IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 25, 2020, accepted October 5, 2020, date of publication October 9, 2020, date of current version October 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3029838

Multi-Shape Task Placement Algorithm Based on Low Fragmentation Resource Management on 2D Heterogeneous Dynamic Partial Reconfigurable Devices

TINGYU ZHOU[1], (Member, IEEE), TIEYUAN PAN[2], MICHAEL CONRAD MEYER[1], (Member, IEEE), YIPING DONG[3], AND TAKAHIRO WATANABE[1], (Life Member, IEEE)

[1]Graduate School of Information, Production and Systems, Waseda University, Kitakyushu 8080135, Japan
[2]DENSO Corporation, Kariya 4488661, Japan
[3]CKS Company Ltd., Wuxi 214072, China

Corresponding author: Tingyu Zhou (shu-yu@asagi.waseda.jp)

This work was supported in part by the Waseda University Tokutei Kadai under Grant 2018K-301 and Grant 2020C-661, and in part by the Int'l Sci-Tech Cooperation Projects under Grant BZ2018031.

ABSTRACT The Dynamic Partial Reconfiguration function of reconfigurable devices permits tasks to be performed simultaneously on a single device. Nevertheless, task placement and resource management problems emerge with the parallelism of reconfigurable devices. Traditional task placement algorithms are based on the assumption of a homogeneous architecture and simplify the task as a rectangular shape, which inevitably results in internal unused areas, thereby wasting a significant amount of programmable resources. To address the resource waste that comes with the assumption of a rectangular task shape and improve the placement quality, we adopted an interval list set to manage available programmable resources for a heterogeneous reconfigurable device and proposed an interval-based placement algorithm combined with a low-fragmentation selection strategy targeting the placement problem of multi-shape tasks. The efficiency of the proposed approach is proved theoretically, and simulation results demonstrate that the rejection ratio is decreased by at least 8.9% with an average fragmentation reduction of 18.1%.

INDEX TERMS Multi-shape task, task placement algorithm, dynamic partial reconfiguration, heterogeneous reconfigurable device.

I. INTRODUCTION

A. BACKGROUND

In recent years, with the increase in transistor integration and shrink of transistor size, the improvement of chip performance has slowed down [1]–[4]. The advent of the era of Big Data has forced researchers to urgently explore new methods to accelerate the analysis and processing of massive amounts of data [5]. In the high-performance computing field, researchers are driving the development of multi-core processing platforms to leverage hardware acceleration within heterogeneous architectures comprising of flexible combinations of Central Processing Units (CPUs), Graphics Processing Units (GPUs), Application-specific Integrated Circuits (ASICs), and Field-programmable Gate Arrays (FPGAs) [6]. An FPGA is a prefabricated reconfigurable device that is more widely built on the hardware acceleration platform compared to GPUs and ASICs due to its performance, flexibility, and scale [7]. For instance, the Azure Services Platform developed by Microsoft exploits an FPGA-enabled hardware architecture to support the high-performance computing of image processing, real-time Artificial Intelligence (AI) calculations, and so on [8].

Generally, a reconfigurable device is comprised of an array of programmable resources, including Configurable Logic Blocks (CLBs), Block Random Access Memories (BRAMs), and Digital Signal Processors (DSPs), which enables implementation of different functions based on consumers

The associate editor coordinating the review of this manuscript and approving it for publication was Sneh Saurabh.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

requirements by entirely or partially reconfiguring those programmable circuits [9]. According to the status of the device when its circuits are programmed, the partial reconfiguration scheme is classified into static and dynamic partial reconfigurations.

1) Static partial reconfiguration – the reconfigurable device is shutdown and will be brought up after the partial reconfiguration is completed;
2) Dynamic partial reconfiguration – the reconfigurable device is still active during the reconfiguration process – permits changing part of the reconfigurable device while the rest parts are still executing.

In a word, the dynamic partial reconfiguration scheme enables part of device circuits to be reconfigured at run-time without disturbing the execution of other parts, so that the flexibility and performance of the reconfigurable device are fully improved. In this work, our discussion focuses on Dynamic Partial Reconfigurable (DPR) devices, especially heterogeneous devices.

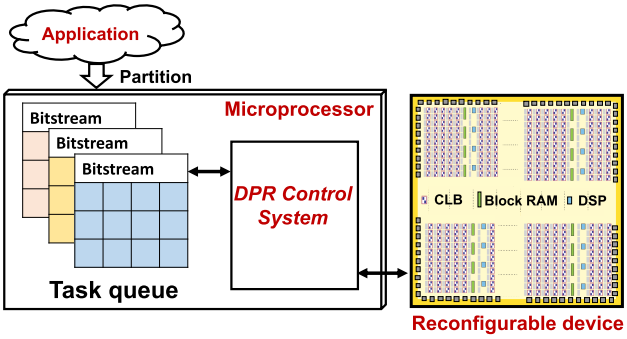


FIGURE 1. Application execution process under the cooperation of a microprocessor and reconfigurable device.

B. MOTIVATION

As shown in Fig. 1, when an application attempts to be executed on a DPR device in cooperation with a DPR control system in a microprocessor, the application firstly is divided into various hardware tasks. Each task is placed on the DPR device as the smallest execution unit and executed. Furthermore, the task is removed from the device once it ends execution so that the programmable resources occupied by the task can be released and reused by other tasks. Under the dynamic partial reconfiguration scheme, the task placement and removal processes have been ongoing at run-time so the resource status can be changed at any time. Therefore, the advantage of the DPR device comes with two critical issues to be solved, which are shown as follows:

1) Task placement - decide where to place each hardware task to ensure the maximum use of limited programmable resources;
2) Resource management - precisely record and update the status of programmable resources on the DPR device efficiently.

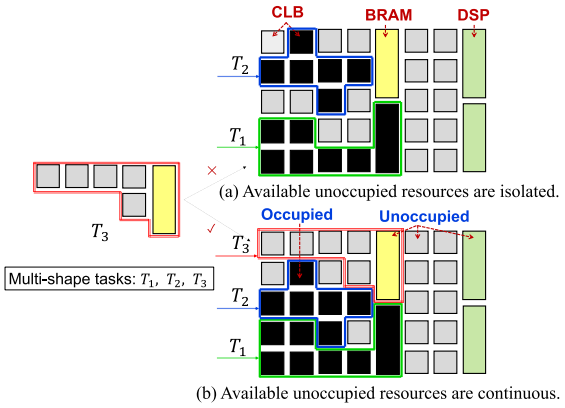


FIGURE 2. Example of multi-shape task (T_1, T_2, and T_3) placement on a 2D heterogeneous DPR device with different fragmentation degrees.

Task placement and resource management problems directly affect the performance of a DPR device. As shown in Fig. 2, in addition to the black areas that have been occupied by executing tasks T_1 and T_2, regions of other colors indicate feasible resources, which are currently unoccupied by any task. Compared to Fig. 2(b), unoccupied areas in Fig. 2(a) are isolated (high fragmentation, described in Section VI-B2 in detail) due to the different placement location of T_2, it is difficult to accommodate the upcoming new task T_3 in advance. The fragmentation problem caused by the placement decision of T_2 in Fig. 2(a) decreases the resource utilization of the DPR device. Moreover, during application execution, the DPR device always spends time monitoring the status of programmable resources since the placement and removal of tasks have been ongoing. In order to avoid the DPR scheme degrading the performance of the reconfigurable device, the DPR control system has to decide a better task placement location for each task and update the programmable resource status efficiently.

Various approaches have been put forward to solve the task placement and resource management problems for a 2D or 3D DPR device [10]–[17]. However, most existing algorithms assume that the task is a regular rectangle shape that results in unnecessary resource waste. On the other hand, existing resource management methods proposed in previous research [12] are based on a homogeneous architecture for a DPR device, which only contains CLB resources on the device. The simplification of device architecture makes it impossible to apply the resource management methods to heterogeneous DPR devices.

C. CONTRIBUTIONS AND ORGANIZATIONS

In this work, we apply a data structure named *interval list set* to record and update the status of limited programmable resources for a 2D heterogeneous DPR device at run-time. However, in the task placement phase, the proposed resource management method brings another issue; which is, when a new task comes, how to search for an unoccupied location for

the new task to execute based on the interval list set. Notably, the search process is more complicated for a multi-shape task, which is needed for the recent DPR device as described later.

In this paper, we propose an interval-based task placement algorithm (IPAL) to quickly process the interval list set to find all feasible unoccupied locations to execute the multi-shape task. Furthermore, a continuous block aware placement strategy (CBAPS) is proposed to select one proper placement location with low fragmentation when several available locations exist. Our contributions are as follows.

1) We outline a novel *continuous block list* to model a heterogeneous multi-shape task.
2) An *interval list set* is applied to manage programmable resources on the 2D heterogeneous DPR device efficiently.
3) We propose an interval-based placement algorithm (IPAL) to search for available locations for a targeted multi-shape task.
4) We propose a placement location selection strategy: CBAPS, to decrease the resource fragmentation problem.
5) We verify the efficiency of the proposed approach theoretically.
6) Finally, we demonstrate the effectiveness of the proposed approach by comparing against various state-of-the-art solutions in terms of runtime overhead, degree of fragmentation, and placement quality through simulation studies.

The remainder of this paper is organized as follows. In Section II, we introduce related works for the task placement problem and deficiencies of existing algorithms. Section III presents a brief introduction to the basic models proposed in this paper. Details of our proposed multi-shape task placement algorithm, selection strategy, and resource management method are described in detail in Section IV. In Section V, the proposed approach is theoretically analyzed, and in Section VI, it is evaluated by simulation. Finally, we conclude in Section VII.

II. RELATED WORKS

The problem of searching for the optimal execution locations on a DPR device for a series of tasks at run-time is known to be an NP-hard problem [18]. Based on different resource management methods, various task placement algorithms have been proposed for the 2D DPR device [10]–[15].

The most commonly applied resource management method for the DPR device is to regard programmable resource arrays as a resource matrix. The status of each programmable resource is represented through the value of an element in the matrix. A task can find an available location by traversing the resource matrix. Furthermore, once a task is assigned or removed from the DPR device, merely the value of each element occupied by the task needs to be updated to record and trace the device resource status at run-time. The above mentioned matrix-based resource management method can be applied to various kinds of DPR device architectures and task models, thus various placement strategies or algorithms are proposed based on the resource matrix.

For homogeneous DPR device architectures, to keep the maximum unoccupied area in the center of the 2D DPR device, Marconi *et al.* [10] presented a quad-corner (QC) task placement strategy that tries to place new tasks in the four corners of the device as much as possible according to the task size as follows: upper-left corner first (for very large tasks), upper-right corner first (for large tasks), lower-right corner first (for medium size tasks), and lower-left corner first (for small tasks). The QC strategy is highly efficient since the strategy searches candidates in the corner corresponding to the size of the new task to find the available location for the task, rather than traversing the entire resource matrix. However, the limited search candidates may cause a targeted task cannot find an unoccupied location even though there is enough space to accommodate the new task [16]. This is an algorithm that sacrifices the quality of placement in exchange for speed.

For heterogeneous DPR device architectures, Enemali *et al.* [11] proposed an expanding the un-usable area strategy (EUAS) that could render portions of the chip unusable to future rectangular hardware tasks. For the homogeneous multi-shape task model, both Esmaeildoust *et al.* [19] and Wang *et al.* [12] proposed approaches which are based on a best-fit placement strategy, which means that traversing the resource matrix to select the location with maximal adjacent value with other tasks or device boundaries to place a targeted task. Once the targeted task cannot find an available location, Esmaeildoust *et al.* [19] proposed a relocation process, that selects a placed task to relocate to a new placement location to create enough available resources for the current targeted task. However, the relocated task needs to search for an available location again and restart, which adversely affects the overall execution time of the entire application. Wang *et al.* [12] proposed a best-fit task shape transformation (BFT) strategy for the homogeneous multi-shape task placement, where an IP core is regarded as the basic unit and the shape of a non-rectangular task is converted by changing the relative position between IP cores to obtain better placement results. Unfortunately, the shape transformation process also requires additional redundant time cost.

In a word, although it is easy to apply a matrix to manage programmable resources, the non-negligible time consumption that traversing a part or entire resource matrix to find an available location for each task brings is disadvantageous for the whole computing system.

In literature, a maximal empty rectangle (MER) list [13]–[15] is applied to represent the unoccupied programmable resources on the 2D homogeneous DPR device. MER is a rectangle that cannot be fully covered by other rectangles, so as long as there are sufficient resources, the targeted task can find an available location by traversing the MER list. Nevertheless, no matter the QC strategy, EUAS, and MER list, the hardware task is assumed to be rectangular to simplify the task placement problem, which produces

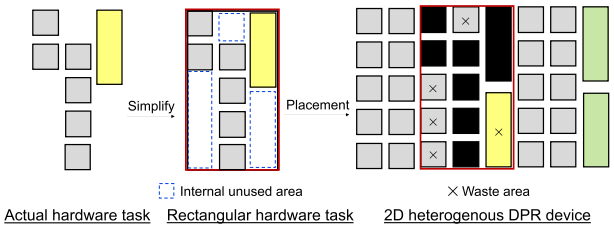


FIGURE 3. Waste areas due to the assumption of rectangular hardware tasks.

some unavoidable internal unused areas (shown in Fig. 3) to fit the actual logic of the hardware task into a rectangular area. Programmable resources in the unoccupied internal area get wasted because they are treated as occupied resources according to the rectangular task model so that blocking the possibility of the space being assigned to other tasks. On the other hand, most existing approaches [10], [12]–[16], [19] consider that the DPR device only contains one kind of programmable resource: CLB. However, to improve the computing performance, the recent DPR device architectures (e.g., Xilinx 7 Series FPGAs, Zynq-7000 SoC, etc.) are all heterogeneous, which means that the device is comprised of multiple programmable resources [20]–[22]. Therefore, it is necessary to propose a task placement algorithm and resource management method for multi-shape tasks on the 2D heterogeneous DPR device.

III. PROBLEM FORMULATION

This section describes the 2D heterogeneous DPR device, multi-shape task, and interval list set models assumed in this paper.

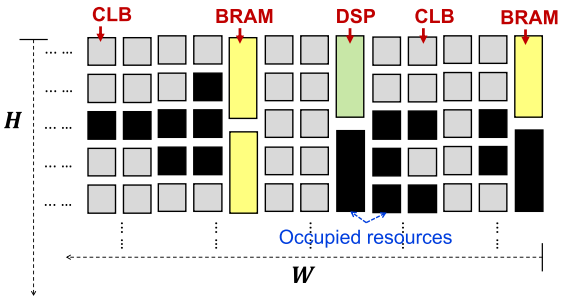


FIGURE 4. Example of a 2D heterogeneous DPR device $F\ (W, H)$. (W: total number of columns; H: total number of rows).

A. TWO-DIMENSIONAL HETEROGENEOUS DYNAMIC PARTIAL RECONFIGURABLE DEVICE MODEL

In this work, we build a 2D heterogeneous DPR device model based on the ZC702 FPGA [22] and three kinds of programmable resources are distributed in row and column. An example of the device is illustrated in Fig. 4. A DPR device is denoted as $F\ (W, H)$, which means that there are W columns for three kinds of resources and H rows for CLBs on the device. Note that CLB size is fixed to 1×1, while DSP and BRAM size is 1×2.5. The coordinate of the bottom-leftmost corner of $F\ (W, H)$ is regarded as $(0, 0)$.

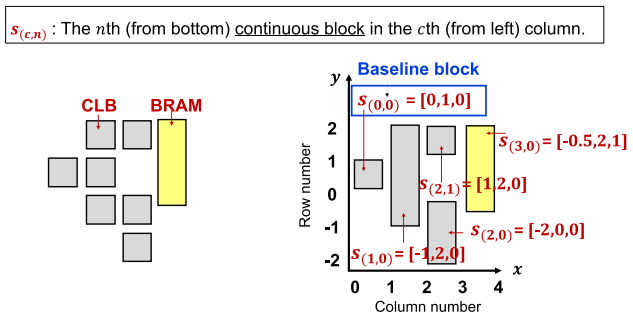
Programmable resources on the DPR device model have two statuses: *unoccupied* and *occupied*. Unoccupied means that the resource is idle and can be assigned to new tasks for execution, while the occupied status means that the resource is already used by an executing task so it prohibits other tasks from being placed at that occupied location. Black-colored regions shown in Fig. 4 represent occupied resources.

B. HETEROGENEOUS MULTI-SHAPE TASK MODEL

A heterogeneous multi-shape task $T\ (a, e, d, S)$ is circuits executable on a 2D heterogeneous DPR device $F\ (W, H)$, where a, e and d are the task arrival time, execution time, and deadline, respectively. If a task cannot be executed until the time $(a + d)$, which is the sum of arrival time a and deadline d, the task cannot be executed anymore. The process is called *rejection* and the higher the proportion of rejected tasks is, the worse the placement quality of the algorithm is.

Based on the proposed 2D heterogeneous DPR device model, different kinds of programmable resources are lined up in columns (shown in Fig. 4). Therefore, a *continuous block list* $S = \{s_{(c,n)} | c = 0, 1, \ldots$ and $n = 0, 1, \ldots\}$ is proposed to represent programmable resources required by the multi-shape task, which is defined as follows:

Definition 1: In each column, the maximal adjacent resources required by the multi-shape task T are defined as a continuous block $s_{(c,n)}$, where c is the column number starting from 0 on the leftmost position, and n is the consecutive block stack in the cth column starting from 0 on the bottom. The nth largest consecutive block in the cth column of the task is denoted as $s_{(c,n)} = [sb, st, sk]$, where sb and st are the bottom and top values of $s_{(c,n)}$ on the y-axis respectively. sk is the type of programmable resource in the cth column: $sk = 0$ for CLB; $sk = 1$ for BRAM; $sk = 2$ for DSP.



(a) Multi-shape task T.

(b) Continuous block list S, $S = \{s_{(0,0)}, s_{(1,0)}, s_{(2,0)}, s_{(2,1)}, s_{(3,0)}\}$.

FIGURE 5. Example of heterogeneous multi-shape task model. (Continuous block $s_{(c,n)} = [sb, st, sk] \rightarrow$ sb: bottom; st: top; sk: type, as defined in Definition 1).

Fig. 5 shows an example of the continuous block list representation method. Note that, $s_{(0,0)}$ (the zeroth continuous block in the zeroth column) is considered as a *baseline block* within the continuous block list, which means that the

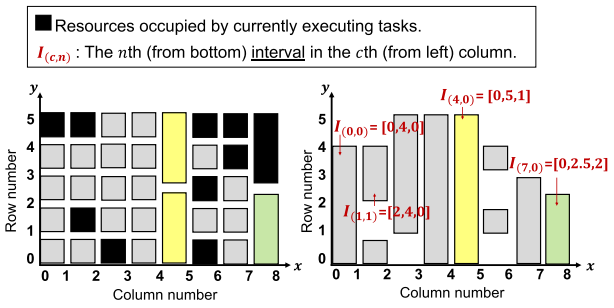
bottom-leftmost corner coordinate of $s_{(0,0)}$ is always marked as (0,0). According to the relative position with the baseline block $s_{(0,0)}$, the information of other continuous blocks in the coordinate axis can be decided. Thus, the continuous block list S for the multi-shape task T can be represented as $S = \{s_{(0,0)}, s_{(1,0)}, s_{(2,0)}, s_{(2,1)}, s_{(3,0)}\}$, shown in Fig. 5.

Similar to previous studies [10]–[17], the heterogeneous multi-shape task is able to be placed at arbitrary locations on the DPR device as long as there are enough unoccupied programmable resources to be assigned. And it is assumed that there are no dependency constraints between different tasks.

C. INTERVAL LIST SET

In order to manage unoccupied programmable resources on the 2D heterogeneous DPR device, an interval list set $List = \{IN_c | c = 0, 1, \ldots, W-1\}$ [23] is applied in this work, where IN_c is the interval list in the cth device column. The definition of an interval $I_{(c,n)} \in IN_c$ is given as follows:

Definition 2: For an interval $I_{(c,n)} = [Ib, It, Ik]$, c is the column number starting from 0 on the leftmost position, and n is the continuous unoccupied rectangles stack in the cth column starting from 0 on the bottom. Ib and It are the bottom and top values of $I_{(c,n)}$ on the y-axis, respectively. Ik is the type of the programmable resource in the cth column, same as Definition 1: $Ik = 0$ for CLB; $Ik = 1$ for BRAM; $Ik = 2$ for DSP.



(a) 2D heterogenous DPR device $F(8,5)$. (b) Interval list set.

FIGURE 6. Example of interval list set on a 2D heterogeneous DPR device $F(8, 5)$. (Interval $I_{(c,n)} = [Ib, It, Ik] \rightarrow$ *Ib*: bottom; *It*: top; *Ik*: type, as defined in Definition 2).

As shown in Fig. 6, the black-colored areas in the DPR device $F(8, 5)$ are resources occupied by currently executing tasks, and other areas are unoccupied resources, in which a targeted task can be placed for execution. Information on intervals for $F(8, 5)$ in each column is shown in Table 1. By updating the interval list in each column, unoccupied programmable resources on the 2D heterogeneous DPR device is managed at run-time.

Two issues that need to be raised are how to quickly find a location where a targeted multi-shape task can be executed based on the interval list set, and how to update the interval list set once a task is assigned or removed. Next section addresses these issues in a way that guarantees the interval list set can be efficiently used to solve the task placement and resource management problems.

TABLE 1. Intervals in each column.

Interval list	Interval $I_{(c,n)} = [Ib, It, Ik]$	
IN_0	$I_{(0,0)} = [0, 4, 0]$	
IN_1	$I_{(1,0)} = [0, 1, 0]$	$I_{(1,1)} = [2, 4, 0]$
IN_2	$I_{(2,0)} = [1, 5, 0]$	
IN_3	$I_{(3,0)} = [0, 5, 0]$	
IN_4	$I_{(4,0)} = [0, 5, 1]$	
IN_5	$I_{(5,0)} = [1, 2, 0]$	$I_{(5,1)} = [3, 4, 0]$
IN_6	$I_{(6,0)} = [0, 3, 0]$	
IN_7	$I_{(7,0)} = [0, 2.5, 2]$	

IV. PROPOSED APPROACH

In this section, we will introduce our proposed approach for task placement and resource management. An overview of the proposed DPR control system is shown in Fig. 7, which consists of three main stages: scheduling, placement, and resource management. The scheduling stage is responsible for deciding which task is selected as the next targeted task to be placed. Our research focuses on the placement and resource management stages and is responsible for selecting a proper execution position for the scheduled targeted task.

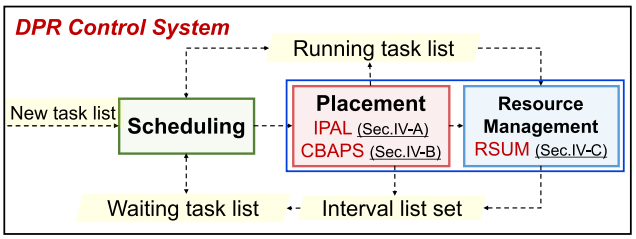


FIGURE 7. Overview of the proposed DPR control system.

In the scheduling stage, the DPR control system selects a task closest to its deadline from a new task list and a waiting task list, as the targeted task to execute. Then, the system searches for all available locations for the targeted task based on the interval list set by the proposed method: IPAL (Section IV-A). If there are not enough programmable resources to execute the targeted task, add the task into the waiting task list for the next scheduling. If several locations can accommodate the target task, select one based on the proposed CBAPS strategy (Section IV-B) and add the task into a running task list. If a task on the DPR device ends execution, it will be removed from the running task list and the DPR device. If a task cannot be scheduled until its deadline, it will be rejected by the system. Note that, once a task is assigned or removed on the DPR device, the interval list set has to be updated for resource management (Section IV-C).

The main variables used to describe the proposed algorithms are summarized in Table 2.

A. INTERVAL-BASED PLACEMENT ALGORITHM (IPAL)

In this section, the proposed interval-based placement algorithm (IPAL) is introduced in detail. The IPAL aims to

TABLE 2. Variable definitions.

Parameter	Definition
$F(W, H)$	A 2D heterogeneous DPR device with W columns and H rows programmable resources.
$T(a, e, d, S)$	A targeted multi-shape task T with arrival time a, execution time e, deadline d, and continuous block list S that will be assigned or removed.
$S = \{s_{(c,n)} \mid c = 0, 1, \ldots$ and $n = 0, 1, \ldots\}$	A continuous block list S, where $s_{(c,n)}$ is the nth (from bottom) continuous block in the cth (from left) column.
$s_{(c,n)} = [sb, st, sk]$	A continuous block $s_{(c,n)} \in S$, where sb and st are the bottom and top value on the y-axis, st is the required resource type.
$List = \{IN_c \mid c = 0, 1, \ldots, W-1\}$	An interval list set $List$, where IN_c is the interval list in column c.
$I_{(c,n)} = [Ib, It, Ik]$	The nth (from bottom) interval in the cth (from left) column and $I_{(c,n)} \in IN_c$, where Ib and It are the bottom and top value on the y-axis, It is the resource type.
$R_{(c,n)} = \{r_i \mid \lambda, i = 0, 1, 2, \ldots\}$	An available range set for a continuous block $s_{(c,n)}$, where $r_i \in R_{(c,n)}$ is one available range and λ is the increment for each available range.
$r_i = [y_{min}, y_{max}]$	An available range $r_i \in R_{(c,n)}$, where y_{min} and y_{max} are the minimal and maximal available y-axis value for a continuous block $s_{(c,n)}$, respectively.

find available locations based on the interval list set, where a targeted multi-shape task T can be accommodated.

The IPAL consists of two main steps: 1) Search and Update of Available Ranges (SUAR) described in Section IV-A1; and 2) Intersection of Updated Available Ranges (IUAR) described in Section IV-A2. The pseudo-code of the proposed IPAL is shown in Algorithm 1.

The search process starts from the zeroth column of the DPR device and scans from left to the right (line 2). For a device column i, firstly find and update available range set $R_{(c,n)}$ (defined in Definition 3 below) for each continuous block $s_{(c,n)}$ when the baseline block $s_{(0,0)}$ of the task T is placed in ith column (line 3-11). Once a continuous block $s_{(c,n)}$ cannot find any available ranges in its corresponding interval list $IN_{(c+i)}$, the search process is stopped (line 7-10) and proceeds to the next cycle (line 2).

Next, intersection results of all updated available ranges for each continuous block are calculated (line 13), which are the locations that $s_{(0,0)}$ can be placed, and guarantee all continuous blocks can be accommodated. Once the location of a baseline block $s_{(0,0)}$ is decided, the assignment information of the entire task can be determined since the relative position between baseline block and other continuous blocks are fixed. Thus, the intersection results are added into the final available range set AR for the targeted multi-shape task T.

At last, if all interval lists from column 0 to (W-1) are finished to scan, all available locations for the multi-shape task T on the 2D heterogeneous DPR device can be obtained. Next, we will detail the process to obtain updated available ranges and calculate intersection results.

Algorithm 1 Interval-Based Placement Algorithm (IPAL)

Require:
2D heterogeneous DPR device: $F(W, H)$;
Interval list set: $List = \{IN_c \mid c = 0, 1, \ldots, W-1\}$;
Targeted multi-shape task: $T\ (a, e, d, S)$;
Continuous block: $s_{(c,n)} \in S$;
Available range set $R_{(c,n)}$ for $s_{(c,n)}$: $R_{(c,n)} = \{r_i \mid \lambda, i = 0, 1, 2, \ldots\}$;
Final available range set for task T: $AR = \{r_i \mid \lambda, i = 0, 1, 2, \ldots\}$.

Ensure:
Obtain final available range set AR for the target task T.

1: $AR \leftarrow \varnothing$;
2: **for** $i \leftarrow 0$ to (W-1) /*Baseline block $s_{(0,0)}$ is placed in ith column.*/ **do**
3: Flag = 0;
4: **for** each $s_{(c,n)}$ in S **do**
5: $R_{(c,n)} \leftarrow \varnothing$;
6: $R_{(c,n)} =$ **SUAR**($s_{(c,n)}$, IN_{c+i}); /*Section IV-A1*/
7: **if** $R_{(c,n)} = \varnothing$ **then**
8: Flag = 1;
9: **break**;
10: **end if**
11: **end for**
12: **if** Flag $\neq$ 1 /*All continuous blocks can find available ranges.*/ **then**
13: $R_{(0,0)} =$ **IUAR**(S); /*Section IV-A2*/
14: add $R_{(0,0)}$ into AR;
15: **end if**
16: **end for**

1) SEARCH AND UPDATE OF AVAILABLE RANGES (SUAR)

In the first step, the proposed IPAL takes a continuous block as the minimum search unit and compares it with intervals of corresponding column to find *available ranges* for the continuous block. *Available* means that not only the resource type of the interval is the same with the continuous block, but also the size of it is enough to accommodate the continuous block.

In Fig. 8, we assume that the resource type of intervals $I_{(i,0)}$ and $I_{(i,1)}$ in the ith column is the same as a continuous block $s_{(c,n)}$. As shown in Fig. 8, interval $I_{(i,1)}$ can accommodate the continuous block $s_{(c,n)}$, while $I_{(i,0)}$ cannot. Thus, in the ith column, only $I_{(i,1)}$ is available for $s_{(c,n)}$.

According to the size of $I_{(i,1)}$ and $s_{(c,n)}$, the available range r_0 for $s_{(c,n)}$ in $I_{(i,1)}$ is $[y_1, y_2]$, which means that the bottom of $s_{(c,n)}$ can be placed between y_1 and y_2 in $I_{(i,1)}$. When all intervals that can accommodate the continuous block are checked in one column, the available range set $R_{(c,n)}$ is obtained, which is defined as follows.

Definition 3: An available range set $R_{(c,n)} = \{r_i \mid \lambda, i = 0, 1, 2, \ldots\}$ is the set of available ranges for a continuous block $s_{(c,n)}$ in the ith device column, where $r_i = [y_{min}, y_{max}]$ is the available range in the ith interval. The increment for

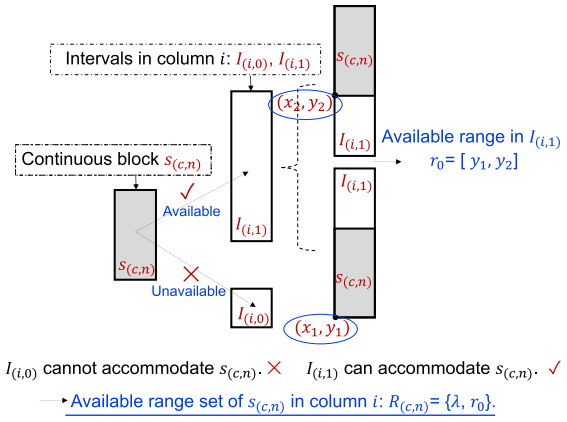


FIGURE 8. Available range set for a continuous block $s_{(c,n)}$ in ith column.

available ranges $\lambda = 1$ for CLB, $\lambda = 2.5$ for BRAM and DSP. For an available range r_i, not all locations between y_{min} and y_{max} are available for the targeted task T. The y-axis value y' of an available location has to satisfy the following equation:

$$y' = y_{min} + k \times \lambda, \tag{1}$$

where $k = 0, 1, 2, \ldots$ and $y' \leq y_{max}$.

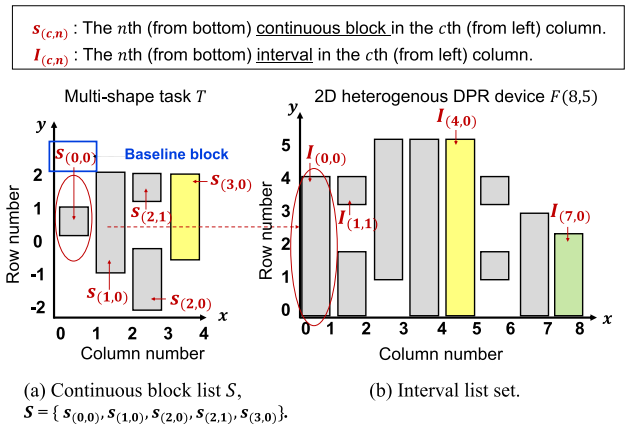


FIGURE 9. Place a multi-shape task T on the DPR device $F(8, 5)$ in Fig. 6.

Fig. 9 shows an example to place a multi-shape task T on the 2D heterogeneous DPR device $F(8, 5)$ in Fig. 6. The search and update of available ranges process is explained with the pseudo-code shown in Algorithm 2. When we search available locations for the multi-shape task T from the zeroth column of the interval list set, the baseline block $s_{(0,0)}$ is firstly compared to the intervals in the zeroth column. Through comparing the size of $I_{(0,0)}$ and $s_{(0,0)}$ (line 6 in Algorithm 2), $s_{(0,0)}$ can be accommodated by $I_{(0,0)}$. Based on the pseudo-code shown in lines 7-14 in Algorithm 2, the new available range is [0, 3] and increment λ is 1, which means that the bottom of $s_{(0,0)}$ can be placed at locations with available y-axis value: 0, 1, 2, 3 in the zeroth column. On the other hand, based on the relative position between the baseline block $s_{(0,0)}$ and $s_{(1,0)}$, the search process of available

Algorithm 2 Search and Update of Available Ranges (SUAR)

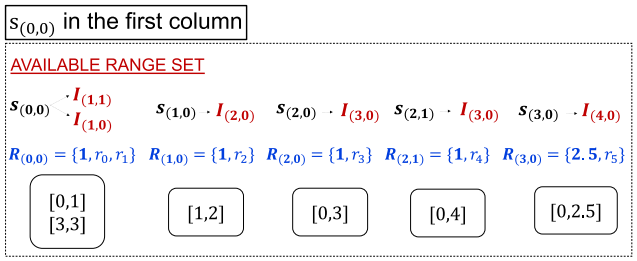
Require:
Interval list in ith column: IN_i;
Interval in the interval list IN_i: $I_{(c,n)} = [Ib, It, Ik]$;
Continuous block: $s_{(c,n)} = [sb, st, sk]$;
Increment: λ.

Ensure:
Obtain available range set $R_{(c,n)}$ for $s_{(c,n)}$ in ith column.

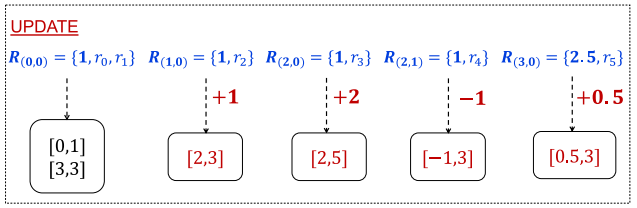
1: **if** $IN_i \neq \varnothing$ **then**
2: **if** $sk \neq Ik$ /*Continuous block and interval are different resource types.*/ **then**
3: $R_{(c,n)} \leftarrow \varnothing$;
4: **else**
5: **for** each interval $I_{(c,n)} = [Ib, It, Ik]$ in IN_i **do**
6: **if** $sb - st \leq It - Ib$ **then**
7: **if** $sk = 1$ **then**
8: /*Type of $s_{(c,n)}$ is CLB.*/
9: $\lambda \leftarrow 1$;
10: **else**
11: /*Type of $s_{(c,n)}$ is BRAM or DSP.*/
12: $\lambda \leftarrow 2.5$;
13: **end if**
14: generate $r_{new} = [Ib, It - (st - sb)]$;
15: update $r_{new} = [Ib - sb, It - (st - sb) - sb]$;
16: add r_{new} and λ into $R_{(c,n)}$;
17: **end if**
18: **end for**
19: **end if**
20: **end if**

ranges for $s_{(1,0)}$ should start from the column 1, where both $I_{(1,0)}$ and $I_{(1,1)}$ are unavailable. As mentioned in lines 7-10 of Algorithm 1, once one continuous block cannot find any available ranges in the corresponding column, the search process returns to the baseline block $s_{(0,0)}$ and starts the next cycle. In this example, we return the search process to the baseline block $s_{(0,0)}$ regarding that the $s_{(0,0)}$ is placed in column 1. It is likely that both interval $I_{(1,0)}$ and $I_{(1,1)}$ in the first column can accommodate $s_{(0,0)}$. Thus, the available range r_0 for $I_{(1,0)}$ is [0, 1] and r_1 for $I_{(1,1)}$ is [3, 3]. The available range set for $s_{(0,0)}$ in the first column is $R_{(0,0)} = \{1, r_0, r_1\}$. Next, $s_{(1,0)}$ and $I_{(2,0)}$, $s_{(2,0)}$ and $I_{(3,0)}$, $s_{(2,1)}$ and $I_{(3,0)}$, $s_{(3,0)}$ and $I_{(4,0)}$ are compared respectively, to get available range set for each continuous block, which is shown in Fig. 10 (a).

It is noteworthy to state that for CLBs, the increment of available ranges is 1 (lines 7-9 in Algorithm 2). However, for BRAMs or DSPs, the increment of available ranges is 2.5 due to the different occupied space modeled in Fig. 4 (lines 10-12 in Algorithm 2). Thus, the available range r_4 for $s_{(2,1)}$ in Fig. 10 (a) is [0, 4], which means that the bottom of $s_{(2,1)}$ can be placed in $y' = 0, 1, 2, 3, 4$ in $I_{(3,0)}$ since the resource type of $s_{(2,1)}$ is CLB. In contrast, the resource type of $s_{(3,0)}$ is BRAM; thus, the available range r_5 is [0, 2.5], which



(a) Search for available range sets.



(b) Update available range sets.

FIGURE 10. Result of search and update of available range sets when continuous block $s_{(0,0)}$ is placed in the first column.

means that the bottom of $s_{(3,0)}$ can be placed in $y' = 0, 2.5$ in $I_{(4,0)}$.

Before proceeding to the next step, all available range sets need to be updated since the relative position relationships between $s_{(0,0)}$ and other continuous blocks have not been considered yet. In Fig. 9, the $s_{(0,0)}$ is regarded to be a baseline block and the bottom of $s_{(1,0)}$ is one CLB lower than that of $s_{(0,0)}$. Therefore, all available ranges in $R_{(1,0)}$ for $s_{(1,0)}$ have to plus 1 to make the bottom of $s_{(1,0)}$ and the baseline block $s_{(0,0)}$ on the same y-axis level (line 15 in Algorithm 2). In contrast, the bottom of $s_{(2,1)}$ is one CLB higher than that of the baseline block $s_{(0,0)}$ so that all available ranges in $R_{(2,1)}$ for $s_{(2,1)}$ have to minus 1 (line 15 in Algorithm 2). According to the relative position with the baseline block, all available ranges for each continuous block have to be updated, which are shown in Fig. 10 (b).

2) INTERSECTION OF UPDATED AVAILABLE RANGES (IUAR)

In the previous step, available ranges for each continuous block have been searched and updated. The final placement location must ensure that it is available for each continuous block. Therefore, we try to obtain the *intersection range* of all updated available ranges. However, the difference in the increment of the available range increases the difficulty of calculating the intersection range. Next, we will summarize methods for calculating the intersection range in different situations.

The pseudo-code for the intersection of updated available ranges $r_i = [a, b]$ and $r_j = [c, d]$ are shown in Algorithm 3. For r_i and r_j, it is evident that if the maximum amount of one updated available range is smaller than the minimal amount

Algorithm 3 Intersection of Two Updated Available Ranges (IUAR)

Require:
Updated available ranges: $r_i = [a, b]$, $r_j = [c, d]$;
Increments of r_i and r_j: λ_1, λ_2;
Increment: $\lambda = 0.0$;

Ensure:
For r_i and r_j, obtain intersection range $r' = [m, n]$ and its increment λ'.

1: **if** $a > d$ or $b < c$ **then**
2: r' is empty;
3: **else**
4: **if** $\lambda_1 = \lambda_2$ /*Increments of r_i and r_j are the same.*/ **then**
5: $\lambda \leftarrow \lambda_1$
6: **if** $\{\mathbf{Max}(a, c) - \mathbf{Min}(a, c)\}$ **Mod** $\lambda = 0$ **then**
7: $m \leftarrow$ **Max** (a, c);
8: $n \leftarrow$ **Min** (b, d);
9: $\lambda' \leftarrow \lambda$;
10: **else**
11: r' is empty;
12: **end if**
13: **else if** $\lambda_1 \neq \lambda_2$ /*Increments of r_i and r_j are different.*/ **then**
14: $\lambda \leftarrow$ **getIncrt**$(a, c, \lambda_1, \lambda_2)$;
15: **if** $\{\mathbf{Max}(a, c) - \mathbf{Min}(a, c)\}$ **Mod** λ z 0 **then**
16: $m \leftarrow$ **Max** (a, c);
17: $n \leftarrow$ **Min** (b, d);
18: $\lambda' \leftarrow 5$;
19: **if** m **Mod** $1.0 \neq 0$ and $m + 2.5 \leq n$ /*m is not an integer multiple of 1.*/ **then**
20: $m \leftarrow m + 2.5$;
21: **else if** m **Mod** $1.0 \neq 0$ and $m + 2.5 > n$ **then**
22: r' is empty;
23: **end if**
24: **else**
25: $m \leftarrow \mathbf{Max}(a, c) + \lambda - \{(\mathbf{Max}(a, c) - \mathbf{Min}(a, c))$ **Mod** $\lambda\}$
26: $n \leftarrow$ **Min** (b, d);
27: $\lambda' \leftarrow 5$;
28: **if** $m > n$ **then**
29: r' is empty;
30: **else**
31: **if** m **Mod** $1.0 \neq 0$ and $m + 2.5 \leq n$ /*m is not an integer multiple of 1.*/ **then**
32: $m \leftarrow m + 2.5$;
33: **else if** m **Mod** $1.0 \neq 0$ and $m + 2.5 > n$ **then**
34: r' is empty;
35: **end if**
36: **end if**
37: **end if**
38: **end if**
39: **end if**

of another updated available range, it is impossible for r_i and r_j to intersect so that the intersection range r' is empty (lines 1−2). Otherwise, two situations need to be considered in Algorithm 3 as follows:

- Increments of r_i and r_j: λ_1 and λ_2 are the same (lines 4−12).
  Lines 6−12: Intersection range r' exists when the difference between a in r_i and c in r_j is an integer multiple of its increment λ. Intersection range r' is shown in lines 7−9, where function **Max** and **Min** are the maximum and minimum value between two inputs.
- Increments of r_i and r_j: λ_1 and λ_2 are different (lines 13−38).
  Note that, all possible increments of the available range is 1, 2.5, 5. Therefore, the intersection of r_i and r_j must be integers, in the case that r_i and r_j have different increments.
  Line 14: The function **getIncrt** is used to get the increment of the updated available range which has the smaller y_{min} and store the increment into λ.
  Lines 15−23: If the difference between a in r_i and c in r_j is an integer multiple of the increment λ, the intersection range r' is shown in lines 16−18. The increment λ' is set to 5 because the intersection must be an integer and be a multiple of 2.5 (line 18). On the other hand, m may be a decimal, but the intersection must be an integer so that m should plus 2.5 to become an integer (lines 19−20). Note that, the intersection range r' is nonexistent when $(m + 2.5)$ is larger than n (lines 21−23).
  Lines 24−35: If the difference between a in r_i and c in r_j is not an integer multiple of the increment λ, the intersection range r' and its increment λ' are shown in lines 25−27, where the value of m is quite different from previously mentioned. That is because the remainder should be considered when (**Max**(a, c)−**Min**(a, c)) is not an integer multiple of the increment λ (line 25). Furthermore, m is an integer or not that is considered as lines 19−20, which are shown in lines 31−35.

Fig. 11 shows an example of the intersection process, where the updated available ranges of each continuous block are compared and the final intersection range is [3, 3], which means that the available range for the multi-shape task T is [3, 3] and the increment is 5. Further, the baseline block $s_{(0,0)}$ can be placed at (3, 3) in the first column. From this step, the placement location of multi-shape task T when $s_{(0,0)}$ is placed in the first column is found.

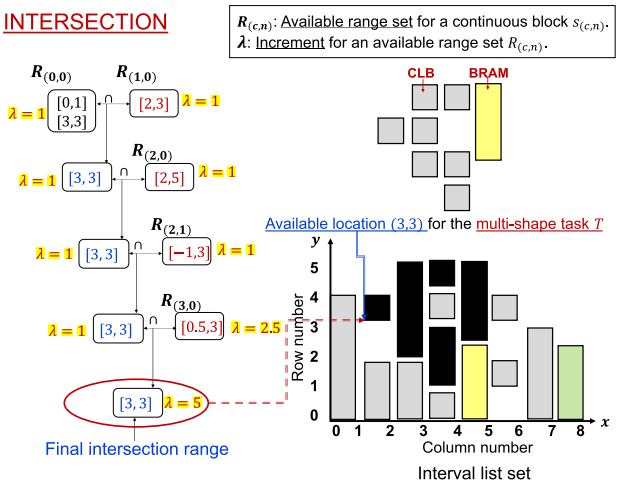


FIGURE 11. Intersection range of updated available ranges.

B. CONTINUOUS BLOCK AWARE PLACEMENT STRATEGY (CBAPS)

All information of available programmable resources on the 2D heterogeneous DPR device in the current status is stored in the interval list set. A targeted multi-shape task can find all available locations by searching the interval list set through applying IPAL mentioned in Section IV-A. If the targeted task can be placed in several locations, it is necessary to use a proper approach as a selection strategy to choose the best one. In this section, a continuous block aware placement strategy (CBAPS) is introduced to be combined with the proposed IPAL.

To select the best available location to place a multi-shape task T, we evaluate the *Continuity* $C(x, y)$ which can avoid increasing the degree of fragmentation of programmable resources on the DPR device as much as possible. The more fragmented available programmable resources are, the more difficult it is to place subsequent tasks.

Continuity is defined as follows:

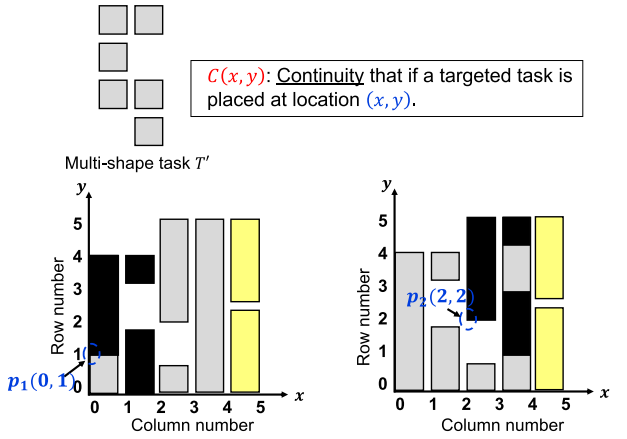
$$C(x, y) = \frac{n}{\sum_{i=0}^{N} Iw_i \times Ih_i}, \tag{2}$$

where n is the number of intervals generated by placing the targeted task in the coordinate of (x, y). N is the number of intervals that the target task occupied, Iw and Ih is the width and height of the occupied interval. A smaller value of $C(x, y)$ means that fewer intervals are generated by the per unit area of the selected intervals after the task assigned. Thus, the available location with the smaller $C(x, y)$ is selected to place the target task in this CBAPS strategy.

Fig. 12 shows an example of how to calculate the $C(x, y)$ in different locations. According to the information of intervals, the targeted task with black color has two available locations: $p_1(0, 1)$ and $p_2(2, 2)$, to place on the DPR device $F(5, 5)$. Based on the 2, the value of $C(0, 1)$ is 0.14 and larger than the value of $C(2, 2) = 0.25$. Therefore, we select the location p_1 to place the targeted task.

C. RESOURCE STATUS UPDATE METHOD (RSUM)

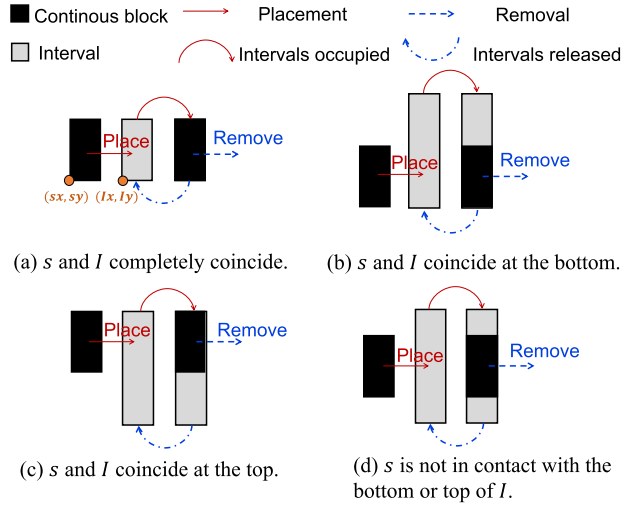
In the placement stage, the proposed interval list set is used to record the unoccupied 2D heterogeneous DPR resources. Once a task is placed or removed on the device, the interval list set has to be updated. Furthermore, placement and removal are two reverse processes, task placement is the



(a) T' is placed at location $p_1(0,1)$. $C(0,1) = \frac{1}{4+2+1} = 0.14$

(b) T' is placed at location $p_2(0,1)$. $C(2,2) = \frac{2}{3+5} = 0.25$

FIGURE 12. Example of CBAPS.



(a) s and I completely coincide.

(b) s and I coincide at the bottom.

(c) s and I coincide at the top.

(d) s is not in contact with the bottom or top of I.

FIGURE 13. Update of intervals for task placement and removal.

process that occupied available programmable resources and removal is the process to release the occupied ones.

As shown in Fig. 13, there are four cases based on the size and relative position of the continuous block $s = [sb, st, sk]$ and its occupied interval $I = [Ib, It, Ik]$. We assume that the placement location of s is (sx, sy) and the location of I is (Ix, Iy). Four cases can be summarized as follows:

1) If $s = I$:
   Placement: delete I,
   Removal: generate $I_{new} = [sb, st, Ik]$;
2) If $s \neq I$ and $sb = Ib$:
   Placement: $Ib \leftarrow st$, **Removal**: $Ib \leftarrow sb$;
3) If $s \neq I$ and $st = It$:
   Placement: $It \leftarrow sb$, **Removal**: $It \leftarrow st$;
4) If $s \neq I$, $st \neq It$ and $sb \neq Ib$:
   Placement: $Ib = st$ and generate $I_{new} = [Ib, sb, Ik]$,
   Removal: combine two intervals into one interval.

Based on different cases, all intervals can be updated when a task is placed or removed on the 2D heterogeneous DPR device.

V. THEORETICAL ANALYSIS

In this section, we will evaluate the proposed approach through theoretical analysis, which includes analysis of time and space complexity.

A. TIME COMPLEXITY

We assume that there is a multi-shape task T with w columns that will be placed on a 2D heterogeneous DPR device $F(W, H)$. There are n continuous blocks in each task column and N intervals in each DPR device column.

For the IPAL, firstly, each continuous block needs to be compared with intervals in its corresponding column to search and update the available ranges, the cost is (nwN). For each continuous block, the maximal number of available ranges in one column is N, which means that each interval in this column is available. In the intersection process, available ranges for two continuous blocks are compared to obtain the final available locations. Therefore, the intersection process costs $((nw-1)N)$. In total, the cost of finding all available locations for the multi-shape task T is $((nwN + (nw-1)N) \times W)$. Therefore, the total time complexity for IPAL is $O(nwNW)$.

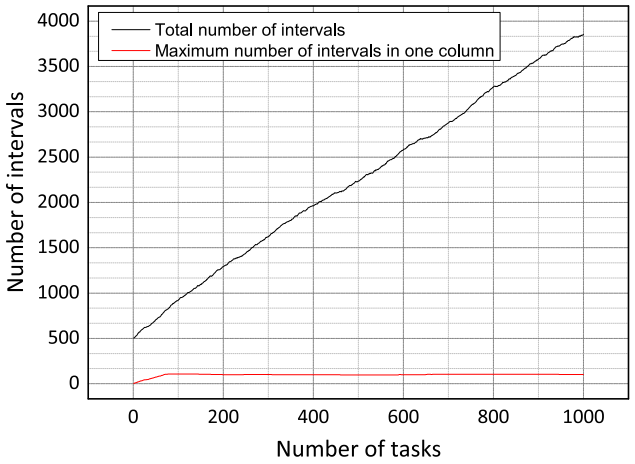


FIGURE 14. Number of intervals when 1000 tasks are placed in a DPR device $F(500, 500)$.

We experiment to explore the number of intervals when 1000 tasks will be placed on the DPR device $F(500, 500)$. Fig. 14 shows the trend of the number of intervals during task placement, where the red line represents the maximum number of intervals N in one column, and the black line is the total number of intervals. Fig. 14 demonstrates that N tends to stabilize as the number of tasks increases and is much smaller than the height of the DPR device. The reason is that, the maximum number of intervals in one column is theoretically at most $H/2$, based on the definition of the interval, where H is the total number of programmable resources

in one column.Thus, after the initial increase, the maximum number of intervals in one column tends to be flat.

B. SPACE COMPLEXITY

Space complexity of the proposed algorithm consists of the following parts: 1) interval list set *List*, 2) available range set $R_{(c,n)}$ for each continuous block. Thus, the total space complexity is $O((W + nw)N)$.

VI. SIMULATION AND DISCUSSION

In this section, we do simulations to evaluate the performance of the proposed approach.

A. SIMULATION SETUP

To evaluate the performance of the proposed approach, we construct a simulation framework using C language and run on macOS 10.15.1, GCC 4.8 on 1.4 GHz Quad-Core Intel Core i5 Processor with 8 GB Memory. Comparison algorithms are given as follows:

- **QC [10]**: Quad-corner strategy based on the 2D DPR device matrix.
- **EUAS [11]**: Expanding un-used area strategy based on the 2D DPR device matrix.
- **BFT [12]**: Adjacency value strategy based on the 2D DPR device matrix.
- **Interval-FF**: Proposed IPAL+RSUM combined with first-fit (FF) strategy based on interval list set.
- **Interval-FRAG**: Proposed IPAL+RSUM combined with proposed CBAPS strategy based on interval list set.

Experiments are performed on the 2D heterogeneous DPR device with 100×100 programmable resources. According to the task size, four task sets with randomly generated parameters as input data are shown in Table 3. Each task set consists of 1000 tasks, where w_r is the range of task width, h_r is the range of top value on the y-axis for a continuous block, e_r represents the range of task execution time.

TABLE 3. Task sets for simulation.

Task set	w_r	h_r	$e_r[ms]$	Number of tasks
TS1 (Small)	[1-5]	[1-5]	[0...500]	1000
TS2 (Middle)	[3-8]	[3-8]		
TS3 (Large)	[5-10]	[5-10]		
TS4 (Mixed)	[1-10]	[1-10]		

B. SIMULATION RESULTS

Three indicators: average runtime, fragmentation degree, and rejection ratio, are proposed to evaluate the performance of the proposed approach. We will introduce these indicators and show simulation results in detail.

1) EVALUATION OF RUNTIME

Average runtime is measured to evaluate the runtime overhead of comparison algorithms. Runtime overhead is the sum of time for **search** for available locations, **select** the proper placement location and **resource management**.

Table 4 shows the simulation results of the average runtime. For each algorithm, the ③ resource management process takes almost negligible time compared to ① search and ② selection processes. Therefore, we focus on and analyze the first two processes in detail.

For all task sets, the QC has the least runtime overhead in search and selection process. That is because the limited searching candidates reduce the exploring range of QC. Furthermore, the QC essentially uses the FF strategy, that is once an available location is found, stop searching and place the targeted task.

Both EUAS and BFT are based on using a resource matrix to find all available locations. Therefore, in the search process, EUAS and BFT almost use the same runtime overhead to traverse the entire resource matrix. As shown in Table 4, our proposed Interval-FF and Interval-FRAG have better runtime overhead in search process than that of BFT and EUAS. The reason is that the proposed IPAL in Interval-FF and Interval-FRAG considers several continuous programmable resources as one searching unit to find available ranges, which has a higher efficiency than traversing the entire DPR device matrix.

For the Interval-FRAG, the proposed CBAPS selection strategy has to search for all available locations on the DPR device to select the best one so additional time is necessary compared to the FF selection strategy in Interval-FF and QC. For the CBAPS, the process of calculating $C(x, y)$ is more complicated than the selection strategy of EUAS and BFT, so it takes more time in the selection process.

2) EVALUATION OF FRAGMENTATION

Fragmentation is an indicator related to the placement quality. The degree of fragmentation negatively correlates with the continuity of available programmable resources on the device. Otherwise, the more dispersed the available programmable resources, the less likely subsequent tasks are assigned on the DPR device. In this work, the fragmentation degree F is calculated by the following equation [24]:

$$F = \begin{cases} \dfrac{\tau - 1}{\chi - 1} \times 100, & for\ \chi > 1; \\ 0, & for\ \chi = 1, \end{cases} \tag{3}$$

where χ is the number of free programmable resources; τ is the quantity of connected programmable resource set, in which any two programmable resources are reachable to each other. A smaller F value means much more continuity of available programmable resources.

Fig. 15 shows the results of fragmentation for different algorithms when TS4 is assigned on the $F(100, 100)$ DPR device. The x-axis represents the total number of tasks on the device in the current status. With the increase in the number of tasks on the DPR device, the fragmentation degree F is increased due to the reduction in the number of free programmable resources χ and the increase in the number of connected programmable resource set τ.

TABLE 4. Evaluation of average runtime.

Algorithm	Process	Average runtime (μs): ① search / ② selection / ③ resource management							
		TS1[1-5]	Ratio	TS2[3-8]	Ratio	TS3[5-10]	Ratio	TS4[1-10]	Ratio
QC [10]	**Total**	**7**	0.06	**8**	0.10	**6**	0.07	**8**	0.08
	①+②+③	(7/<1/<1)		(8/< 1/<1)		(6/<1/<1)		(7/<1/1)	
EUAS [11]		**113**	0.90	**63**	0.76	**66**	0.77	**81**	0.78
		(109/4/<1)		(61/2/<1)		(64/2/<1)		(77/3/1)	
BFT [12]		**151**	1.20	**88**	1.06	**95**	1.10	**119**	1.14
		(109/42/<1)		(60/28/<1)		(61/34/<1)		(77/41/1)	
Interval[1]-FF		**15**	0.12	**17**	0.20	**17**	0.20	**23**	0.22
		(15/<1/<1)		(17/<1/< 1)		(16/<1/1)		(22/<1/1)	
Interval[1]-FRAG[2]		**126**	1.00	**83**	1.00	**86**	1.00	**104**	1.00
		(38/88/<1)		(25/58/<1)		(21/64/1)		(32/71/1)	

[1] Our proposed IPAL + RSUM.
[2] Our proposed selection strategy CBAPS.

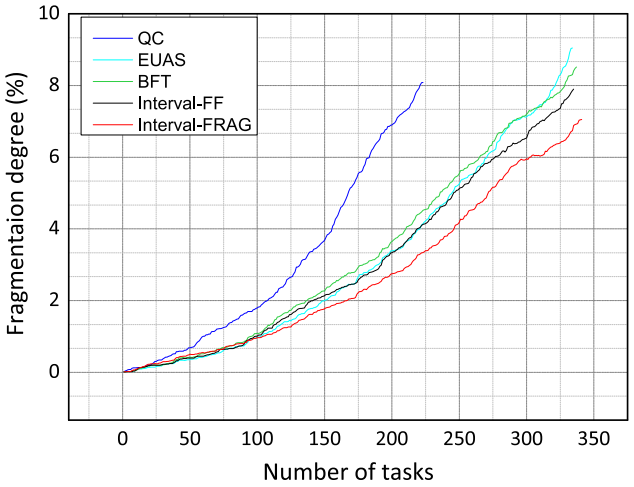


FIGURE 15. Evaluation of fragmentation.

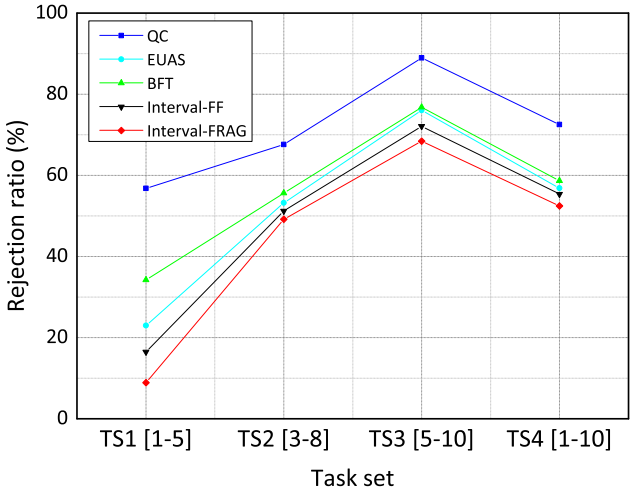


FIGURE 16. Evaluation of placement quality for different task sizes.

As shown in Fig. 15, the proposed Interval-FRAG (red line) has the smallest fragmentation degree. That is because the proposed CBAPS select the intervals that generate fewer new intervals to place the new task. Furthermore, although QC has the highest speed, it makes the available resources on the device more fragmented compared to other algorithms.

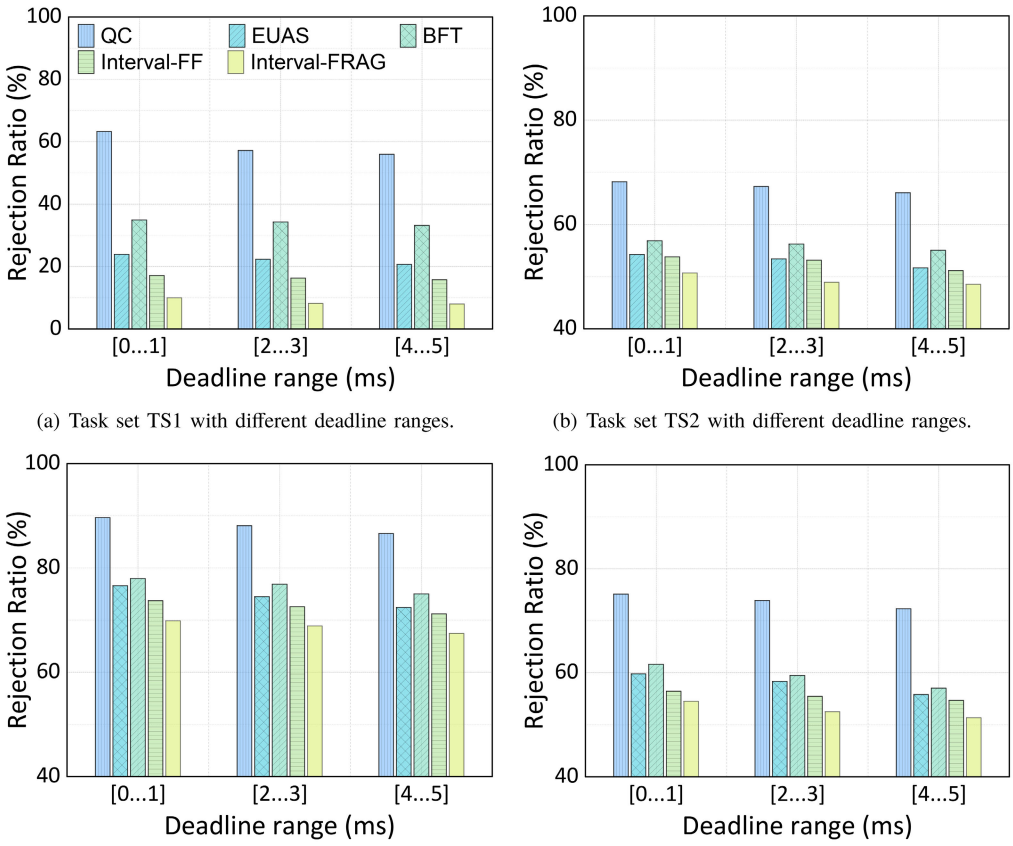
3) EVALUATION OF PLACEMENT QUALITY

If a task cannot be executed before its deadline, it will be rejected. **Rejection ratio** ($\Re$) is proposed to evaluate the task placement quality. The equation to calculate the rejection ratio is shown as follows [16]:

$$\Re = \frac{\sum\limits_{\forall t_i \in REJECT} \vartheta_i \times e_i}{\sum\limits_{\forall t_j \in TOTAL} \vartheta_j \times e_j} \times 100, \tag{4}$$

where ϑ_i is the size of rejected task t_i, ϑ_j is the size of task t_j in total task set, and e represents the execution time.

From the results shown in Fig. 16 and Fig. 17, in different cases, the rejection ratio of QC is the highest. That is because QC searches for available locations quickly by reducing the number of search candidates, which results in the problem that even if there are enough programmable resources assigned to the targeted task, the task still cannot find the available locations. This is an algorithm that sacrifices the quality of placement in exchange for speed. In addition, the rejection ratio of EUAS is lower than that of BFT, since the expanding un-used area strategy in EUAS tries to avoid placements that render certain portions of the chip unusable, which is more efficient than calculating the adjacent value in the BFT. The rejection ratio of Interval-FF is better than that of EUAS although Interval-FF adopts the first-fit strategy. The reason is that the runtime overhead of Interval-FF is much shorter than that of EUAS. Once multiple tasks are waiting to be placed at the same time, the less time-consuming algorithm can find the placement location for the task faster, thereby reducing the waiting time of subsequent tasks. At last, the proposed Interval-FRAG has the best placement quality since the fragmentation degree is much lower than other algorithms. Lower fragmentation means that



(a) Task set TS1 with different deadline ranges.

(b) Task set TS2 with different deadline ranges.

(c) Task set TS3 with different deadline ranges.

(d) Task set TS4 with different deadline ranges.

FIGURE 17. Evaluation of placement quality for different deadline ranges with task set TS1 to TS4.

the programmable resources are more continuous so that the subsequent task is much easier to be placed, which increases the placement quality of the system.

Fig. 16 also shows the relationship between the task size and rejection ratio. The simulation in Fig. 16 is done based on the same task deadline range [0…5] (ms). With the increase of task size, the rejection ratio for Interval-FRAG is increased. That is because the larger task size means the more continuous blocks the task has. Time complexity of IPAL is related to the number of continuous blocks, which has been proven in Section V-A.

Fig. 17 shows the rejection ratio with different task deadline ranges for each task set. For each task set, the rejection ratio is decreased with the deadline range increased from [0…1] to [4…5] (ms), that is because a shorter deadline means that the task has a higher urgency for execution, and can easily miss its deadline.

VII. CONCLUSION AND FUTURE WORK

In this paper, to address the task placement and resource management problems for multi-shape tasks on a 2D heterogeneous DPR device, a continuous block list is proposed to model the multi-shape task in columns, and an interval list set is applied to manage heterogeneous programmable resources. Simulation results show that the resource management method proposed based on the interval list set decreases the time for multi-shape tasks to find all available locations by an average of 62.6% compared to the traditional matrix-based resource management methods (EUAS and BFT). Furthermore, the rejection ratio is decreased by at least 8.9% with an average fragmentation reduction of 18.1%. Therefore, by applying the proposed task placement algorithm and resource management method, the performance of 2D heterogeneous DPR devices has been greatly improved, allowing such devices to meet the growing demand for real-time computing systems in the future.

There are three main stages: scheduling, placement, and resource management in the DPR control system. In this paper, the task dependency and data transfer are ignored in the scheduling stage. As a future work, a scheduling algorithm that considers task size, shape, and task-dependency could further improve the performance of the 2D heterogeneous DPR device.

REFERENCES

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

[2] T. N. Theis and H.-S.-P. Wong, "The end of Moore's law: A new beginning for information technology," *Comput. Sci. Eng.*, vol. 19, no. 2, pp. 41–50, Mar. 2017.
[3] E. P. DeBenedictis, "Moore's law: A hard act to follow," *Computer*, vol. 52, no. 12, pp. 114–117, Dec. 2019.
[4] J. Shalf, "The future of computing beyond Moore's law," *Philos. Trans. Roy. Soc. A*, vol. 378, no. 2166, 2020, Art. no. 20190061.
[5] G. Bello-Orgaz, J. J. Jung, and D. Camacho, "Social big data: Recent achievements and new challenges," *Inf. Fusion*, vol. 28, pp. 45–59, Mar. 2016.
[6] M. Sandrieser, S. Benkner, and S. Pllana, "Explicit platform descriptions for heterogeneous many-core architectures," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 1292–1299.
[7] A. Putnam, "FPGAs in the datacenter: Combining the worlds of hardware and software development," in *Proc. Great Lakes Symp. VLSI*, 2017, p. 5.
[8] B. Wilder, *Cloud Architecture Patterns: Using Microsoft Azure*. Sebastopol, CA, USA: O'Reilly Media, 2012.
[9] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Found. Trends Electron. Design Autom.*, vol. 2, no. 2, pp. 135–253, 2007.
[10] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "A novel fast online placement algorithm on 2D partially reconfigurable devices," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2009, pp. 296–299.
[11] G. Enemali, A. Adetomi, and T. Arslan, "Expanding the un-usable area strategy for improved utilization of reconfigurable FPGAs," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul. 2017, pp. 139–144.
[12] C. Wang, W. Wu, S. Nie, and D. Qian, "BFT: A placement algorithm for non-rectangle task model in reconfigurable computing system," *IET Comput. Digit. Techn.*, vol. 10, no. 3, pp. 128–137, May 2016.
[13] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement," in *Proc. 41st Annu. Conf. Design Autom. (DAC)*, 2004, pp. 960–965.
[14] X. Iturbe, K. Benkrid, T. Arslan, C. Hong, and I. Martinez, "Empty resource compaction algorithms for real-time hardware tasks placement on partially reconfigurable FPGAs subject to fault ocurrence," in *Proc. Int. Conf. Reconfigurable Comput. (FPGAs)*, Nov. 2011, pp. 27–34.
[15] T. Pan, L. Zeng, Y. Takashima, and T. Watanabe, "A fast MER enumeration algorithm for online task placement on reconfigurable FPGAs," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. 99, no. 12, pp. 2412–2424, 2016.
[16] T. Zhou, T. Pan, M. C. Meyer, Y. Dong, and T. Watanabe, "A fast online task placement algorithm for three-dimensional dynamic partial reconfigurable devices," *IEEE Access*, vol. 8, pp. 36903–36918, 2020.
[17] T. Zhou, T. Pan, M. C. Meyer, Y. Dong, and T. Watanabe, "An interval-based mapping algorithm for multi-shape tasks on dynamic partial reconfigurable FPGAs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2020, pp. 127–130.
[18] K. W. Tindell, A. Burns, and A. J. Wellings, "Allocating hard real-time tasks: An NP-hard problem made easy," *Real-Time Syst.*, vol. 4, no. 2, pp. 145–165, Jun. 1992.
[19] M. Esmaeildoust, M. Fazlali, A. Zakerolhosseini, and M. Karimi, "Fragmentation aware placement algorithm for a reconfigurable system," in *Proc. 2nd Int. Conf. Electr. Eng.*, Mar. 2008, pp. 1–5.
[20] B. Przybus, "Xilinx redefines power, performance, and design productivity with three new 28 nm FPGA families: Virtex-7, Kintex-7, and Artix-7 devices," Xilinx, San Jose, CA, USA, White Paper WP373, 2010. [Online]. Available: https://mikrokontroler.pl/wp-content/uploads/pliki/wp373_V7_K7_A7_Devices.pdf
[21] I. Xilinx, *series FPGAs Configurable Logic Block: User Guide*. San Jose, CA, USA: Xilinx, 2014.
[22] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Glasgow, U.K.: Strathclyde Academic Media, 2014.
[23] T. Pan, L. Zhu, L. Zeng, T. Watanabe, and Y. Takashima, "An online task placement algorithm based on MER enumeration for partially reconfigurable device," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. 99, no. 7, pp. 1345–1354, 2016.
[24] M. Handa and R. Vemuri, "Area fragmentation in reconfigurable operating systems," in *Proc. ERSA*, 2004, pp. 77–83.

TINGYU ZHOU (Member, IEEE) was born in Chongqing, China, in April 1995. She received the B.E. degree in software engineering from Sichuan University, in 2016, and the M.E. degree from the Graduate School of Information, Production and Systems, Waseda University, in 2017, where she is currently pursuing the Ph.D. degree. Since 2019, she has been a Research Associate with the IPS Research Center, Waseda University. Her research interests include reconfigurable computing and run-time mapping algorithm.

TIEYUAN PAN was born in Donggang, China, in October 1988. He received the B.E. and M.E. degrees from the University of Kitakyushu, in 2012 and 2014, respectively, and the Ph.D. degree from the Graduate School of Information, Production and Systems, Waseda University, in 2017. From April 2015 to April 2017, he had been a Research Associate with the IPS Research Center, Waseda University. He is currently working with DENSO Corporation. His research interests include combinatorial algorithm for VLSI layout design and its applications.

MICHAEL CONRAD MEYER (Member, IEEE) received the B.S. degree in computer engineering and the M.A. degree in engineering management from the Rose-Hulman Institute of Technology, Terre Haute, IN, USA, in 2012 and 2013, respectively, and the Ph.D. degree in computer science and engineering from The University of Aizu. He worked for Texas Instruments before starting his Ph.D., and before that he had worked for Syntheon developing biomedical devices. He is currently an Assistant Professor with Waseda University. He was previously a Postdoctoral Researcher with The University of Aizu, Japan, and as a member of the Data Networking Laboratory. His research interests include on and off chip networks, reliability, and photonics.

YIPING DONG was born in Jiangsu, China, in 1983. He received the B.E. degree in electronics and engineering from Southeast University, China, in 2006, and the M.S. and Dr.Eng. degrees from the Graduate School of Information, Production and System, Waseda University, Japan, in 2008 and 2011, respectively. He worked in the field of NoC, FPGA, and artificial neural network with Waseda University. He joined the Research and Development Center, HITACHI Corporation, in 2012, where he worked in the field of image processing and FPGA. In August 2014, he joined China Key System & Integrated Circuit Company Ltd., where he worked in the field of design and application of FPGA and NoC design.

TAKAHIRO WATANABE (Life Member, IEEE) was born in Ube, Japan, in October 1950. He received the B.E. and M.E. degrees in electrical engineering from Yamaguchi University, and the Dr.Eng. degree from Tohoku University. In 1979, he joined the Research and Development Center, TOSHIBA Corporation, where he worked in the field of LSI design automation. In August 1990, he joined the Department of Computer Science and Systems Engineering, Yamaguchi University. In April 2003, he moved to the Graduate School of Information, Production and Systems, Waseda University. His current research interests include EDA algorithm, microprocessor and MPSoC, NoC, FPGA, and their applications. He is also a member of IEICE and IPSJ.