electronics

MDPI

Article

Sw/Hw Partitioning and Scheduling on Region-Based Dynamic Partial Reconfigurable System-on-Chip

Qi Tang [1,*], Biao Guo [1,2] and Zhe Wang [1,2]

[1] College of Electronic Science and Technology, National University of Defense Technology, Changsha 410073, China; guobiao@hnu.edu.cn (B.G.); wangzhe_0024@hnu.edu.cn (Z.W.)
[2] College of Electrical and Information Engineering, Hunan University, Changsha 410082, China
* Correspondence: qitangnudt@nudt.edu.cn

Received: 29 July 2020; Accepted: 19 August 2020; Published: 21 August 2020

Abstract: A heterogeneous system-on-chip (SoC) integrates multiple types of processors on the same chip. It has great advantages in many aspects, such as processing capacity, size, weight, cost, power, and energy consumption, which result in it being widely adopted in many fields. The SoC based on region-based dynamic partial reconfigurable (DPR) FPGA plays an important role in the SoC field. However, delivering its powerful capacity to the consumer depends on the efficient Sw/Hw partitioning and scheduling technology that determines the resource volume of the DPR region, the mapping of the application to the DPR region and other processors, and the schedule of the task and its reconfiguration. This paper first proposes an exact approach based on the mixed integer linear programming (MILP) for the Sw/Hw partitioning and scheduling problem. The proposed MILP is able to solve the problem optimally; however, its scalability is poor, despite that we carefully designed its formulation and tried to make it as concise as possible. Therefore, a multi-step hybrid method that combines graph partitioning and MILP is proposed, which is able to reduce the time complexity significantly with the solution quality being degraded marginally. A set of experiments is carried out using a set of real-life applications, and the result demonstrates the effectiveness of the proposed methods.

Keywords: dynamic partial reconfiguration; Sw/Hw partitioning and scheduling; system-on-chip; mixed-integer linear programming; FPGA

1. Introduction

As the user requirements keep increasing and technology advances, reconfigurable systems become ubiquitous currently, including traditional computers, smart phones, multimedia systems, and software-defined radio systems. The reconfigurable systems utilize various kinds of flexible structures and can be applied to high-performance computing, e.g., multimedia, communication, and radar signal processing. The primary feature of the reconfigurable systems, despite the diversity of their structures, is the programmability. The field programmable gate array (FPGA) is an important programmable device, the function of which can be changed by dynamically loading new bitstreams that implement various kinds of hardware accelerators. Such a reconfiguration, however, needs to re-write all logics of the FPGA. Since the bitstream file for the whole FPGA is quite large, a remarkable reconfiguration overhead results. However, the application may be small in size, requiring only a few reconfigurable resources; thus, such a full reconfiguration increases the reconfiguration latency and also incurs resource waste. Therefore, a dynamic partial reconfiguration is introduced to most modern FPGAs. Dynamic partial reconfigurable (DPR) FPGAs enable run-time reconfiguration of a specific portion of the FPGA circuit without affecting the execution of other parts of the FPGA. Compared with full dynamic reconfiguration, dynamic partial reconfiguration can speed up the load of a hardware accelerator, thus augmenting the response speed of the platform. For example, it takes more than 8 ms to

reconfigure the whole Xilinx Zynq-7010 platform, which provides a reconfiguration speed of 400 MB/s; if the application only occupies 25% of the total resources, the reconfiguration time can be reduced to 2 ms using dynamic partial reconfiguration [1]. Besides, dynamic partial reconfiguration makes it easier and more flexible to share FPGA resources among multiple applications either in time or space. Applications can share the same portion of logics in time or share the FPGA in space by using different portions of logics dynamically. Therefore, dynamic partial reconfiguration enables runtime virtualization of the FPGA, which is proven to be effective and efficient in many real-time real-life applications [2,3]. Further, the FPGA resources may not be enough to accommodate a whole application, i.e., the number of resources provided by the FPGA is less than the required volume of resources of the application. For this case, dynamic partial reconfiguration makes it possible to share the FPGA resource in the time domain and space domain among different application tasks; and by swapping in and out tasks of the application on FPGA reconfigurable logics, the application can be executed on the resource-constrained platform, which helps to reduce the cost of the system and enables the reuse of old resource-constrained platforms to run resource-intensive applications.

The reconfiguration dimension is an important feature of the DPR FPGA, and it can be classified as 1D and 2D reconfigurations [4,5]. The 1D reconfiguration has to modify all configurable logic blocks (CLB) in a column if any CLB in the column is needed, i.e., a column must be reconfigured entirely. Example 1D reconfigurable devices include Xilinx Virtex and Virtex-2 (Pro). The 2D reconfiguration allows reconfiguring a portion of a column. Example 2D reconfigurable devices include Xilinx Virtex-4/5/6, 7-series FPGAs (Artix, Kintex, and Virtex-7) and Zynq-7000, Ultrascale(+) SoCs.

The granularity of the reconfiguration in the 2D reconfigurable devices also differs. For example, Virtex-4 allows the reconfiguration region to be a height that is a multiple of 16 CLBs, while Virtex-5 and later series FPGAs support the minimal reconfiguration granularity of a tile (reconfigurable frame), which is the intersection of a row (clock region) and a block/column. The tile size also differs: for Xilinx Virtex-5, Virtex-6, and 7-series FPGAs, the CLB tile size is 20, 40, and 50 CLBs, respectively.

This paper focuses on the region-based 2D DPR FPGA [6–10]. Specifically, we focus on the SoC that integrates the embedded microprocessor and DPR FPGA, e.g., Xilinx Zynq, Ultrascale(+), and Altera Arria-10 SoCs. For simplicity, we name such SoCs as DPR SoCs in this papr. A DPR SoC is a kind of heterogeneous SoC that integrates multiple types of processors on the same chip and has great advantages in many aspects such as processing capacity, size, weight, cost, power, and energy consumption. The software/hardware (Sw/Hw) partitioning and scheduling technology is a key technology for ensuring the real-time performance on the above mentioned DPR SoC. The modules/tasks of the application can be potentially deployed either to the soft core (microprocessor) or the FPGA reconfigurable region with multiple implementations using for example the high level synthesis (HLS) tool [11–13]. Therefore, several questions have to be solved for this problem to obtain the optimal solution with the shortest schedule length: how large is each region, where to deploy the application task, when to schedule each task, either on the microprocessor or the FPGA region, when to start the reconfiguration of the task on the FPGA while avoiding contention for the reconfiguration port (e.g., internal configuration access port (ICAP) and processor configuration access port (PCAP) on Zynq), and how to take into account configuration pre-fetching to hide the reconfiguration delay.

This Sw/Hw partitioning and scheduling problem is a combinatorial optimization problem. Mixed integer linear programming (MILP) is an approach for solving combinatorial optimization problems optimally. However, building a proper and concise model for the problem is the key part for problem solving, which has great impacts on the time complexity. Despite that the MILP-based approach paves the way to solve the problem and produce the optimal result, its scalability is weak. Even for the medium-scale problem, the MILP-based method may need tens of hours to search the solution space to yield the optimal solution, making it hard to use in practice. The flaw of the MILP-based method motivated us to develop a multi-step hybrid method with less time complexity. The hybrid method combines the graph partitioning approach and the MILP. While the graph partitioning method divides the problem into a set of sub-problems, the MILP solves each sub-problem

with the partial result of the former sub-problem as the input. Further, a priority-based graph partitioning method is proposed to effectively divide the problem into a set of small sub-problems. Therefore, the contributions of this paper are:

- The formal definition of the partitioning and scheduling problem on DPR SoC integrating the CPU and 2D DPR FPGA.
- An accurate MILP formulation for the partitioning and scheduling problem.
- A multi-step hybrid method that integrates the priority-based graph partitioning method.
- Extensive experiments with a set of real-life applications.

The remainder of the paper is organized as follows. In Section 2, we discuss the related work. Section 3 introduces the platform model, the application model, and the studied problem. The proposed MILP model is introduced in Section 4, and the multi-step hybrid approach is illustrated in Section 5. Section 6 presents and analyzes the experimental results. We conclude the paper in Section 7.

2. Related Work

The authors of [14] studied the effectiveness of using DPR at reducing the size, cost, power, and energy consumption by experimentally evaluating a set of real-life signal processing hardware accelerators on the DPR FPGA. The result also demonstrated the usefulness of DPR in improving real-time performance and reducing resource consumption.

The work in [15] utilized the DPR FPGA to tolerate the faults occurring at runtime and improve the system reliability. The proposed method was demonstrated to be advantageous at improving the resource utilization and task acceptance ratio and reducing the fragmentation ratio.

The usage of the DPR FPGA in cloud computing to speed up provided services is an emerging research area, which was studied in [16]. The work in [16] proposed a benefit-based scheduling metric to guide task assignment on the DPR FPGA, and the result demonstrated that the method was advantageous at reducing resource consumption.

In [17], the authors proposed two optimal module/task placement and scheduling methods for applications with temporal precedence constraints on 2D DPR FPGAs. The proposed methods can be used to optimize either the schedule length or the FPGA size. In this work, each task was modeled as a three-dimensional box in space and time; hence, the placing and scheduling problem was transformed into a packing problem in a container with a specific size. However, the proposed approach models the reconfiguration time in the task execution time, without taking into account the single reconfigure port constraint and configuration prefetch. Therefore, the produced result is inaccurate, since multiple reconfigurations can be carried out in parallel. Besides, ignoring the reconfiguration pre-fetching also worsens the schedule performance. The work in [18] presented a time-step ILP-based exact approach for the Sw/Hw partitioning problem (placing and scheduling) on an SoC with a microprocessor and a 1D DPR FPGA. The proposed model takes into account practical physical constraints, i.e., an Hw should occupy continuous FPGA columns and one reconfiguration port constraint; and configuration prefetch was utilized to reduce reconfiguration overhead so as to further improve the performance. Besides, a heuristic algorithm based on Kernighan-Lin Fiduccia–Mattheyses (KLFM) was also proposed. However, module reuse was not shown in this work.

The paper [4] proposed a time-step ILP formulation for placing the scheduling task graphs on the 1D DPR FPGA, with configuration prefetch, a single reconfiguration port, and module reuse being taken into account. Besides, the 1D ILP formulation was also extended to the SoC with multiple 2D DPR FPGAs. Since there are multiple FPGAs on the SoC, the same number of concurrent reconfigurations are enabled in the 2D ILP; however, it ignores the fact that on each FPGA, only one reconfiguration is supported at the same time. The authors also proposed a heuristic called Napoleon for the problem. The heuristic considers reconfiguration pre-fetching, module reuse, and anti-fragmentation techniques. The result showed the effectiveness of module reuse compared to the work in [18].

The authors of [19] proposed an ant colony optimization (ACO) approach for mapping, scheduling, and placing directed acyclic graphs (DAG) on the SoC with a 1D DPR FPGA and one or several instruction processors (i.e., microprocessor and/or digital signal processor (DSP)). Reconfiguration pre-fetching is utilized in the proposed method to hide reconfiguration overhead, and reconfiguration port contention is taken into account. The result demonstrated that ACO is more effective in schedule length compared with KLFM [18].

The work in [20] introduced a duplication-based time-step ILP model with reconfiguration pre-fetching and a single reconfiguration constraint for placing and scheduling precedence constrained task chains on the 1D DPR FPGA. The proposed method makes use of the data parallelism provided by the application task and replicates the same task several times on the DPR FPGA to improve the real-time performance. The proposed model assumes that the up-bound duplication number of each task is known a priori; and each copy of a child task is assumed to start after all instances of the ancestor task finish execution. Besides, the execution time of each duplication of the same task can differ, which is resolved by the ILP model. Further, the authors proposed a semi-online heuristic algorithm PARLGRANfor the same problem.

Region-based Sw/Hw partitioning on the SoC with the 2D DPR FPGA and microprocessor was studied in [5,10,21], with reconfiguration pre-fetching and a single reconfiguration port constraint being taken into account. These methods assume that the up-bound number of reconfigurable regions is fixed. The work in [5] proposed a mixed-integer linear programming (MILP) formulation for mapping, scheduling DAG, and sizing reconfigurable regions on the heterogeneous SoC with a microprocessor and a 2D region-based DPR FPGA to optimize power, energy, and schedule length. The work in [21] proposed two heuristics for the same problem to reduce the algorithm complexity. Floorplanning [22] is required to ensure the feasibility of the final solution produced by the algorithms in [5,21], and if the solution is infeasible, the proposed algorithms have to be re-executed by virtually reducing the number of FPGA resources [21]. Note that in [5,21], the reconfiguration of the first task on each reconfiguration region was not taken into account.

The authors of [10] proposed an MILP model for the Sw/Hw partitioning problem on the heterogeneous SoC with a microprocessor and a 2D region-based DPR FPGA, assuming tasks of the application to be clustered with the available heuristic algorithm, and the task of a cluster was limited to being mapped to either the microprocessor or a specific reconfigurable region of the FPGA. The independent set-based algorithm (ISBA) [23] is used to map the DAG to clusters by merging tasks that are not active in one or more periods to a cluster. The work of [10] is the most related work to the studied problem in this paper; hence, it was also selected for performance evaluation. Again, as in [5,21], the reconfiguration of the first task on the region was ignored.

The authors of [24] developed a simulated annealing algorithm for partitioning, scheduling, and floorplanning of the application on the DPR FPGA. The novelty of this work was that it solved all critical problems with respect to deploying an application on the DPR FPGA. However, the proposed method ignores the possibility of deploying the task of the application on the CPU. The work in [25] studied how to evaluate the reconfiguration cost in terms of time for the DPR FPGA. The authors coined the term "DPR profitability", targeting real-time systems, and an innovative approach for calculating the DPR time and the worst-case bound was developed. The work in [26] studied the automotive placement on the DPR FPGA to reduce resource utilization and the total wirelength. The selection of the Partial Reconfigurable Region (PRR) was based on the shapes, location, and communication with others.

The works in [8,9] proposed a hybrid mapping-scheduling algorithm (HMS) to capture the pipeline behaviors of the application and reduce the reconfiguration overhead on the region-based DPR FPGA with a fixed number of reconfigurable regions. The application is first translated to sequential snapshots that are composed of a set of Hw cores; each snapshot is then mapped separately by clustering the snapshot into islands [6,7]; each island is then assigned to a reconfigurable region. Since islands in subsequent snapshots may be compatible, i.e., they share the same Hw core, the reconfiguration overhead can be potentially reduced, hence improving the schedule performance.

The papers [6,7] studied reducing reconfiguration overhead when scheduling multiple applications sequentially on the region-based DPR FPGA with a fixed number of reconfigurable regions of the same size. The proposed algorithm captures the fact that many applications share some hardware cores to perform common operations. This kind of similarity in applications can be used to reduce reconfigurations when switching applications on the FPGA using dynamic partial reconfiguration.

One novelty of the works in [6–9] was that different Hw cores were allowed to be combined into a single island/cluster that was then assigned to a reconfiguration region to take into account that the region size may be large enough to hold multiple Hw cores. Such a combination contributes to less reconfiguration overhead, since the reconfiguration overhead of an Hw core is directly related to the size of the reconfigurable region. If some Hw cores are combined into one Hw, then less reconfiguration is required.

3. Problem Description

We consider Sw/Hw partitioning and scheduling of an application on the hardware platform with the system architecture shown in Figure 1. The application is specified as a directed acyclic graph, which is also called the task dependency graph since the edge in the graph represents inter-task data dependency. The application model can be extracted from the high-level language of the application code, e.g., C, VHDL, etc. As the works in [10,18,23], the feasibility of the Sw/Hw partitioning and scheduling result still needs to be verified using the floorplanning tool [22,27,28]. If the result is impractical, the algorithm can be iterated by reducing the total resource number of the platform until a feasible result is found. Further, integrating the MILP-based floorplanning technique [22] into the proposed MILP approach is also possible.

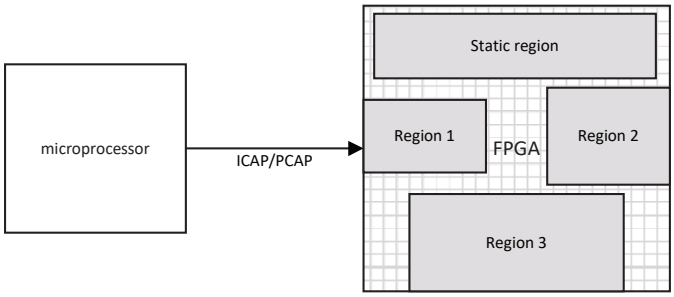

Figure 1. An example dynamic partial reconfigurable (DPR) SoC platform. ICAP, internal configuration access port; PCAP, processor configuration access port.

3.1. Platform Model

The target platform architecture is as shown in Figure 1. The platform is composed of a microprocessor/CPU and an FPGA with DPR ability. The FPGA can be divided into a static region and a set of dynamic reconfigurable regions that are denoted as $\mathbf{PR} = \{PR_1, PR_2, ..., PR_{|\mathbf{PR}|}\}$. Each reconfigurable region composes a set of different reconfigurable resources, including CLB, DSP, BRAM, etc. We use $\mathbf{H} = \{h_1, h_2, ..., h_{|\mathbf{H}|}\}$ to denote the set of different resource types of the FPGA. For each kind of resource $h \in \mathbf{H}$, N_h^k represents the number of resources h in the reconfigurable region $PR_k \in \mathbf{PR}$. For each kind of resource, TN_h denotes the total number of resources h. Each reconfigurable region PR_i is associated with an integer RT_i, representing the time needed to reconfigure this region. $RT_i = S_{bit}^i / B_{cfg}$, where S_{bit}^i is the bitstream size of region PR_i and B_{cfg} is the bandwidth of the reconfiguration port. $S_{bit}^i = \sum_{h \in \mathbf{H}} S_h * N_h^i$, where S_h is the size of the resource h measured in bits. In the platform, the local communication either in the microprocessor or in the FPGA is treated as delay-free by using local memory access [10]. An example DPR SoC platform abstracted from the Xilinx Zynq-series SoC is shown in Figure 1. The SoC is composed of a microprocessor and an FPGA, and the FPGA has three dynamic partial reconfigurable regions and a static region. The FPGA can be partially reconfigured by the microprocessor using the ICAP/PCAP port.

3.2. Application Model

As many other works [2,10,12], the application is modeled as a directed acyclic graph (DAG) $\mathbf{G}(\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is a finite set of nodes representing the tasks of an application and $\mathbf{E}$ is a finite set of directed edges denoting data precedences among tasks. Each task $v \in \mathbf{V}$ is associated with two computation costs cs_v and ch_v of integers, representing the computation time it takes to complete the execution of the task on the microprocessor and the FPGA, respectively. Each task is also associated with a set of resource consumptions corresponding to each kind of resource, e.g., CLB, DSP, BRAM, etc. $\forall v \in \mathbf{V}$ and $\forall h \in \mathbf{H}$, N_h^v represents the required number of resource h to execute task v on the FPGA. The task can be mapped to a specific reconfigurable region only if the type and number of resources provided by the region are those requiredby the task match. Each edge $e \in \mathbf{E}$ is defined as a tuple (src, dst), with src and dst being the source and destination/child task of the edge, respectively. Each edge $e \in \mathbf{E}$ is associated with a communication cost w_e of the integer, denoting the communication time required to move the dependence data from the source task to the destination task when they are deployed onto different kinds of processing elements. The task without any child is called the exit task, and we use $\mathbf{V}_{exit}$ to represent the set of exit tasks of the DAG.

Figure 2 shows an example application [18] modeled with DAG. The application has eight tasks and nine edges. Table 1 records the task parameters of the DAG in Figure 2. Task parameters include the execution time on the CPU (Sw time; the unit is cycles), the execution time on the FPGA (Hw time; the unit is cycles), and the required CLB number of each task (CLB num).

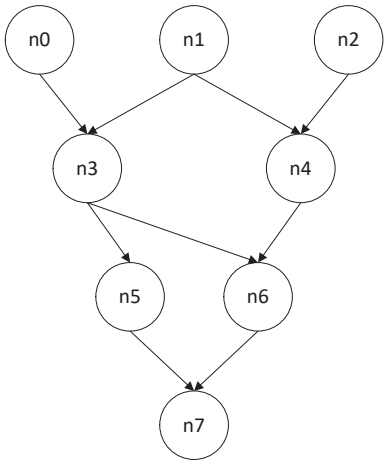

Figure 2. An example DAG.

Table 1. Task parameters of the example DAG. num, number.

Task Name	Sw Time (Cycle)	Hw Time (Cycle)	CLB Num
n_0	23	5	3
n_1	9	2	3
n_2	11	2	1
n_3	14	3	1
n_4	10	2	2
n_5	7	3	4
n_6	6	2	2
n_7	4	2	1

3.3. The Sw/Hw Partitioning and Scheduling Problem

With the platform and application model, the Sw/Hw partitioning and scheduling problem can be described to map and schedule tasks of the application to different computing element (the CPU and FPGA static and dynamic regions) while complying with the different physical constraints of the DPR-enabled heterogeneous SoC. These constraints include those that are common to static

non-preemptive scheduling on multiprocessors, e.g., tasks on the processor should execute in series, and a task cannot be interrupted by other tasks when it is executing [15,29,30]. These constraints also include those that are specific to the DPR FPGA, e.g., the single reconfigure port constraint [12,31]; a task on the FPGA must be reconfigured before executing, reconfiguration, and execution on the same region of the FPGA and must happen in series; the number of each kind of resource of a region must be enough to accommodate each task assigned to it; and the aggregate resource volume of all regions should not exceed that provided by the FPGA. What is more, the features provided by the hardware have to be employed for the sake of better performance, e.g., reconfiguration pre-fetching. Reconfiguration pre-fetching enables hiding part of the reconfiguration delay by utilizing the physical character of the FPGA, that the reconfiguration of one region and the execution on other regions can be carried out in parallel [31].

To summarize, for the target Sw/Hw partitioning and scheduling problem, the following solutions are to be resolved with the physical constraints being satisfied and the features of the platform being employed:

- The resource type and number on each reconfiguration region.
- The mapping of each task (the task may be mapped to the CPU or a reconfiguration region).
- The start execution time of each task.
- The start reconfiguration time of the task that is mapped to the FPGA.
- The execution order and reconfiguration order of all tasks.

For the application in Figure 2, with the task parameters described in Table 1, assuming that the communication cost of each edge in Figure 2 is one and the CLB number of each task is equal to the reconfiguration delay, Figure 3 shows the Sw/Hw partitioning and scheduling of the application on the platform with eight CLBs. In Figure 3, the rectangles with labels n_i and r_i represent the execution and reconfiguration of task n_i, respectively.

According to the Sw/Hw partitioning and scheduling result in Figure 3, the following observations can be obtained:

- The resource number of the FPGA is eight, while the aggregate resource number of the application by adding the resource requirement of each task of the DAG is 17, showing that the DPR ability of the hardware enables the execution of the application on a resource-constrained platform. By partitioning the FPGA into three partial dynamic regions, the resource of the FPGA is shared by the application in both time and space.
- Reconfigurations of different tasks are carried out in series. As shown in the figure, the reconfiguration port is busy during the time interval $[0, 19]$, and no pairs of reconfigurations overlap in time.
- Reconfiguration and execution of two tasks can happen in parallel (reconfiguration pre-fetching), e.g., n_0 on PR_1 and r_2 on PR_2 overlap during $[5, 6]$. When task n_0 is executing on PR_1, the reconfiguration of task n_2 is carried out on PR_2, thus hiding its reconfiguration delay.
- In each region, reconfiguration and execution happen in turn, e.g., in region PR_2, the reconfiguration and execution sequence is $r_2, n_2, r_3, n_3, r_7, n_7$.
- The Sw/Hw partitioning and scheduling technique can speed up the execution of the application remarkably. While the schedule length of the application on the CPU takes 84 time units, the Sw/Hw partitioning and scheduling technique produces a schedule with the schedule length being only 20.

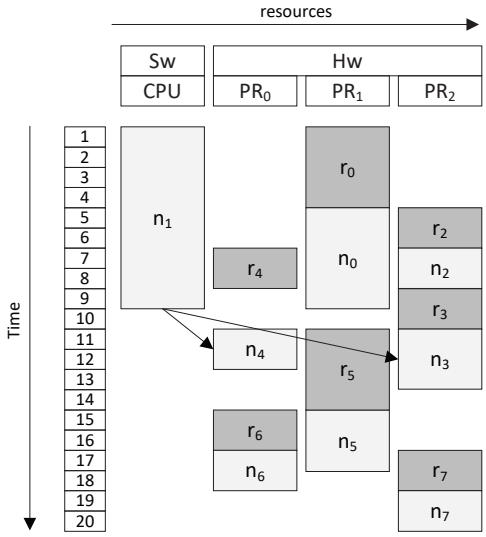

Figure 3. The Sw/Hw partitioning and scheduling result of the example DAG.

4. The Proposed MILP Formulation

This section introduces the exact approach for the Sw/Hw partitioning and scheduling problem based on the MILP.

Reconfiguration is an important part of the Sw/Hw partitioning and scheduling problem. Due to the physical features of the FPGA, each task assigned to an FPGA region should be reconfigured before it is executed. The reconfiguration renders extra time overhead, and the length of the reconfiguration time is directly related to the resource volume of the FPGA region to be reconfigured and the bandwidth of the reconfiguration port. To take into account the reconfiguration time in the schedule, we make use of the concept of the reconfiguration node [10]. Each task of the application is associated with a reconfiguration node. $\forall v \in \mathbf{V}$, R_v is the reconfiguration node of task v. Note that the reconfiguration node R_v may not be carried out in the schedule ultimately, and whether or not it should be executed is figured out by the algorithm. When the task is assigned to the microprocessor, reconfiguration is unnecessary; otherwise, it is indispensable. The time taken by the reconfiguration node R_v is determined by the resource volume of the reconfiguration region where task v is assigned, and the resource volume of the region is determined by the set of tasks that are finally mapped onto it.

4.1. Variables

To model the Sw/Hw partitioning and scheduling problem with MILP, the following variables are used:

$\forall v \in \mathbf{V}$, we use the integer variables s_v and r_v to represent the task start execution time and start reconfiguration time (if the task is mapped to the reconfigurable region).

$\forall v \in \mathbf{V}, PR_i \in \mathbf{PR}$, we introduce the binary variable m_{vi} to represent if the task v is finally mapped to the reconfigurable region PR_i. If $m_{vi} = 1$, this means that the task v is mapped to PR_i; otherwise, the task is not mapped to PR_i, i.e., the task is either mapped to the microprocessor or to other reconfigurable regions. If $\forall PR_i \in \mathbf{PR}$, $m_{vi} = 0$, then task v is mapped to the microprocessor.

$\forall e \in \mathbf{E}$, we use the binary variable d_e to represent if the source and destination tasks of edge e are mapped to different computation elements (one to the microprocessor and the other to the FPGA). It equals one if the source task and the destination task are mapped to different processing elements; otherwise, they are mapped to the same processing element (both mapped to the microprocessor or the FPGA).

$\forall a, b \in \mathbf{V}, a \neq b$, we use the auxiliary binary variable z_{ab} to represent the execution order of these two tasks on the microprocessor if they are both mapped to the microprocessor (otherwise, it is a free

variable). It equals zero if task a executes before b on the microprocessor; otherwise, b is executed before a.

$\forall a, b \in \mathbf{V}, a \neq b$, we use the auxiliary binary variable x_{ab} to represent the reconfiguration order of tasks a and b on the FPGA if they are both mapped to the FPGA (otherwise, it is a free variable), so as to model the single reconfiguration controller constraint. It equals zero if task a is reconfigured before b on the FPGA; otherwise, task b is reconfigured before a.

For each $PR_k \in \mathbf{PR}$ and $h \in \mathbf{H}$, we use the integer variable N_h^k to represent the resource number of h on the reconfiguration region PR_k.

The integer variable sl is used to represent the schedule length.

4.2. Constraints

Each task of the application can be mapped to only one processing element (the microprocessor or the FPGA); and if the task is assigned to the FPGA, it can be mapped to only one reconfigurable region. Therefore, the following constraint should hold for all $v \in \mathbf{V}$.

$$\sum_{PR_i \in \mathbf{PR}} m_{vi} \leq 1 \tag{1}$$

Note that if $\sum_{PR_i \in \mathbf{PR}} m_{vi} = 1$, then the task is mapped to the FPGA; otherwise, it is mapped to the microprocessor. Hence, $\sum_{PR_i \in \mathbf{PR}} m_{vi}$ gives the information about to where the task is mapped. For simplicity, we use the notation h_v to denote $\sum_{PR_i \in \mathbf{PR}} m_{vi}$ in the following.

The following constraints should hold for all $a, b \in \mathbf{V}, a \neq b$ to guarantee sequential task execution on the microprocessor.

$$s_a + (1 - h_a) * cs_a \leq s_b + (h_a + h_b) * M_1 + z_{ab} * M_1 \tag{2}$$

$$s_b + (1 - h_b) * cs_b \leq s_a + (h_a + h_b) * M_1 + (1 - z_{ab}) * M_1 \tag{3}$$

where M_1 is a large positive constant integer that makes $s_a + (1 - h_a) * cs_a \leq s_b + M_1$ and $s_b + (1 - h_b) * cs_b \leq s_a + M_1$ always hold. We first consider the case when $h_a = 0$ and $h_b = 0$, i.e., both tasks a and b are mapped to the microprocessor. If $z_{ab} = 0$, the constraint (2) is reduced to $s_a + (1 - h_a) * cs_a \leq s_b$, which guarantees that task b starts execution after task a finishes; while the constraint (3) is reduced to $s_b + (1 - h_b) * cs_b \leq s_a * M_1$, which always holds owing to the existence of M_1. Otherwise, if $z_{ab} = 1$, the constraint (2) is reduced to $s_a + (1 - h_a) * cs_a \leq s_b + M_1$, which always holds; while the constraint (3) is reduced to $s_b + (1 - h_b) * cs_b \leq s_a$, which guarantees that task a starts execution after task b finishes. However, if $h_a = 1$ or $h_b = 1$, both the constraints (2) and (3) always hold owing to the definition of M_1; in this case, variables s_a, s_b, and z_{ab} are free. The use of the large positive constant integer M_1 is a trick to make the constraint linear [29]. The value of M_1 is application related. Generally, the single-processor schedule length on the CPU, i.e., the sum of the execution time of each task of the application on the CPU, satisfies the requirement.

The following constraints guarantee the data precedence for each edge $e(a, b) \in \mathbf{E}$.

$$d_e \geq h_a - h_b \tag{4}$$

$$d_e \geq h_b - h_a \tag{5}$$

$$s_a + (1 - h_a) * cs_a + h_a * ch_a + w_e * d_e \leq s_b \tag{6}$$

If $h_a \neq h_b$, the constraints (4) and (5) make $d_e = 1$, i.e., tasks a and b are mapped to different processing elements, and the data precedence between them renders communication delay. Note that the constraints (4)–(6) do not guarantee d_e to be zero for the case when $h_a = h_b$. However, the MILP solver is able to do this in order to minimize the schedule length. Based on the value of d_e, communication delay is

taken into account by the constraint (6), which makes sure task b starts execution after the dependent data from task a are ready.

The following constraint should hold for all $a \in \mathbf{V}$ to make sure that a task executes after its reconfiguration if it is mapped to the FPGA.

$$r_a + \sum_{PR_i \in \mathbf{PR}} (m_{ai} * RT_i) \leq s_a + (1 - h_a) * M_2 \tag{7}$$

where M_2 is a large positive constant integer that makes $r_a + \sum_{PR_i \in \mathbf{PR}} (m_{ai} * RT_i) \leq s_a + M_2$ always hold. If $h_a = 1$, i.e., task a is mapped to the FPGA, then the constraint (7) is reduced to $r_a + \sum_{PR_i \in \mathbf{PR}} (m_{ai} * RT_i) \leq s_a$, making task a execute after its reconfiguration finishes. Otherwise, if $h_a = 0$, i.e., task a is mapped to the microprocessor, the constraint is reduced to $r_a + \sum_{PR_i \in \mathbf{PR}} (m_{ai} * RT_i) \leq s_a + M_2$, which always holds due to the definition of M_2.

Since both m_{ai} and RT_i are variables, the constraint (7) is not linear. To make it linear, we introduce the constraint (8) to replace (7).

$$r_a + RT_i \leq s_a + (1 - m_{ai}) * M_2, \forall PR_i \in \mathbf{PR} \tag{8}$$

When $m_{ai} = 1$, task a is mapped to region PR_i for execution, and the constraint is reduced to $r_a + RT_i \leq s_a$, making the reconfiguration of task a finish before its execution. If $m_{ai} = 0$, the constraint is reduced to $r_a + RT_i \leq s_a + M_2$, which always holds since M_2 is a large positive constant integer.

The following constraints are introduced for all $a, b \in \mathbf{V}, a \neq b$ to guarantee serial reconfiguration on the FPGA. When both h_a and h_b equal one, i.e., tasks a and b are both assigned to the FPGA, the constraints (9) and (10) assure that the reconfiguration times of tasks a and b do not overlap.

$$r_a + \sum_{PR_i \in \mathbf{PR}} (m_{ai} * RT_i) \leq r_b + (2 - h_a - h_b) * M_3 + x_{ab} * M_3 \tag{9}$$

$$r_b + \sum_{PR_i \in \mathbf{PR}} (m_{bi} * RT_i) \leq r_a + (2 - h_a - h_b) * M_3 + (1 - x_{ab}) * M_3 \tag{10}$$

On each reconfigurable region, the execution and reconfiguration of different tasks should not overlap in the time domain, which is captured by the following constraints that should hold for all $a, b \in \mathbf{V}, a \neq b$ and $PR_i \in \mathbf{PR}$. When both m_{ai} and m_{bi} equal one, i.e., tasks a and b are both assigned to the FPGA reconfigurable region PR_i, the constraints (11) and (12) guarantee that the execution of task a is carried out before the reconfiguration of task b (when $x_{ab} = 0$) or the execution of task b is carried out before the reconfiguration of task a (when $x_{ab} = 1$).

$$s_a + m_{ai} * ch_a \leq r_b + (2 - m_{ai} - m_{bi}) * M_4 + x_{ab} * M_4 \tag{11}$$

$$s_b + m_{bi} * ch_b \leq r_a + (2 - m_{ai} - m_{bi}) * M_4 + (1 - x_{ab}) * M_4 \tag{12}$$

To model the resource constraint, we use the integer variable N_h^k for each $PR_k \in \mathbf{PR}$, denoting the number of resources h of the reconfiguration region PR_k. The following equations are introduced to model the resource constraint.

$$m_{vk} * N_h^v \leq N_h^k, \forall PR_k \in \mathbf{PR}, v \in \mathbf{V}, h \in \mathbf{H} \tag{13}$$

$$\sum_{PR_k \in \mathbf{PR}} N_h^k \leq TN_h, \forall h \in \mathbf{H} \tag{14}$$

where TN_h is the total volume of resource h on the FPGA.

Constraint (13) makes sure the resource number of each reconfiguration region is large enough to hold any task assigned to it.

Constraint (14) guarantees that the aggregate resource number of each kind of resource of all reconfigurable regions does not exceed that provided by the platform.

The following constraint is introduced to bound the schedule length by the finish time of the exit task $a \in \mathbf{V}_{exit}$.

$$s_a + (1 - h_a) * cs_a + \sum_{PR_i \in \mathbf{PR}} (m_{ai} * ch_a) \leq sl \tag{15}$$

4.3. Model Complexity

For the proposed MILP model, there are $|\mathbf{V}|$ variables each for s_v and r_v, $|\mathbf{E}|$ variables for d_e, $|\mathbf{V}| * (|\mathbf{V}| - 1)/2$ variables each for z_{ab} and x_{ab}, $|\mathbf{V}| * |\mathbf{PR}|$ variables for m_{vi}, $|\mathbf{H}| * |\mathbf{PR}|$ variables for N_h^k, and one variable for sl. Hence, The variable number of the MILP is:

$$|\mathbf{V}|^2 + |\mathbf{V}| * (|\mathbf{PR}| + 1) + |\mathbf{H}| * |\mathbf{PR}| + |\mathbf{E}| + 1 \tag{16}$$

For the MILP model, there are $|\mathbf{V}|$ constraints for the constraint (1), $|\mathbf{V}| * (|\mathbf{V}| - 1)$ constraints for the constraints (2) and (3), $3 * |\mathbf{E}|$ for the constraints (4)–(6), $|\mathbf{V}| * |\mathbf{PR}|$ constraints for the constraint (8), $|\mathbf{V}| * (|\mathbf{V}| - 1) * |\mathbf{PR}|$ constraints for the constraints (9) and (10), $|\mathbf{V}| * (|\mathbf{V}| - 1) * |\mathbf{PR}|$ constraints for the constraints (11) and (12), $|\mathbf{V}| * |\mathbf{H}| * |\mathbf{PR}|$ constraints for the constraint (13), $|\mathbf{H}|$ constraints for the constraint (14), and $|\mathbf{V}_{exit}|$ constraints for the constraint (15). Hence, the constraint number of the MILP formulation is:

$$|\mathbf{V}|^2 * (2 * |\mathbf{PR}| + 1) + 3 * |\mathbf{E}| + |\mathbf{V}| * |\mathbf{PR}| * (|\mathbf{H}| - 1) + |\mathbf{H}| + |\mathbf{V}_{exit}| \tag{17}$$

4.4. Comment

The variable and constraint complexity of the proposed MILP formulation can be potentially reduced using the structural information of the DAG. One critical piece of information encoded in the DAG is the data precedences (direct and indirect) among the tasks of the application. For an edge of the DAG, its source task and the destination task have to be executed in the dedicated order, as captured by the constraint (6); hence, the number of variable z_{ab} can be reduced using such information, and the corresponding constraints can also be reduced. Besides, the same trick can be applied to tasks with similar indirect data precedence. Further, on the same reconfigurable region, the reconfiguration order of the tasks should also comply with the data precedence, which can be used to reduce the number of variable x_{ab} and the corresponding constraints.

5. Multi-Step Hybrid Algorithm

The former illustrated MILP-based approach paves the way to resolve the optimal solution for the Sw/Hw partitioning and scheduling problem; however, this exact approach does not work well for even medium-scale problems owing to its exponential increase of the time complexity with the problem scale. Its poor scalability motivated us to develop a low-time complexity method that produces reasonably good solutions for the target Sw/Hw partitioning and scheduling problem.

The proposed low-time complexity method is a multi-step hybrid approach. The whole problem is divided into a set of sub-problems that are easier to solve than the original problem; besides, the sub-problems are solved one by one with the partial result of the former one being the input of the next one. The proposed approach mixes the graph partitioning method that divides the whole problem into a set of nested sub-problems, and the former proposed MILP method is utilized for each sub-problem. The graph partitioning approach partitions the DAG of the application into a sequence of ordered nested sub-graphs, with the former sub-graph being contained in the next sub-graph. Hence, the problem is divided to solve each sub-problem corresponding to each sub-graph generated by the graph partitioning approach. Since the former sub-graph is contained in the next sub-graph, the complexity of the sub-problem increases with the order of the sub-problem using MILP. To reduce

the complexity, part of the result of the former sub-problem is used as the input of the next sub-problem, thus reducing the problem scale. In the following, we first introduce the structure of the multi-step hybrid algorithm, and then, a priority-based graph partitioning method that is used in the proposed multi-step hybrid algorithm is proposed.

5.1. Algorithm Structure

Algorithm 1 illustrates the structure of the multi-step hybrid algorithm.

Firstly, the DAG of the application is partitioned, generating an ordered set of sub-DAGs ($\mathbf{G}' = \{\mathbf{G}'_1, \mathbf{G}'_2, ..., \mathbf{G}'_{|\mathbf{G}'|}\}$) that satisfies the following requirements:

1. $\forall \mathbf{G}'_i, \mathbf{G}'_{i+1}, \mathbf{G}'_i \subset \mathbf{G}'_{i+1}$
2. $\forall \mathbf{G}'_i, \mathbf{G}'_j, j > i, \mathbf{G}'_i \nleftarrow \mathbf{G}'_j$

$\mathbf{G}'_i \subset \mathbf{G}'_{i+1}$ makes $\mathbf{G}'_i$ a sub-graph of $\mathbf{G}'_{i+1}$. Since $\mathbf{G}'_{i+1}$ is scheduled directly after $\mathbf{G}'_i$, the partial result yielded by scheduling $\mathbf{G}'_i$ can be used as the input of scheduling $\mathbf{G}'_{i+1}$. To better reuse the scheduling results for the former sub-problem, the condition $\mathbf{G}'_i \subset \mathbf{G}'_{i+1}$ has to be satisfied during graph partitioning.

The second condition means that there is no communication edge directed from $\mathbf{G}'_j$ to $\mathbf{G}'_i$. Note that tasks in $\mathbf{G}'_i$ are scheduled before those in $\mathbf{G}'_j$, as the proposed algorithm presents. Intuitively, it is better to schedule ancestors (a task is the ancestor of its child task) of the DAG that are more significant since their schedules affect more child tasks of the application; hence, this condition is utilized while partitioning the application.

Then, each sub-DAG $\mathbf{G}'_i \in \mathbf{G}'$ is scheduled using the MILP approach with the partial schedule result $psr_{\mathbf{G}'_{i-1}}$ of the former sub-DAG $\mathbf{G}'_{i-1}$ as the input. Note that sub-DAGs in $\mathbf{G}'$ are scheduled according to the order obtained by the graph partitioning method, and the first sub-DAG $\mathbf{G}'_1$ is scheduled without any partial schedule result. The partial schedule result is selected carefully, including the mapping information of each task, the execution and reconfiguration order of all tasks on each independent processing element (microprocessor and FPGA region), and the inter-processor communication indicator d_e. However, the exact task start execution time and reconfiguration time and the resource volume of each FPGA region are not fixed according to the former sub-problem, thus leaving more solution space for performance optimization.

Finally, scheduling of the last sub-DAG $\mathbf{G}'_{|\mathbf{G}'|}$ produces the result for the original problem.

Algorithm 1 MSHA ($\mathbf{G}$, $\mathbf{P}$)

1: partition the DAG $\mathbf{G}$ to an ordered set of sub-DAGs $\mathbf{G}' = \{\mathbf{G}'_1, \mathbf{G}'_2, ..., \mathbf{G}'_{|\mathbf{G}'|}\}$, $\forall \mathbf{G}'_i, \mathbf{G}'_{i+1}, \mathbf{G}'_i \subset \mathbf{G}'_{i+1}$; $\forall \mathbf{G}'_i, \mathbf{G}'_j, j > i, \mathbf{G}'_i \nleftarrow \mathbf{G}'_j$
2: **for** each $\mathbf{G}'_i \in \mathbf{G}'$ **do**
3: extract partial schedule result $psr_{\mathbf{G}'_{i-1}}$ for the sub-problem corresponding to $\mathbf{G}'_{i-1}$
4: solve the Sw/Hw partitioning and scheduling problem of $\mathbf{G}'_i$ on $\mathbf{P}$ with $psr_{\mathbf{G}'_{i-1}}$ as the input
5: **end for**

5.2. Priority-Based Graph Partitioning

As illustrated in the former subsection, partitioning the original problem into a set of small sub-problems is a key step of the multi-step hybrid algorithm. We coined the concept of sub-DAG, each of which corresponds to a sub-problem. The sub-DAGs are nested one-by-one; hence, the result of the former sub-problem can be partly transferred to the next sub-problem, thus reducing the complexity of solving the next sub-problem. In this subsection, we develop a priority-based graph partitioning method that partitions the original DAG into a set of nested sub-DAGs that meet our requirements.

To obtain the set of sub-DAGs and reserve the features needed by the proposed multi-step hybrid algorithm, we order the tasks of the original DAG with priorities that guarantee the topological order of the DAG. We use two kinds of static priorities: the first is the static bottom level (sbl), and the second is the static top level (stl). The static bottom level of a task refers to the maximum length of all paths that originate from the task to the exit task of the DAG without considering the communication cost. Since execution on the FPGA is of the most important for the performance (the task execution time on the microprocessor is 3–5 times slower than the Hw implementation on the FPGA [18]), the task cost on the FPGA is used for computing the priorities. A task with a high static bottom level implies that it may have more direct and/or indirect successors. Such a task is more important and should be scheduled earlier, i.e., be partitioned into the sub-DAGs ordered in the former part of the ordered sub-DAG list $\mathbf{G}'$. For two tasks a and b with $sbl_a > sbl_b$, either a and b are unconnected or b is a direct or indirect successor of task a, i.e., $a \nleftarrow b$. The static top level of a task refers to the maximum length of all paths that originate from the source task of the DAG to the task. A task with a large static top level may have more predecessors, and scheduling it earlier may potentially reduce the finish time of the last task on the longest path originating from it.

Algorithm 2 illustrates the details of the priority-based graph partitioning method. The algorithm has two inputs: the first is the DAG, and the second is the maximum task number of the sub-DAGs. The value of m should be set according to the solving time of the MILP formulation. Generally, the larger the value of m is, the higher the time complexity. According to our experiment, $m \leq 10$ works well. The PBGPalgorithm firstly computes the sbl and stl of the DAG recursively. Then, the tasks of the DAG are ordered with these two priorities. Based on the ordered task list, tasks in it are assigned to the sub-DAGs, as the first two for-loops show. Then, edges are added to each sub-DAG. Note that the output edge of a task in the sub-DAG is assigned to the next sub-DAG that contains the destination task of the edge, according to the first condition required to partition the DAG into sub-DAGs.

The sub-DAGs generated by Algorithm 2 comply with the conditions presented in Algorithm 1. For $\mathbf{G}'_i$, its tasks and edges also belong to $\mathbf{G}'_{i+1}$, as the second and the third for-loops show; hence, the first condition is satisfied. Since the proposed algorithm orders the tasks of the DAG according to the static bottom level, no edge would direct from a latter task to a former one in the sub-DAG list. While assigning tasks in the ordered task list to the sub-DAGs, the task with a smaller index in the task list is assigned to the sub-DAG with a smaller index, as the first for-loop shows; hence, $\forall \mathbf{G}'_i, \mathbf{G}'_j, j > i, \mathbf{G}'_i \nleftarrow \mathbf{G}'_j$.

Algorithm 2 PBGP (**G**, m)

compute the static bottom level of each task of **G**
compute the static top level of each task of **G**
order the tasks of the DAG in non-increasing order of the static bottom level; for tasks with the same static bottom level, order them in non-increasing order of the static top level, thus obtaining the task list $\mathbf{V}' = v'_1, v'_2, ..., v'_{|\mathbf{V}'|}$
for each $v'_i \in \mathbf{V}'$ **do**
    assign v'_i to sub-graph $\mathbf{G}'_{floor(i/m)+1}$
end for
for each $\mathbf{G}'_i \in \mathbf{G}'$ **do**
    assign $\mathbf{G}'_{i-1}$ to sub-graph $\mathbf{G}'_i$ (suppose $\mathbf{G}'_0 =$)
end for
for each $\mathbf{G}'_i \in \mathbf{G}'$ **do**
    for each $v' \in \mathbf{G}'_i$ **do**
        assign each input edge of v' to $\mathbf{G}'_i$
        for each $e \in outputEdge_{v'}$ **do**
            if $dst(e) \in \mathbf{G}'_i$ && $e \notin \mathbf{G}'_i$ **then**
                assign e to $\mathbf{G}'_i$
            end if
        end for
    end for
end for

To illustrate how the PBGP algorithm works, we take the application in Figure 2 with the parameters in Table 1 as an example. Table 2 shows the sbl and stl of each task of the DAG in Figure 2 and the sub-DAG(s) where each task is partitioned. Figure 4 shows the graph partitioning result using PBGP algorithm with $m = 5$. As shown in the graph, the DAG is divided into two sub-DAGs, i.e., $\mathbf{G}'_1$ and $\mathbf{G}'_2$. Each task and edge in $\mathbf{G}'_1$ is contained in $\mathbf{G}'_2$, and no edge is directed from $\mathbf{G}'_2$ to $\mathbf{G}'_1$, showing that the proposed PBGP algorithm complies the requirements of the graph partitioning approach. Note that such a partition also results in the optimal result as the MILP approach.

Table 2. Task priorities and graph partitioning of the example DAG. sbl, static bottom level; stl, static top level.

Task Name	sbl	stl	sub-DAG
n_0	13	0	$\mathbf{G}'_1, \mathbf{G}'_2$
n_1	10	0	$\mathbf{G}'_1, \mathbf{G}'_2$
n_2	8	0	$\mathbf{G}'_1, \mathbf{G}'_2$
n_3	8	5	$\mathbf{G}'_1, \mathbf{G}'_2$
n_4	6	2	$\mathbf{G}'_1, \mathbf{G}'_2$
n_5	5	8	$\mathbf{G}'_2$
n_6	4	8	$\mathbf{G}'_2$
n_7	2	11	$\mathbf{G}'_2$

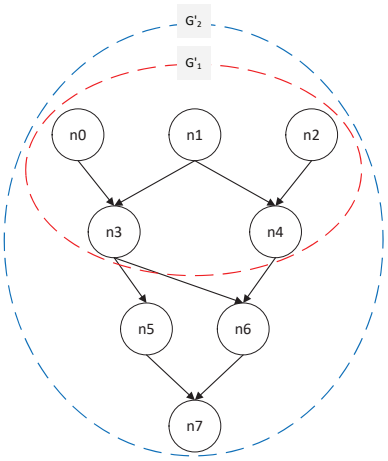

Figure 4. The graph partitioning result of the example DAG.

6. Experiments and Results

In this section, we evaluate the performance of the proposed methods experimentally. The proposed methods are compared with the cluster-based MILP (cMILP) [10] that combines ISBA (independent set-based algorithm) [23], since cMILP solves a similar problem of this work. In the following, the target platform and benchmarks used in the experiment are described; then, the experimental results are presented and discussed.

6.1. Target Platform Configuration

The target platform was composed of a CPU and a DPR-enabled FPGA, as illustrated in Section 3. Specifically, we targeted the Zynq-7000 series SoC platform, which provides an ARM Cortex-A9 CPU and a Xilinx XC7Z020 FPGA. Reconfiguration of the FPGA was carried out by the ICAP, which has a 32 bit input and output interface and is clocked at 100 MHz; hence, the reconfiguration speed achieved up to 3.2 G bit/s.

For each application to be scheduled on the target platform, the resource volume of the FPGA was set as 50% and 70% of the aggregate resource requirement of the application.

6.2. Benchmark

We used a set of real-life applications [32,33] to test the performance of the proposed methods. These real-life applications included Ferret, fast Fourier transform (FFT), Gauss elimination, Gauss–Jordan, the JPEG encoder, the Laplace equation, LUdecomposition, parallel Gauss elimination, parallel mean value analysis (MVA), parallel tiled QR factorization, the quadratic equation solver, cyber shake, epigenomics, LIGO, montage, SIPHT, molecular dynamics, channel equalizer, modem, MP3 decoder block parallelism, time division-synchronous code division multiple access (TDSCDMA), and wireless local area network (WLAN) 802.11a receiver. The parameters of these applications are shown in Table 3. As shown in the table, the task number of these applications ranged from eight to 50.

Table 3. Parameters of the set of real applications. MVA, mean value analysis.

App Name	Task Num	Edge Num
Ferret	20	19
FFT	16	20
Gauss elimination	14	19
Gauss–Jordan	15	20
JPEG encoder	8	9
Laplace equation	16	24
LUdecomposition	14	19
Parallel Gauss elimination	12	17
Parallel MVA	16	24
Parallel tiled QR factorization	14	21
Quadratic equation solver	15	15
Cyber shake	20	32
Epigenomics	20	22
LIGO	22	25
Montage	20	30
SIPHT	31	34
Molecular dynamics	41	71
Channel equalizer	14	21
Modem	50	86
MP3 decoder block parallelism	27	46
TDSCDMA	16	20
WLAN 802.a receiver	24	28

6.3. Experimental Results

Table 4 shows the Sw/Hw partitioning and scheduling result of the set of real applications shown in Table 3 on the platform with the resource volume of the FPGA being set as 70% of the aggregate resource requirement of the application.

Table 4 compares the proposed MILP and multi-step hybrid method with the cMILP. The performance comparison was carried out from two aspects, i.e., the schedule length (SL) and the time used to find the solution. Since the MILP-based method is hard to solve even for medium-scale problems, we set a timeout of 2 h for the MILP solver. When timeout happens, the best feasible solution that has been found by the MILP solver is used as the final solution, and the time is set as timeout. When timeout does not happen, the optimal solution is found, and the time represents that used to find the optimal result.

Table 4. Sw/Hw partitioning and scheduling result of the set of real applications on the platform with 70% resource. cMILP, cluster-based MILP; SL, schedule length.

App Name	MILP		Hybrid		cMILP	
	SL	**Time**	**SL**	**Time**	**SL**	**Time**
Ferret	**63**	timeout	64	**4.180**	70	3810.600
FFT	**42**	timeout	47	**4.046**	52	41.320
Gauss elimination	**51**	2191.970	53	**0.634**	72	2.100
Gauss–Jordan	**61**	timeout	76	**3.774**	75	102.150
JPEG encoder	**40**	0.190	**40**	0.245	**40**	**0.030**
Laplace equation	**88**	timeout	90	**1.643**	103	6.860
LU decomposition	**66**	6292.630	73	3.104	83	**3.030**
Parallel Gauss elimination	**45**	18.960	47	0.976	61	**0.730**
Parallel MVA	**60**	timeout	64	**1.585**	78	3.810
Parallel tiled QR factorization	**53**	33.290	58	**0.570**	65	1.410
Quadratic equation solver	**44**	timeout	48	**4.233**	54	12.510
Cyber shake	82	timeout	**75**	**31.997**	78	timeout
Epigenomics	88	timeout	**78**	**4.196**	84	timeout
LIGO	92	timeout	93	**9.477**	**90**	timeout
Montage	**96**	timeout	**96**	**6.536**	110	1455.890
SIPHT	276	timeout	**106**	**1821.770**	154	timeout
Molecular dynamics	469	timeout	**164**	**87.734**	260	timeout
Channel equalizer	**63**	timeout	**63**	2.264	72	**0.470**
Modem	-	timeout	**220**	**300.149**	612	timeout
MP3 decoder block parallelism	140	timeout	143	**136.323**	**136**	timeout
TDSCDMA	**74**	timeout	78	**1.282**	101	3.190
WLAN 802.11a receiver	149	timeout	**138**	**2.643**	160	timeout

The proposed MILP approach is competitive in terms of schedule length. As shown in the table, for 14 applications among the 22 applications, the MILP produces schedules that are better than or equal to the other two methods. What is more, the proposed MILP approach is able to find the optimal solution for the Sw/Hw partitioning and scheduling problem. As shown in the table, for all problem instances without timeout, i.e., Gauss elimination, JPEG encoder, LU decomposition, parallel Gauss elimination, and parallel tiled QR factorization, the proposed MILP approach produces the Sw/Hw partitioning and scheduling solution with the smallest schedule length, e.g., for the Gauss elimination, the SL of the MILP is 51, while that of the hybrid approach and cMILP is 53 and 72, respectively. Note that for the JPEG encoder, all three methods can produce the optimal solution. For those problem instances when timeout happens, the SL of the MILP approach may be worse than the other two methods owing to the large solution space to be searched. What is more, among the three methods, the proposed MILP approach does not achieve optimality in terms of time complexity for any problem instance. For the modem with 50 tasks, the MILP cannot find any feasible solution when timeout happens, demonstrating the complexity of the MILP.

Among the 22 applications, the multi-step hybrid approach produces the best solution among the three methods for nine applications, i.e., JPEG encoder, cyber shake, epigenomics, montage, SIPHT, molecular dynamics, channel equalizer, modem, and WLAN 802.11a receiver. What is more, for seven applications, i.e., cyber shake, epigenomics, montage, SIPHT, molecular dynamics, modem, and WLAN 802.11a receiver, the multi-step hybrid approach produces the best solution in terms of both SL and time complexity. For other cases, the multi-step hybrid approach also shows great advantages in balancing the SL and time complexity, e.g., for Ferret, the multi-step hybrid approach produces the schedule with SL being 64 using only 4.180 s, while the MILP produces a better schedule with more than 2 h (timeout); further, cMILP takes 3810.6 s to produce a schedule with the SL being 70, which is far worse than the hybrid.

The multi-step hybrid approach performs better than or equal to cMILP for 21 applications in terms of SL. Among these 21 applications, the hybrid approach performs better than cMILP for 17 applications in terms of time complexity; while for the other four applications, i.e., JPEG encoder, LU decomposition, parallel Gauss elimination, and channel equalizer, the hybrid approach is still competitive in terms of time complexity. For these applications, the time complexity of the hybrid approach is higher than cMILP by several seconds at most, which is negligible, e.g., for channel equalizer, the time for cMILP is 0.470 s, while that for the hybrid approach is 2.264 s. Note that for the other three applications, the difference in the time complexity of the hybrid approach and cMILP is much smaller than for channel equalizer.

The competitiveness of cMILP in terms of SL can only be observed for three applications, i.e., JPEG encoder, LIGO, and MP3 decoder block parallelism. For JPEG encoder, since its task number is only eight, and all three methods produce the optimal result in terms of SL using less than 1 s, making the competitiveness of cMILP negligible. For LIGO, cMILP finds the solution with the SL being 90 using more than 2 h (timeout), while the hybrid approach takes 9.477 s to yield a schedule with the SL being 93, showing that the hybrid approach achieves a better balance between SL and time used. For MP3 decoder block parallelism, cMILP produces a solution with the SL being 136 using more than 2 h (timeout), while the hybrid approach takes 9.477 s to yield a schedule with the SL being 143 using about 2 min. Though the SL of the hybrid approach is 4.8% worse than cMILP, its time complexity is hundreds of times less than cMILP.

Table 5 shows the Sw/Hw partitioning and scheduling result of the set of real applications shown in Table 3 on the platform with the resource volume of the FPGA being set as 50% of the aggregate resource requirement of the application. As the former experiment, a timeout of 2 h is set for the MILP solver, and the best feasible solution that was found by MILP solver is used as the solution when timeout happens.

Similar to Table 4, the data in Table 5 show the optimality of the proposed MILP approach and the advantage of the proposed multi-step hybrid method in achieving a good balance between schedule length and time complexity. Among the 22 applications, MILP produces the best solution in terms of SL for 13 applications while costing the longest time among the three methods. The multi-step hybrid method shows a comparative advantage in terms of SL and time among the three methods for 14 and 11 applications, respectively. However, cMILP only produces the shortest SL for only one application, i.e., LIGO. What is more, for this application, the multi-step hybrid method produces the same result with much less time.

Comparing Tables 4 and 5, we can find that the same schedule performance can be gained with less resources for some cases, e.g., for Gauss elimination, the MILP approach finds the optimal result for the cases in Tables 4 and 5 since no timeout has happened. However, the schedule lengths are both 51. Note that it is possible to use the proposed method to optimize the resource number of the platform by iteratively executing the algorithm for the problem with various resource volumes. However, it remains for further study to minimize the resource of the platform while respecting the performance requirement of the application.

Table 5. Sw/Hw partitioning and scheduling result of the set of real applications on the platform with 50% resource.

App Name	MILP		Hybrid		cMILP	
	SL	Time	SL	Time	SL	Time
Ferret	**62**	timeout	**62**	**61.079**	70	4812.580
FFT	**43**	timeout	44	194.704	52	**42.540**
Gauss elimination	**51**	36.650	**51**	2.153	73	**0.880**
Gauss–Jordan	**61**	timeout	68	286.532	75	**104.960**
JPEG encoder	**43**	0.260	**43**	0.287	45	**0.060**
Laplace Equation	**88**	timeout	**88**	**4.743**	103	7.020
LU decomposition	**66**	2622.600	70	16.004	83	**3.020**
Parallel Gauss elimination	**45**	19.250	51	6.116	61	**0.730**
Parallel MVA	**60**	timeout	63	102.353	78	**8.750**
Parallel tiled QR factorization	**53**	75.110	**53**	4.841	65	**3.220**
Quadratic equation solver	**44**	timeout	45	162.174	54	**30.830**
Cyber shake	94	timeout	**73**	**190.580**	86	timeout
Epigenomics	88	timeout	**82**	**132.572**	84	timeout
LIGO	92	timeout	**89**	**104.929**	**89**	timeout
Montage	100	timeout	**95**	**29.636**	110	1348.030
SIPHT	247	timeout	**106**	**1817.340**	154	timeout
Molecular dynamics	469	timeout	**165**	**2408.910**	260	timeout
Channel equalizer	**63**	5359.830	64	7.680	72	**0.230**
Modem	-	timeout	**233**	**1752.740**	602	timeout
MP3 decoder block parallelism	140	timeout	**131**	**636.780**	139	timeout
TDSCDMA	**74**	timeout	78	9.796	101	**3.270**
WLAN 802.11a receiver	149	timeout	**144**	**27.217**	160	timeout

7. Conclusions

Sw/Hw partitioning and scheduling comprise a critical technique for efficient use of SoCs based on dynamical partial reconfigurable FPGAs. Scheduling execution and reconfiguration of the application task together with mapping tasks of the application to heterogeneous processing elements and reconfigurable regions are important for the timing performance of the system. This paper proposed a novel MILP formulation for the Sw/Hw partitioning and scheduling on DPR FPGA-based SoCs. MILP is an exact approach and is able to produce the optimal solution with the sacrifice of time. However, owing to the formulation complexity, MILP is hard to solve even for medium-sized problems. Hence, a multi-step hybrid approach is also proposed to achieve a balance between the solution quality and time complexity. A set of real-life applications is used for performance evaluation, and the experimental results demonstrate the effectiveness of the proposed methods.

Author Contributions: Conceptualization, Q.T.; methodology, Q.T.; validation, Q.T., B.G., and Z.W.; writing, original draft preparation, Q.T.; writing, review and editing, B.G. and Z.W. All authors read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rossi, E.; Damschen, M.; Bauer, L.; Buttazzo, G.; Henkel, J. Preemption of the Partial Reconfiguration Process to Enable Real-Time Computing With FPGAs. *ACM Trans. Reconfig. Technol. Syst.* **2018**, *11*, 1–24. [CrossRef]
2. Biondi, A.; Balsini, A.; Pagani, M.; Rossi, E.; Marinoni, M.; Buttazzo, G. A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs. In Proceedings of the 2016 IEEE Real-Time Systems Symposium (RTSS), Porto, Portugal, 29 November–2 December 2016; pp. 1–12. [CrossRef]

3. Saha, S.; Sarkar, A.; Chakrabarti, A. Scheduling dynamic hard real-time task sets on fully and partially reconfigurable platforms. *IEEE Embed. Syst. Lett.* **2015**, *7*, 23–26. [CrossRef]
4. Cordone, R.; Redaelli, F.; Redaelli, M.A.; Santambrogio, M.D.; Sciuto, D. Partitioning and scheduling of task graphs on partially dynamically reconfigurableFPGAs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2009**, *28*, 662–675. [CrossRef]
5. Deiana, E.A.; Rabozzi, M.; Cattaneo, R.; Santambrogio, M.D. A multiobjective reconfiguration-aware scheduler for FPGA-based heterogeneous architectures. In Proceedings of the 2015 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015, Mexico City, Mexico, 7–9 December 2015; pp. 1–6. [CrossRef]
6. Beretta, I.; Rana, V.; Atienza, D.; Sciuto, D. A mapping flow for dynamically reconfigurable multi-core system-on-chip design. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2011**, *30*, 1211–1224. [CrossRef]
7. Beretta, I.; Rana, V.; Atienza, D.; Sciuto, D. Island-Based Adaptable Embedded System Design. *IEEE Embed. Syst. Lett.* **2011**, *3*, 53–57. [CrossRef]
8. Clemente, J.A.; Rana, V.; Sciuto, D.; Beretta, I.; Atienza, D. A hybrid mapping-scheduling technique for dynamically reconfigurable hardware. In Proceedings of the 21st International Conference on Field Programmable Logic and Applications, FPL 2011, Chania, Greece, 5–7 September 2011; pp. 177–180. [CrossRef]
9. Clemente, J.A.; Beretta, I.; Rana, V.; Atienza, D.; Sciuto, D.; Clemente, J.A.; Sciuto, D. A Mapping-Scheduling Algorithm for Hardware Acceleration on Reconfigurable Platforms. *ACM Trans. Reconfig. Technol. Syst.* **2014**, *7*, 1–27. [CrossRef]
10. Ma, Y.; Liu, J.; Zhang, C.; Luk, W. HW/SW partitioning for region-based dynamic partial reconfigurable FPGAs. In Proceedings of the 2014 32nd IEEE International Conference on Computer Design, ICCD 2014, Seoul, Korea, 19–22 October 2014; pp. 470–476. [CrossRef]
11. Cong, J.; Liu, B.; Neuendorffer, S.; Noguera, J.; Vissers, K.; Zhang, Z. High-level synthesis for FPGAs: From prototyping to deployment. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2011**, *30*, 473–491. [CrossRef]
12. Nane, R.; Sima, V.M.; Pilato, C.; Choi, J.; Fort, B.; Canis, A.; Chen, Y.T.; Hsiao, H.; Brown, S.; Ferrandi, F.; et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2016**, *35*, 1591–1604. [CrossRef]
13. Meeus, W.; Van Beeck, K.; Goedemé, T.; Meel, J.; Stroobandt, D. An overview of today's high-level synthesis tools. *Des. Autom. Embed. Syst.* **2012**, *16*, 31–51. [CrossRef]
14. Pezzarossa, L.; Kristensen, A.; Schoeberl, M.; Sparso, J. Can real-time systems benefit from dynamic partial reconfiguration? In Proceedings of the 2017 IEEE Nordic Circuits and Systems Conference, NORCAS 2017: NORCHIP and International Symposium of System-on-Chip, Linkoping, Sweden, 23–25 October 2017; doi:10.1109/NORCHIP.2017.8124984. [CrossRef]
15. Wang, G.; Liu, S.; Sun, J. A dynamic partial recon fi gurable system with combined task allocation method to improve the reliability of FPGA. *Microelectron. Reliab.* **2018**, *83*, 14–24. [CrossRef]
16. Dai, G.; Shan, Y.; Chen, F.; Wang, Y.; Wang, K.; Yang, H. Online scheduling for FPGA computation in the Cloud. In Proceedings of the 2014 International Conference on Field-Programmable Technology, FPT 2014, Shanghai, China, 10–12 December 2014; pp. 330–333. [CrossRef]
17. Fekete, S.P.; Kohler, E.; Teich, J. Optimal FPGA module placement with temporal precedence constraints. In Proceedings of the Design, Automation and Test in Europe, Munich, Germany, 13–16 March 2001; pp. 658–665. [CrossRef]
18. Banerjee, S.; Bozorgzadeh, E.; Dutt, N.D. Integrating physical constraints in HW-SW partitioning for architectures with partial dynamic reconfiguration. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2006**, *14*, 1189–1202. [CrossRef]
19. Ferrandi, F.; Lanzi, P.L.; Pilato, C.; Sciuto, D.; Tumeo, A. Ant colony optimization for mapping, scheduling and placing in reconfigurable systems. In Proceedings of the 2013 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2013, Torino, Italy, 24–27 June 2013; pp. 47–54. [CrossRef]
20. Banerjee, S.; Bozorgzadeh, E.; Dutt, N. Exploiting application data-parallelism on dynamically reconfigurable architectures: Placement and architectural considerations. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2009**, *17*, 234–247. [CrossRef]

21. Purgato, A.; Tantillo, D.; Rabozzi, M.; Sciuto, D.; Santambrogio, M.D. Resource-efficient scheduling for partially-reconfigurable FPGA-based systems. In Proceedings of the 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, 23–27 May 2016; pp. 189–197. [CrossRef]
22. Rabozzi, M.; Lillis, J.; Santambrogio, M.D. Floorplanning for partially-reconfigurable FPGA systems via mixed-integer linear programming. In Proceedings of the 2014 IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines, FCCM 2014, Boston, MA, USA, 11–13 May 2014; pp. 186–193. [CrossRef]
23. He, R.; Ma, Y.; Zhao, K.; Bian, J. ISBA: An Independent Set-Based Algorithm for Automated Partial Reconfiguration Module Generation. In Proceedings of the International Conference on Computer-Aided Design, San Jose, CA, USA, 12–14 September 2012; pp. 500–507.
24. Chen, S.; Huang, J.; Xu, X.; Ding, B.; Xu, Q. Integrated Optimization of Partitioning, Scheduling, and Floorplanning for Partially Dynamically Reconfigurable Systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *39*, 199–212. [CrossRef]
25. Valente, G.; Mascio, T.D.; Andrea, G.D.; Pomante, L. Dynamic Partial Reconfiguration Profitabilit for Real-Time Systems. *IEEE Embed. Syst. Lett.* **2020**, *663*, 1–4. [CrossRef]
26. Naouss, M.; Hannachi, M. Optimized Placement Approach on Reconfigurable FPGA. *Int. J. Model. Optim.* **2019**, *9*, 82–86. [CrossRef]
27. Rabozzi, M.; Member, S.; Durelli, G.C.; Member, S.; Miele, A.; Lillis, J.; Santambrogio, M.D.; Member, S. Floorplanning Automation for Feasible Placements Generation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 151–164. [CrossRef]
28. Montone, A.; Santambrogio, M.D.; Sciuto, D.; Memik, S.O. Placement and Floorplanning in Dynamically Reconfigurable FPGAs. *ACM Trans. Reconfig. Technol. Syst.* **2010**, *3*, 1–34. [CrossRef]
29. Tang, Q.; Wu, S.F.; Shi, J.W.; Wei, J.B. Optimization of Duplication-Based Schedules on Network-on-Chip Based Multi-Processor System-on-Chips. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 826–837. [CrossRef]
30. Sinnen, O. *Task Scheduling for Parallel Systems*; John Wiley & Sons: Hoboken, NJ, USA, 2007; Volume 60.
31. Vipin, K.; Fahmy, S.A. FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. *ACM Comput. Surv.* **2018**, *51*, 1–39. [CrossRef]
32. Canon, L.C.; Sayah, M.E. A Comparison of Random Task Graph Generation Methods for Scheduling Problems. *arXiv* **2019**, arXiv:1902.05808v1.
33. Tang, Q.; Basten, T.; Geilen, M.; Stuijk, S.; Wei, J.B. Mapping of Synchronous Dataflow Graphs on MPSoCs Based on Parallelism Enhancement. Journal of Parallel and Distributed Computing. *J. Parallel Distrib. Comput.* **2017**, *101*, 79–91. [CrossRef]