *International Federation for Information Processing*, Springer, New York, NY, USA, 2005.

[19] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "3-D floorplanning: simulated annealing and greedy placement methods for reconfigurable computing systems," *Design Automation for Embedded Systems*, vol. 5, no. 3, pp. 329–338, 2000.

[20] S. P. Fekete, E. Kohler, and J. Teich, "Optimal FPGA module placement with temporal precedence constraints," in *Proceedings of the Conference Design Automation and Test in Europe*, pp. 658–665, Munich, Germany, 2001.

[21] J. Teich, S. P. Fekete, and J. Schepers, "Optimization of dynamic hardware reconfigurations," *The Journal of Supercomputing*, vol. 19, no. 1, pp. 57–75, 2001.

[22] F. Rivoallon and A. Cosoroaba, *Achieving Higher System Performance with the Virtex 5 Family of FPGAs*, Xilinx White Paper, San Jose, Calif, USA, 2006.

[23] I. Belaid, F. Muller, and M. Benjemaa, "Off-line placement of reconfigurable zones and off-line mapping of hardware tasks on FPGA," in *Proceedings of the Design and Architectures for Signal and Image Processing (DASIP '09)*, Sophia Antipolis, France, September 2009.

[24] J. Clausen, *Branch and Bound Algorithms-Principles and Examples*, University of Copenhagen, Copenhagen, Denmark, 1999.

[25] G. Pataki, M. Tural, and E. B. Wong, "Basis reduction and the complexity of branch-and-bound," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1254–1261, Austin, Tex, USA, January 2010.

[26] S. M. Azam, M. ur-Rehman, A. K. Bhatti, and N. Daudpota, "Parallel branch and bound model using logarithmic sampling (PBLS) for symmetric traveling salesman problem," in *Proceedings of the World Academy of Science, Engineering and Technology*, vol. 6, pp. 66–69, June 2005.

[27] J.-K. Hao, P. Galinier, and M. Habib, "Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes," *Revue d'Intelligence Artificielle*, vol. 13, no. 2, pp. 283–324, 1999.

[28] http://opencores.org/.

[29] "Virtex-5 FPGA Configuration User Guide," Xilinx white paper, August 2009.

[30] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó, "A comparison of complete global optimization solvers," *Mathematical Programming*, vol. 103, no. 2, pp. 335–356, 2005.

[31] http://www.aimms.com/.

[32] J. A. Clemente, C. González, J. Resano, and D. Mozos, "A hardware task-graph scheduler for reconfigurable multitasking systems," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, pp. 79–84, Cancun, Mexico, December 2008.

[33] L. Devaux, D. Chillet, S. Pillement, and D. demigny, "Flexible communication support for dynamically reconfigurable fpgas," in *Proceedings of the 5th Southern Conference on Programmable Logic (SPL '09)*, pp. 65–70, São Paulo, Brazil, April 2009.