FEditor: Consecutive Task Placement With Adjustable Shapes Using FPGA State Frames

Yanyan Li, Yu Chen, Zhiqian Xu, Yawen Wang, Hai Jiang, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Field Programmable Gate Arrays (FPGAs) are widely adopted in datacenters, where each FPGA is exclusively assigned to a task. This strategy results in significant resource waste and increased task rejections. To address this issue, placement algorithms adjust the locations and shapes of tasks based on Dynamic Partial Reconfiguration, which partitions an FPGA into multiple rectangular areas for sharing. However, existing schemes are designed for static task sets without adjustable shapes, incapable of optimizing the placement problem in datacenters. In this paper, FEditor is proposed as the first consecutive task placement scheme with adjustable shapes. It expands the planar FPGA models into three-dimensional ones with timestamps to accommodate consecutive tasks. To reduce the complexity of three-dimensional resource management, *State Frames* (*SFs*) are designed to compress the models losslessly. Three metrics and a nested heuristic algorithm are used for task placement. Experimental results demonstrate that FEditor has improved resource utilization by at least 19.8% and acceptance rate by at least 10% compared to the referenced algorithms. *SFs* and the nested algorithm accelerate the task placement by up to 10.26×. The suitability of FEditor in datacenter environments is verified by its time efficiency trends.**

***Index Terms*—Adjustable shapes, consecutive tasks, FPGA, placement algorithms.**

I. Introduction

FIELD Programmable Gate Arrays (FPGAs) are flexible, energy-efficient, and high-performance accelerators based on the Non-Von Neumann architecture in datacenters [1], [2], [3]. A task is processed on an FPGA by loading its bitstream [9], [10] rather than being encoded into instructions [19]. FPGAs can be reconfigured via bitstreams for different functionalities [10] and such flexibility can help improve resource sharing. As tasks on FPGAs have their own dedicated memory and computing resources, context switching, inter-process communication, and memory access arbitration can all be streamlined. This enables FPGAs to consume only the necessary energy and time for task executions [6]. Thus, FPGAs have been widely adopted [27], [31], [34], [38], [41].

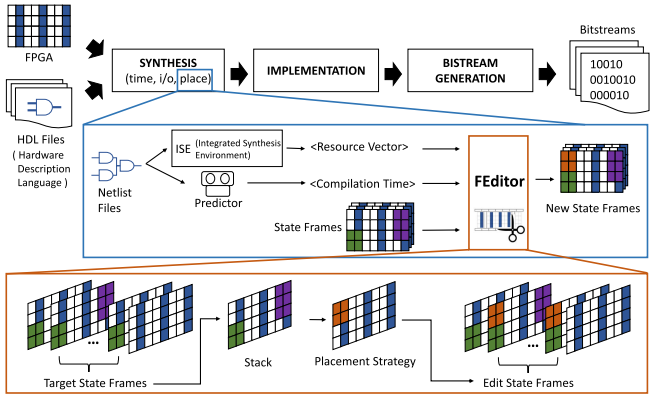


Fig. 1. Compilation flow of FPGAs with FEditor.

With the advancement of semiconductor technology, FPGAs integrate a greater diversity and quantity of hardware resources [4], [7]. However, current datacenters assign individual FPGAs with static partitions to tasks [1], [2], [3] rather than sharing them among multiple tasks. To simplify management, all FPGAs have the same quantity and layout of hardware resources. Such rigidity incurs internal fragmentation in each unit for small tasks while large tasks could be rejected as they cannot seamlessly occupy those unused regions in multiple units [39].

As shown in Fig. 1, the placement algorithm arranges task regions and manages resources on FPGAs. Therefore, it should adapt to datacenter environments to reduce the waste and rejection rates. In datacenters, resources on an FPGA are different and non-uniformly arranged, while tasks arrive consecutively with different resource requirements. The placement algorithm must meet the following requirements.

1) Dynamic Task Placement With Lifetime Awareness: the algorithm should map the consecutively arrived tasks onto FPGAs with the consideration of all tasks' life cycles for usage optimality.

2) Support of Adjustable Task Shapes With Resource Type Awareness: the algorithm should reshape tasks to satisfy resource requirements when they are placed in different locations, as resource distribution varies there.

Received 8 May 2025; revised 25 August 2025; accepted 7 October 2025. Date of publication 13 October 2025; date of current version 20 November 2025. This work was supported by the National Natural Science Foundation of China (Grant No. 62472043). Recommended for acceptance by A. Li. *(Yanyan Li and Yu Chen contributed equally to this work.)(Corresponding authors: Yawen Wang; Hai Jiang.)*

Yanyan Li, Yu Chen, and Yawen Wang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: liyanyan@bupt.edu.cn; chenyu501@bupt.edu.cn; wangyawen@bupt.edu.cn).

Zhiqian Xu and Hai Jiang are with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: zhiqian.xu@bupt.edu.cn; hai.jiang@bupt.edu.cn).

Keqin Li is with the College of Computer Science and Electronic Engineering, HunanUniversity, and the National Supercomputing Center in Changsha, Hunan, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz NY12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TPDS.2025.3620384

With these two requirements, tasks can be allocated to specific regions, whose resources meet the requirements, rather than the entire FPGA with static partitions to reduce the resource waste and task rejection rates.

Unfortunately, current placement algorithms cannot satisfy these two requirements at the same time due to the assumption of a static task set with non-adjustable shapes. FEditor is the first placement algorithm with considerations of both features. It maximizes resource utilization and acceptance rate based on Dynamic Partial Reconfiguration (DPR) [37] and the slotless partitioning [15]. DPR partitions an FPGA into multiple rectangular regions, enabling a task to load its bitstream onto a pre-placed region without any interference with others. The slotless partitioning enables the placement algorithm to dynamically reshape each region for resource utilization optimization.

As shown in Fig. 1, FEditor can be integrated within the conventional compilation flow. It obtains a resource requirement vector and a predicted compilation time from a netlist file. Then, it stacks all *State Frames* (*SFs*) within the life cycle of a task and generates an optimal placement strategy. At last, these selected *SFs* are edited based on the strategy for a relatively optimal placement.

FEditor expands the planar FPGA model into a three-dimensional (3D) model with a time dimension to support dynamic task placement with lifetime awareness. To losslessly compress the 3D model, FEditor introduces *SFs* to record essential states, simplify resource management, and transform the three-dimensional placement problem into several finite two-dimensional sub-problems.

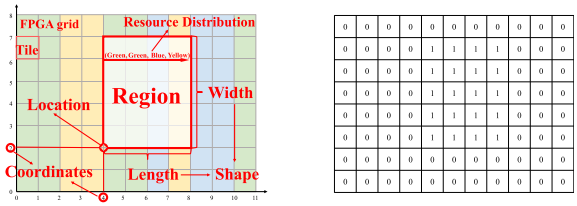
FEditor also expands the placement strategy into a quadruplet, including location (x or y axis coordinates) and shape (length and width) factors, to support adjustable task shapes with resource type awareness. Three metrics are proposed to evaluate candidate placements and formulate the placement problem as a combinatorial optimization. A nested heuristic algorithm is developed to accelerate the process by partitioning the search space and parallelizing computations.

Overall, FEditor makes the following contributions:

1) FEditor[1] is the first dynamic placement algorithm supporting consecutive tasks with adjustable shapes. It expands the FPGA model into a 3D version and the placement strategy uses a quadruplet for task placement.
2) *State Frames* (*SFs*) are proposed to simplify resource management by compressing expanded FPGA models and transforming the 3D placement problem into finite 2D sub-problems for acceleration.
3) FEditor designs three metrics on fragmentation and compactness to evaluate candidate placements as well as a nested heuristic algorithm to accelerate the combinatorial optimization.

This paper is organized as follows. Section II reviews related work. Section III describes the expanded 3D problem. Section IV defines the 2D sub-problems. Metrics are detailed in Section V. Section VI formulates the problem and describes the nested algorithm. Experimental results are presented in Section VII. Section VIII concludes and states the future work.

[1]https://github.com/NGDC-bupt/FEditor



(a) Definitions (b) Bitmap abstraction

Fig. 2. Terms for task placement. Different colors indicate different tile types. The grid/layout is abstracted into a bitmap.

II. Related Work

There have been numerous previous placement algorithms based on fixed-size regions such as DML [11] and ExHiPR [35], where FPGAs were pooled and partitioned into slots for task execution [42]. Although they were relatively easy, the resource utilization was low.

R3TOS [15] wrapped tasks with input and output data buffers and partitioned FPGAs into rectangular regions with arbitrary shapes, named slotless partitioning. Such partitioning enabled placement algorithms to adjust the location and shape of a region for a task's requirement.

There are many algorithms focusing on optimizing locations based on the slotless partitioning. Maximal Empty Rectangles (MERs) were used to manage single-type-resource FPGAs [5], [32] by assuming that there is only one type of resource on an FPGA board. MERs can be split and merged as a buddy system [23], [24], [25], although other memory management methods are introduced as well [29], [40]. A binary tree was introduced to organize these MERs [8] and a multifork tree was proposed to accelerate the splitting and merging [37]. For better flexibility, vertex links were developed [30] to record boundary vertices of idle spaces and edges were weighted by the execution durations of corresponding tasks. A placement strategy is generated based on these weights [33], although only static task sets are considered. To place consecutive tasks, the FPGA model was expanded with a time dimension [30] as the Three-Dimensional Bin Packing (3DBP) problem [21], [22], [26]. However, these algorithms were designed for single-type-resource FPGAs due to the complexity of three-dimensional resource management.

To support placement on multi-type-resource FPGAs, a region with the same resource distribution, which will be described in Fig. 2(a), was selected as the task required [12], [17], [20]. Multi-type-resource FPGAs are equipped with multiple types of resources on the board. Although overlap graphs were introduced to handle placement conflicts [13], [36], they were only designed for static task sets and suffered from low acceptance rate. A few algorithms have attempted to update overlap graphs dynamically for consecutive tasks, but suffered from high computational overheads [18].

Few existing algorithms focus on adjusting both the location and shape of a region simultaneously. Identical micro-slots could be dynamically merged into a large region for a task [14]. This mechanism needs extra resources to implement communications. For flexibility, tasks are constructed from pre-defined modular components and their alternative implementations [8]. Look-up tables are used to record all feasible locations and

TABLE I
NOTATIONS AND DESCRIPTIONS

x_F	Maximum row index
y_F	Maximum column index
T	Theoretically infinite but set as the last timestamp for analysis
$\boldsymbol{f_k}$	The vector of piecewise functions
k	The type of resources in a tile, including clb, bram and dsp
x	The x-index of a candidate region
y	The y-index of a candidate region
l	The length of a candidate region
w	The width of a candidate region
$\boldsymbol{r_s}$	The resource supplement vector of a region
$\boldsymbol{r_t}$	The resource requirement vector of a task
$\boldsymbol{t}$	The timestamp vector of a task
p	The placement of a task during an iteration
$state$	The continuous states of an FPGA
SF	The state frame
CT	The set of Collected Timestamps
$\boldsymbol{S}$	The bitmap of occupation state of an FPGA
$\boldsymbol{S_p}$	The stacked bitmap in a duration

shapes of modules and place consecutive tasks by enhanced overlap graphs. However, such mechanism was limited to single-type-resource FPGAs.

III. THE EXPANDED 3D PROBLEM

Multi-resource-type FPGAs and consecutive tasks are modeled to address the consecutive task placement problem with adjustable shapes. Its NP-hardness is also proved. Related notations are listed in Table I.

A. FPGA Model

Resources on FPGAs, including configurable logic blocks (clbs), block random access memories (brams), and digital signal processors (dsps), are interconnected via switch boxes. Multiple identical resources compose a minimal placement unit, called a tile. They have the same physical size and are arranged in a grid on an FPGA. Since the quantity of semiconductors in each tile remains the same and different types of resources change in the number of transistors required for the composition, the number of resources in a tile varies with its type. For instance, a CLB tile might contain eight clbs while a DSP tile has two dsps. Each column in the FPGA grid consists of tiles with the same type. The placement algorithm assembles neighboring tiles into a region assigned to a task. As shown in Fig. 2(a), the FPGA board can be abstracted as a grid with coordinate axes. The resource distribution indicates the order in which different resources are arranged along the row dimension.

FEditor further expands the planar model into a 3D model with a time dimension to support consecutive placements. The grid plane is assigned with an infinite height, which abstracts the FPGA into a box. The region of a task is assigned with

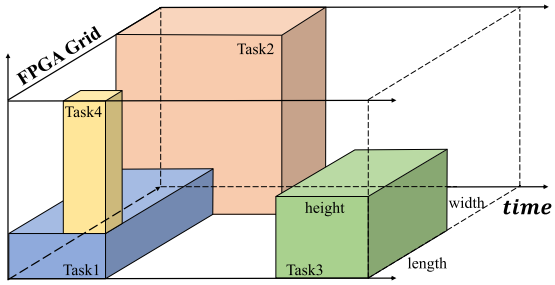


Fig. 3. A placement example of four tasks. Tasks are labeled based on their arrival orders. Tasks 1, 2 and 4 overlap in time. Task 3 overlaps with tasks 1 and 4 in space. Here, the resource difference is ignored for simplicity.

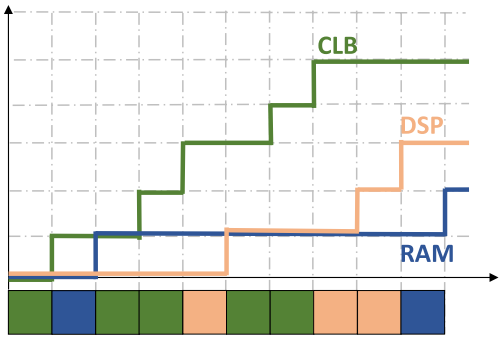


Fig. 4. Piecewise functions of different resources. The shaded blocks at the bottom illustrate the resource distribution of every row.

the same height as the execution period, which abstracts the task into a cuboid. Since resources are non-uniformly arranged on an FPGA, different locations where a task could be settled lead to different task shapes. Therefore, the bottom surface of a cuboid is adjustable according to the resource requirement and the location as shown in Fig. 3.

Therefore, each FPGA can be formulated as a quadruplet:

$$\text{FPGA} = (x_F, y_F, T, \boldsymbol{f_k}), k \in \{\text{clb}, \text{bram}, \text{dsp}\}, \quad (1)$$

where x_F and y_F denote the maximum row and column indices, T is its height and $\boldsymbol{f_k}$ represents piecewise functions of different resource types. T is theoretically finite and set as the timestamp for analysis when all tasks are removed. Regions on an FPGA can be formulated as (x, y, l, w), where (x, y) indicates the coordinates of the bottom-left corner and (l, w) specifies its length and width. Obviously, a feasible placed region must be within an FPGA area, stated as:

$$C1: \ 0 \le x < x + l \le x_F, \quad (2)$$

$$C2: \ 0 \le y < y + w \le y_F. \quad (3)$$

Since tiles with the same x are identical, FEditor uses a piecewise function for each resource type to abstract its distribution on an FPGA as shown in Fig. 4. The number of tiles between two x-values can be calculated by subtracting their function values. Therefore, the quantity of a certain type of resource in a placed region can be calculated by:

$$r_{s,k} = w \cdot (f_k(x + l) - f_k(x)), k \in \{\text{clb}, \text{bram}, \text{dsp}\}. \quad (4)$$

Resource supply of a placement can be modeled as a vector:

$$\boldsymbol{r_s} = \langle r_{\text{clb}}, r_{\text{bram}}, r_{\text{dsp}} \rangle. \tag{5}$$

Since a tile is the unit of axes on an FPGA grid, FEditor gets the overall quantity of resources in a region by $L1$ norm:

$$\sum r_{s,k} = ||\boldsymbol{r_s}||_1 = l \cdot w. \tag{6}$$

B. Task Model

FEditor specifies a task in three aspects, including timestamp $\boldsymbol{t}$, resource requirement $\boldsymbol{r_t}$ and placement p, denoted as:

$$\text{task} = (\boldsymbol{t}, \boldsymbol{r_t}, p). \tag{7}$$

Each task is assigned with four timestamps, including arrival timestamp t_{in}, loading timestamp t_{load}, removal timestamp t_{remove} and deadline timestamp t_{ddl}. Arriving timestamp t_{in} can be recorded when the netlist file of a task is received by FEditor. As shown in Fig. 1, a task is placed into a region in the synthesis phase and compiled into a bitstream in the implementation and bitstream generation phases before being loaded to an FPGA. FEditor calculates the real loading timestamp as:

$$t_{\text{load}} = t_{\text{in}} + t_{\text{FEditor}} + t_2 + t_3. \tag{8}$$

In (8), t_2 is the period of implementation phase, and t_3 is the period of the bitstream generation phase. A deep learning model predicts the value of $t_2 + t_3$ based on the netlist file with accuracy exceeding 95% [35]. FEditor uses the model to predict the time, denoted as $\Delta t_{2,3}$ and sets a maximum duration $\Delta t_{\text{FEditor}}$ based on historical placements to ensure its completion before the timeout. Therefore, an approximate loading timestamp can be calculated as:

$$t'_{\text{load}} = t_{\text{in}} + \Delta t_{\text{FEditor}} + \Delta t_{2,3} \geq t_{\text{load}}. \tag{9}$$

The removal timestamp t_{remove} indicates when to release resources. FEditor obtains it by adding execution period to the loading timestamp. The execution period can be derived from the quantity of its data. Additionally, a deadline timestamp is set for each task based on its Service Level Agreement (SLA). Tasks can be delayed but have to terminate before the deadline. This relaxation aligns with current datacenter practices. Based on the above, the complete form of the timestamp vector is expressed as:

$$\boldsymbol{t} = \langle t_{\text{in}}, t'_{\text{load}}, t_{\text{remove}}, t_{\text{ddl}} \rangle. \tag{10}$$

FEditor uses Integrated Synthesis Environment (ISE) to extract the resource requirement of a task from its netlist file. The requirement is formulated as:

$$\boldsymbol{r_t} = \langle r_{\text{clb}}, r_{\text{bram}}, r_{\text{dsp}} \rangle. \tag{11}$$

The placement of a task is a region on an FPGA. Therefore, the formulation of a placement p is:

$$p = (x, y, l, w). \tag{12}$$

As a legal placement must satisfy resource requirements and a tile should only be assigned to one task at each moment, constraints are derived as:

$$C3: \ r_{s,k} \geq r_{t,k}, k \in \{\text{clb}, \text{bram}, \text{dsp}\}, \tag{13}$$

$$C4': \ \text{tasks cannot overlap in both time and space.} \tag{14}$$

C. 3D Problem Definition

Based on the above models, the expanded 3D placement problem is defined, similar to the 3DBP [21], [22], [26].

Definition 1: The expanded 3D placement problem for consecutive tasks with adjustable shapes is defined as:

- *Input:* An FPGA, $(x_F, y_F, T, \boldsymbol{f_k})$, a list of early placed tasks, $[(\boldsymbol{t}, \boldsymbol{r_t}, p)]$, and an arrived task, $(\boldsymbol{t_0}, \boldsymbol{r_{t,0}})$.
- *Output:* A placement strategy, p_0, if and only if $C1$–$C4'$ are all satisfied while achieving the most on resource utilization and acceptance rate calculated by time-overlapped tasks, or a rejection.

Compared with a conventional 3DBP, the internal substances of the box and cuboids in FEditor are unevenly distributed, because the resources on a FPGA grid are different. This makes the expanded placement problem harder than the 3DBP. To prove this NP-hardness, a reduction from the 3DBP is constructed.

Theorem 1: The expanded placement problem for consecutive tasks with adjustable shapes on FPGAs is NP-hard.

Proof: The 3DBP problem can be reduced to an instance of the task placement problem. In 3DBP, there exists a given set of items $I = \{i_1, i_2, \ldots, i_n\}$ and a box with size, (L, W, H). Each item i_j has a 3D size, (l_j, w_j, h_j). The 3DBP solver needs to assign every item a location (x_j, y_j, t_j). Its constraints require that items do not overlap and do not exceed the box, which can be formulated as:

$$\forall i, 0 \leq x_j + l_j \leq L, 0 \leq y_j + w_j \leq W, 0 \leq t_j + h_j \leq H. \tag{15}$$

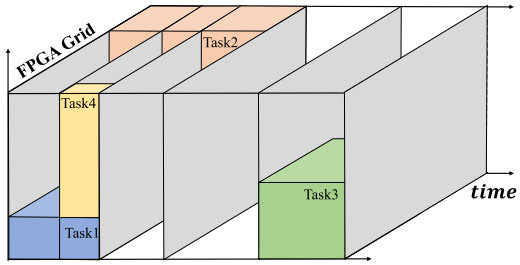
We define the FPGA in the instance as (L, W, H). For each task, let its $\boldsymbol{t}$ be $\langle t_j, t_j, t_j + h_j \rangle$, $\boldsymbol{r_t}$ be $\langle l_j w_j, 0, 0 \rangle$ and p be (x_j, y_j, l_j, w_j). Constraints of 3DBP are equivalent to those of $C1, C2$ and $C5$. The assertion stated in the preceding paragraph holds.

If 3DBP has a solution, we can construct placements in the instance. To simplify, we denote the solution as s with a location, (x_j, y_j, t_j) to each item i_j with size, (l_j, w_j, h_j). This solution can be transformed into the placements in the instance. A task T_j is assigned (x_j, y_j, l_j, w_j) with attributes $\langle t_j, t_j, t_j + h_j \rangle$ and $\langle l_j w_j, 0, 0 \rangle$. Since the constraints are equivalent, these placements are valid.

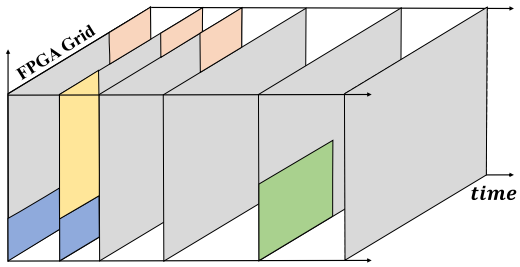
As the transformation above only involves variable mapping, the reduction from 3DBP to the placement problem can be completed in polynomial time. Therefore, the problem is at least as difficult as 3DBP. As 3DBP is a proved NP-hard problem, the placement for consecutive tasks with adjustable shapes on FPGAs is NP-hard. ■

IV. The Simplified 2D Sub-Problems

Developing a high-performance solver for the expanded placement problem is extremely challenging due to its complexity. An unconventional representation, *State Frames*



(a) Special timestamps



(b) Generated *SFs*

Fig. 5. The generation of *SFs* in the example shown in Fig. 3.

(*SFs*) is used to simplify the resource management and the expanded 3D problem. With *SFs*, the 3D problem is transformed into a finite number of 2D sub-problems, providing an opportunity for the development of high-performance solvers.

A. State Frames

Rather than recording information about the box and cuboids as shown in Fig. 3, FEditor records the state of an FPGA. The state is defined as the occupation states of tiles as shown in Fig. 2(b). FEditor uses $(t, s_{x,y})$ to represent the occupation state of a tile, where t indicates the timestamp and $s_{x,y}$ shows whether the tile at location (x, y) is occupied(1) or idle(0) with this timestamp. Therefore, the FPGA state is formulated as:

$$state = (t', \boldsymbol{S}), t' \in [0, T], \boldsymbol{S} = (s_{x,y}) \in \{0,1\}^{x_F \times y_F}. \quad (16)$$

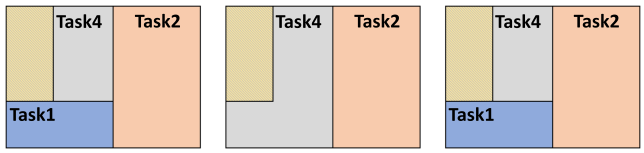
The state exhibits the continuous characteristics of the variable t', while tasks do not arrive continuously. Therefore, the gap between this state representation and the actual task arrivals leads to substantial amounts of redundant information, which complicates resource management.

State Frames eliminate unnecessary information for the special states of an FPGA. They are generated whenever a task is loaded onto or removed from the FPGA, as the variable $\boldsymbol{S}$ only changes in those moments. To simplify, a set CT is defined to include all related timestamps. An *SF* can be formulated as:

$$SF = (t, \boldsymbol{S}), t \in CT, \boldsymbol{S} = (s_{x,y}) \in \{0,1\}^{x_F \times y_F}. \quad (17)$$

This operation is similar to extracting frames from the continuous model, as shown in Fig. 5(a) and (b).

SFs compress the 3D FPGA model losslessly as the occupation state of every tile with any timestamp can be obtained. To get (t_i, s_{x_0,y_0}), FEditor finds out an *SF* $(t_j, \boldsymbol{S_j})$ satisfying $t_j \le t_i < t_{j+1}$. The state is equal to $\boldsymbol{S_j}[x_0, y_0]$.



(a) t^4_{in} (b) $t^{1,4}_{\text{remove}}$ (c) Stacked *SF*

Fig. 6. The stacking process of *SFs* related to task 4 in the example shown in Fig. 3.

Since compilation time is considered in FEditor, early arrived tasks may be loaded onto the FPGA after later arrived tasks, such as tasks 3 and 4 in Fig. 3. As the resources assigned to the early task will not be used until its loading time, the latter task can be overlapped with the earlier one in space to increase resource utilization. To avoid conflicts, FEditor selects *SFs* throughout the entire life-cycle of the task and evaluates a placement strategy on all selected *SFs*. In this way, the 3D continuous placement problem is simplified into a finite number of 2D sub-problems. Rather than generating a placement strategy according to each selected *SF*, FEditor stacks all those *SFs* to construct one feasible search space. Therefore, a candidate placement is generated from the stacked *SF* and evaluated from all selected *SFs*. We formulate this operation as:

$$\boldsymbol{S_p} = \text{stack}(SF_m, \ldots, SF_n) = \boldsymbol{S_m} \vee \cdots \vee \boldsymbol{S_n}, \quad (18)$$

where m and n correspond to t_{load} and t_{remove} respectively, as shown in Fig. 6. Since the placed tasks are not recorded, the complex constraint $C4$ can be simplified into:

$$C4: \ \boldsymbol{S_p}[x + l', y + w'] = 0, 0 \le l' \le l, 0 \le w' \le w. \quad (19)$$

Additionally, conditions in (18) are formulated as:

$$C5: \ 0 \le t_m \le t_n \le T, \quad (20)$$

$$C6: \ t_m \le t_{\text{load}} \text{ and } t_{m+1} > t_{\text{load}}, \quad (21)$$

$$C7: \ t_n \le t_{\text{remove}} \text{ and } t_{n+1} > t_{\text{remove}}. \quad (22)$$

B. 2D Sub-Problem Definition

Based on *SFs* and the stacking process, the simplified 2D sub-problems can be stated as the follows.

Definition 2: The simplified 2D placement sub-problems for consecutive tasks with adjustable shapes are defined as:

- *Input:* An FPGA, $(x_F, y_F, T, \boldsymbol{f_k})$, a list of created *SFs*, $[(t, \boldsymbol{S})]$, and an arrived task, $(\boldsymbol{t_0}, \boldsymbol{r_{t,0}})$.
- *Output:* An edited list of *SFs*, $[(t, \boldsymbol{S})]$ based on a placement p_0, if and only if $C1$–$C7$ are all satisfied while achieving the most on resource utilization and acceptance rate calculated by *SFs*, or a rejection with non-edited *SFs*.

Straightforwardly, solvers of both problems obey the same framework. First, time-overlapped tasks or *SFs* are found out. Second, a candidate placement is generated. Third, legality check is conducted. Fourth, metric calculations are performed for a legal placement. Fifth, task lists or *SFs* are updated. To evaluate the simplification, we assume that the second and fifth steps are the same while the first step can be completed in one

search with a hash. We can focus on the processing time of legality check and metric calculation.

For rejections, the two solvers handle deadlines in the same way, updating timestamps and repeating the framework. Therefore, the proof is established for a legal candidate.

Theorem 2: The simplified 2D sub-problems reduce computation times compared with the expanded 3D problem.

Proof: We suppose that there are n early placed tasks and the candidate placement strategy is $p_0 = (x_0, y_0, l_0, w_0)$.

In the simplified 2D sub-problems, the legality check is conducted once on the stacked *SF*. The metric calculations are applied to every selected *SF*. Therefore, the computation time is equal to the number of selected *SFs*, $O(m)$. Theoretically, each task can create at most 2 *SFs*. Therefore, $0 < m \leq 2n'$, where n' is the number of time-overlapped tasks.

In the expanded 3D problem, the legality check is conducted on every task, resulting in the computation time of $O(n')$. Additionally, the metric calculations are performed on all time-overlapped tasks. Therefore, the cost of metric calculations is also $O(n')$.

The overall computation time of the simplified 2D sub-problems is $O(1+m) \leq O(1+2n')$ while the expanded 3D problem is $O(2n')$. Tasks in datacenters arrive consecutively and densely, leading to multiple tasks' arriving and completing at the same time. Additionally, placement algorithms will optimize metrics to overlap tasks for higher resource utilization and acceptance rate. Both features reduce the number of *SFs* generated by each task. Therefore, with $m \ll 2n'$, the simplification by *SFs* is proved. ■

V. Metrics on Fragmentation and Compactness

To solve the 2D sub-problems with high resource utilization and acceptance rate, FEditor relies on three metrics for internal fragmentation, external fragmentation, and placement compactness to evaluate task placement candidates.

A. Internal Fragmentation

Internal fragmentation quantifies the extent of unused resources within placed regions. Those resources cannot be allocated to other tasks until the task is removed. This occupation will decrease resource utilization. A metric is designed to help minimize internal fragmentation. As the quantities of different resources vary from each other, the metric values of different types are measured respectively. A resource supply vector and a resource requirement vector are used to calculate an internal fragmentation vector as:

$$\Delta \boldsymbol{r} = \left\langle \frac{r_{s,k} - r_{t,k}}{r_{s,k}} \right\rangle_3, k \in \{\text{clb, bram, dsp}\}. \tag{23}$$

With the help of constraint $C3$, we derive that:

$$\Delta \boldsymbol{r} \propto \boldsymbol{r_s} - \boldsymbol{r_t} \geq \boldsymbol{0}. \tag{24}$$

FEditor calculates the metric for internal fragmentation based on the weighted $L1$ norm of the vector:

$$a = \Omega_w(\Delta \boldsymbol{r}) = \sum_k \Delta r_k \cdot w_k. \tag{25}$$

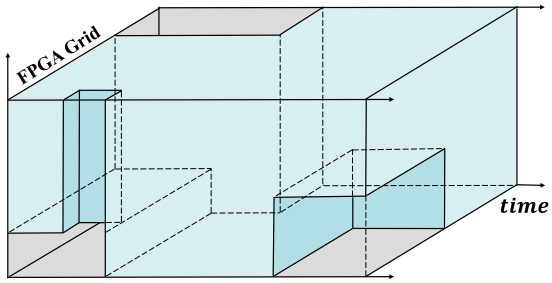


Fig. 7. The three-dimensional objects of idle resources in the example illustrated in Fig. 3.

Rare resources are considered first. Therefore, the weight is calculated by the exponential decay of resource quantities on the entire FPGA, R_k:

$$w_t = \frac{e^{-kR_k}}{\sum_k e^{-kR_k}}. \tag{26}$$

Based on the formulation, the value range of metric a is [0,1). When quantities of resources in the region exactly meet the requirement, a is 0. The more resources are wasted, the closer to 1 the value is. As this metric can be calculated for a placement, we denote $a = f_a(x, y, l, w)$.

B. External Fragmentation

Only neighbouring tiles can be assembled into a region. External fragmentation evaluates the degree of dispersion by the compactness of idle resources based on their shapes. Tasks can be rejected if idle resources are scattered. In other words, there are task sets that can be accepted in compact shapes and rejected in scattered shapes. Therefore, FEditor defines that the more compact the idle shapes are, the greater the number of acceptable task sets will be.

Since FPGAs are modeled as a box, idle resources are shaped into three-dimensional irregular objects as shown in Fig. 7. Accepted tasks are cuboids that can be placed within those objects. FEditor defines *Fill Set* as the set of those accepted cuboids, elements of which can exactly fill the given idle object. Therefore, the number of *Fill Sets* reflects the compactness of the object.

We use $\Gamma(f)$ to denote the number of *Fill Sets* for an idle space f. If a task placement p_1 exhibits less external fragmentation than another one p_2, we can derive:

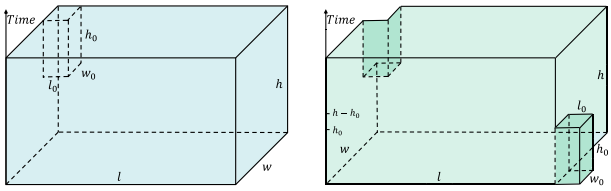
$$\Gamma(f_1) > \Gamma(f_2). \tag{27}$$

Theorem 3: A cuboid owns the most Fill Sets.

Proof: Without losing generality, let f_1 be a cuboid and f_2 be an irregular object. Since the two candidates are generated from identical $\boldsymbol{S_p}$, the overall search space can be formulated as $(x_F, y_F, t_{\text{remove}} - t_{\text{load}})$. Therefore, the volume of remaining resources, denoted as V_f, can be derived as:

$$V_f = V_b - V_o - V_t, \tag{28}$$

where V_b, V_o and V_t are the volumes of the partial box, other tasks and candidate placement, respectively. Since different



(a) Cuboid (b) Irregular Object

Fig. 8. Two possible shapes of idle resources.

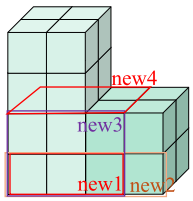


Fig. 9. Generation of new cuboids with adjacent surfaces.

placements only influence V_c, we only calculate its value:

$$\begin{aligned} V_c &= (l_p w_p) \cdot (t_{\text{remove}} - t_{\text{load}}) \\ &= ||\boldsymbol{r}_{\boldsymbol{s},p}||_1 \cdot \Delta t \\ &= (||\boldsymbol{r}_{\boldsymbol{t},p}||_1 + ||\Delta \boldsymbol{r}_p||_1) \cdot \Delta t, p \in \{1, 2\}. \end{aligned} \tag{29}$$

As FEditor will minimize the internal fragmentation, we can suppose both candidates have no fragmentation:

$$f_a(x_1, y_1, l_1, w_1) = f_a(x_2, y_2, l_2, w_2) = 0. \tag{30}$$

According to (30) and (25), we can derive:

$$\Delta \boldsymbol{r}_{,1} = \Delta \boldsymbol{r}_{,2} = \boldsymbol{0}. \tag{31}$$

Therefore, the values of V_c of two candidates are identical:

$$V_{f,1} = V_{f,2}. \tag{32}$$

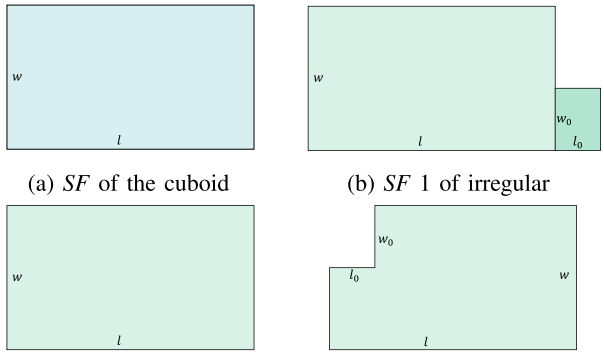
Based on the equality, we suppose a basic example shown in Fig. 8, where f_2 removes a small cuboid (w_0, h_0, l_0) from the cuboid (w, h, l) (i.e., f_1) and appends it to another surface. Since the bottom and top surfaces along the time axis are fixed as t_{load} and t_{remove}, the small cuboid can only be appended to surrounding surfaces. Similarly, we can regard f_1 as placing the convex cuboid into the hollow. For simplicity, the convex cuboid is denoted as f_{conv} and the common part of f_1 and f_2 is denoted as f_{comm}. Therefore, we can count their *Fill Sets:*

$$\Gamma(f_p) = \Gamma(f_{\text{comm}}) + \Gamma(f_{\text{conv}}) + \Gamma_p(f_{\text{conv}}). \tag{33}$$

$\Gamma(f_{\text{comm}})$ and $\Gamma(f_{\text{conv}})$ are the numbers of *Fill Sets* of the common part and the convex cuboid respectively. $\Gamma_p(f_{\text{conv}})$ indicates the number of new *Fill Sets* created by combining f_{comm} and f_{conv} according to f_p.

A unit cuboid represents a tile with a height of one time unit. Constructing a *Fill Set* involves grouping these unit cuboids within an object. $\Gamma_p(f_{\text{conv}})$ is determined by counting the number of new cuboids that f_{conv} adds to f_{comm} as shown in Fig. 9. As only adjacent surfaces will expand the cuboid in f_{comm}, each surface will generate new *Fill Sets* as:

$$\Gamma_0(f_{\text{conv}}) = l_0 \cdot w_0 \cdot h_0. \tag{34}$$



(a) *SF* of the cuboid (b) *SF* 1 of irregular

(c) *SF* 2 of irregular (d) *SF* 3 of irregular

Fig. 10. *SFs* of the case.

The convex cuboid is adjacent to f_1 with three surfaces and f_2 with one. We can derive:

$$\begin{aligned} \Delta\Gamma_{12} &= \Gamma_1(f_{\text{conv}}) - \Gamma_2(f_{\text{conv}}) \\ &= (3 - 1) \cdot \Gamma_0(f_{\text{conv}}) > 0. \end{aligned} \tag{35}$$

Therefore, the cuboid in this case owns more *Fill Sets:*

$$\Delta\Gamma = \Gamma_{\text{cuboid}} - \Gamma_{\text{object}} > 0. \tag{36}$$

Moreover, we can reduce all irregular objects to this basic case through partitioning. If p_2 creates multiple isolated irregular objects, we assess each object separately and evaluate the quality by their summaries. According to (36), the summary yields a positive result:

$$\Delta\Gamma = \sum_{\text{objects}} \Delta\Gamma_{\text{object}} = \sum_{\text{objects}} (\Gamma_{\text{cuboid}} - \Gamma_{\text{object}}) > 0. \tag{37}$$

If those objects are all cuboids, the number of their *Fill Sets* will also be smaller than that of an entire cuboid. The *Fill Set* containing the exact larger cuboid will be incompatible with the smaller ones. Therefore, the difference is at least one:

$$\Delta\Gamma = \Gamma_{\text{big\_cuboid}} - \sum \Gamma_{\text{small\_cuboid}} \geq 1. \tag{38}$$

Consequently, the theorem holds in all cases.

Theorem 4: A rectangle owns the most Fill Sets.

Proof: The case in Fig. 8 creates multiple *SFs* as illustrated in Fig. 10. In period $[t_{\text{load}}, t_{\text{remove}}]$, f_1 generates one *SF* lasting for the whole period, while f_2 generates three. We can calculate based on the rectangular:

$$\Gamma(f_1) = 0, \tag{39}$$

$$\Gamma(f_2) = w_0 l_0 h_0 - 2 \cdot w_0 l_0 h_0 = -w_0 l_0 h_0 < 0. \tag{40}$$

We get the same conclusion with Theorem 3:

$$\Gamma(f_1) > \Gamma(f_2). \tag{41}$$

As proof in Theorem 3, this theorem holds in all cases. ■

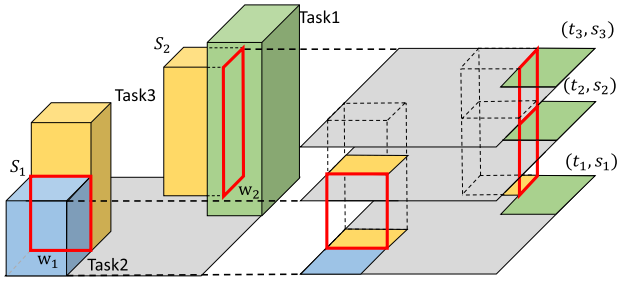
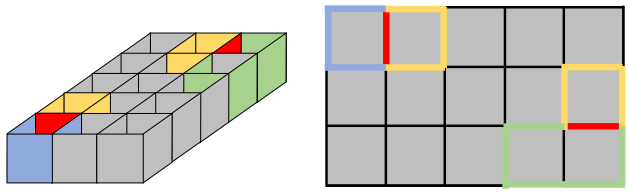


Fig. 11. Measuring allocated compactness.



(a) Possible surfaces (b) Possible edges

Fig. 12. Placement compactness of the two candidates. Yellow ones are boundaries of candidate. Blue and green ones are placed tasks'. Red ones are adjacent surfaces or edges created by candidate placements.

FEditor uses the difference between idle spaces and their minimal bounding rectangles to represent external fragmentation, as defined in Theorem 4:

$$b = \sum_{\text{islands}} \left(\frac{\text{area}_{\text{island}}}{\sum_{\text{islands}} \text{area}_{\text{island}}} \cdot \frac{\text{area}_{\text{island}}}{\text{minRecArea}_{\text{island}}} \right) = \frac{1}{\sum_{\text{islands}} \text{area}_{\text{island}}} \cdot \sum_{\text{islands}} \frac{\text{area}^2_{\text{island}}}{\text{minRecArea}_{\text{island}}}. \quad (42)$$

For simplicity, we denote $b = f_b(x, y, l, w)$ and its value range is [0,1]. When the remaining space is a rectangle, $b = 1$. If there is no idle resource, $b = 0$.

C. Placement Compactness

Placement compactness evaluates idle resources in task placements to avoid hollows in idle spaces. FEditor uses the area of adjacent surfaces between tasks and boundaries to quantify placement compactness. Maximizing this metric will make the placement strategy closer to other tasks and FPGA boundaries. Therefore, hollows can be minimized.

In Fig. 11, there are two candidate placements for task 3 while tasks 1 and 2 have been loaded onto the FPGA. As shown in red rectangles, adjacent surface S_2 owns the higher area. Therefore it exhibits better placement compactness. We can calculate those areas:

$$S_1 = w_1 \cdot \min(t_{\text{task1}}, t_{\text{task3}}), \quad (43)$$

$$S_2 = w_2 \cdot \min(t_{\text{task2}}, t_{\text{task3}}). \quad (44)$$

There are three *SFs* in Fig. 11. The adjacent surfaces crossing multiple *SFs* can be decomposed into multiple segments aligned with *SFs:*

$$S_1 = e_1 \cdot (t_2 - t_1), \quad (45)$$

$$S_2 = S_2^1 + S_2^2 = e_2 \cdot (t_2 - t_1) + e_2 \cdot (t_3 - t_2). \quad (46)$$

As shown in Fig. 12(a), surfaces between two *SFs* have the same height. Therefore, the area of an adjacent surface can be converted into multiple areas of a unit surface:

$$S = n \cdot S_{\text{unit}}. \quad (47)$$

The maximal coefficient of the area equals to the sum of boundaries of a placement:

$$S_{\text{ideal}} = 2 \cdot (l + w) \cdot height. \quad (48)$$

Therefore, we have the ratio between areas of adjacent surfaces and maximal values to quantify placement compactness as:

$$\Theta(S) = \frac{S}{S_{\text{ideal}}} = \frac{e \cdot height}{2 \cdot (l + w) \cdot height} = \frac{e}{2 \cdot (l + w)}. \quad (49)$$

FEditor uses the following equation to calculate the area S:

$$w = ||(Alloc_k \wedge A_{\text{edge}}) \vee (Alloc'_k \wedge \boldsymbol{S_p})||_0, \quad (50)$$

$$Alloc'_k = \bigvee_{direct} Alloc_k^{direct}, direct \in \{l, r, u, d\}. \quad (51)$$

The first intersection operation in (50) computes adjacent surfaces between the candidate placement and boundaries while the second computes those between the candidate one and early placed tasks. $Alloc_k$ denotes a matrix for early placed tiles. A_{edge} is a pre-defined matrix in which cells adjacent to boundaries are labeled. $Alloc'_k$ denotes the boundary of the placing region, which can be calculated in (51). The superscript $direct$ indicates the direction of each boundary. $\boldsymbol{S_p}$ is calculated in (18). After applying the union operation on the two matrices and calculating $L0$ norm, we get a scalar value representing adjacent edges.

According to the *SF* illustrated in Fig. 12(b), an adjacent surface in an *SF* is an adjacent edge. Along with the calculation of adjacent surface, we derive the metric based on adjacent edges:

$$\Theta(e) = \frac{e}{e_{\text{ideal}}} = \frac{e}{2 \cdot (l + w)} = \Theta(S). \quad (52)$$

Therefore, $\Theta(S)$ can be calculated based on *SFs*.
We define placement compactness as:

$$c = \frac{||(Alloc_k \wedge A_{\text{edge}}) \vee (Alloc'_k \wedge \boldsymbol{S_p})||_0}{2 \cdot (l + w)}. \quad (53)$$

Similarly, we denote $c = f_c(x, y, l, w)$ and its range is [0,1]. When the placement has no adjacent surfaces, $c = 0$. When the placement fills a hollow in idle spaces, $c = 1$.

VI. The Optimization Function and Algorithm

Based on the above three metrics, the simplified 2D subproblems can be formulated as a combinatorial optimization problem. A nested heuristic algorithm is implemented and explained for acceleration.

A. The Optimization Function

To increase resource utilization and task acceptance rate, FEditor defines an optimal placement strategy to minimize fragmentation and maximize compactness. Based on the above three metrics, the optimization function of the simplified 2D sub-problems can be formulated as:

$$\mathcal{P}: \min \frac{\sum_{t_i} (\alpha a_i + \beta(1-b_i) + \gamma(1-c_i)) \cdot (t_{i+1} - t_i)}{t_m - t_n},$$
$$s.t. \quad C1 - C7,$$
$$C8: \alpha + \beta + \gamma = 1, 0 \leq \alpha, \beta, \gamma \leq 1. \tag{54}$$

α, β, γ are hyperparameters, whose values will be evaluated in Section VII. Constraints $C1 - C7$ are described in the above equations while $C8$ limits the range of hyperparameters. According to the definition of *SFs*, the overall metric values should be evaluated over all selected *SFs*. This formulation uses their durations to maintain fairness among *SFs*.

B. The Nested Heuristic Algorithm

Since the placement problem is NP-hard, FEditor uses heuristic algorithms to make placements. Since searching a four dimensional space using conventional algorithms is quite costly, a nested heuristic algorithm is implemented to speed up the process. The outer loop is developed using Simulated Annealing (SA) [28], while the inner loop employs Particle Swarm Optimization (PSO) [16]. The shape of a region is related to its location, as resource distributions vary in FPGA locations. This feature enables the nested optimization. Specifically, SA is responsible for optimizing the location (x, y) whereas PSO aims to find the optimal shape (l, w) for each (x, y).

Additionally, SA and PSO can both run in parallel to further accelerate the process. The FPGA grid can be partitioned into multiple areas. Each SA worker determines the location within one area. This approach partitions the search space. PSO worker can process initializations and updates of particles in parallel for acceleration.

The pseudocode of the nested algorithm is shown in Algorithm 1. The functions, $launch*()$, launch multiple threads to run SA (lines 1–12) or PSO (lines 14–25) in parallel. Specifically, $launchSAs()$ (line 32) needs to decide whether to update the global bests. The algorithm will attempt to place the task until success or expiration (lines 30). The SA loop will call the PSO loop at each temperature (lines 2 and 9–10). SA updates the best strategy based on Metropolis Criterion [28] (line 5–7). PSO will initialize (lines 15–16) and update particles iteratively (lines 17–25). Both SA and PSO will terminate the iteration when either the iteration limit is reached or the current best placement stabilizes.

Algorithm 1: The Nested Heuristic Algorithm.

Input: Task:$(\boldsymbol{t}, \boldsymbol{r_t})$, *SFs:*$[(t_i, \boldsymbol{S_i})]$
Output: Placement Strategy:(x, y, w, h)
1: **Function** SALoop$(best, bcost, region)$:
2: **while** $T \geq T_0$**or**$cnt \geq CNT$ **do**
3: $pos \leftarrow$ UpdatePos$(region, pos)$
4: $bestSA, bcostSA \leftarrow$ PSOLoop()
5: **if** $bcostSA < bcost$**or** $random <$ exp$(-(bcostSA - bcost)/T)$ **then**
6: $best,\ bcost \leftarrow$ UpdateGB$(bestSA, bcostSA)$
7: $cnt \leftarrow 0$
8: **end if**
9: $T \leftarrow T * \alpha$
10: $cnt \leftarrow cnt + 1$
11: **end while**
12: **endFunction**
13:
14: **Function** PSOLoop:
15: $particles \leftarrow$ launchPSOs(InitParticle, $pos, \boldsymbol{S_p}, \boldsymbol{r_t}$)
16: $bestSA,\ bcostSA \leftarrow$ InitLB$(particles)$
17: **while** $iter \geq 0$**or**$cnt \leq CNT$ **do**
18: $particles \leftarrow$ launchPSOs(UpdateParticles, $particles$)
19: **if** $p \in particles$**and**$p.c < bcostSA$ **then**
20: $bestSA,\ bcostSA,\ particles \leftarrow$ UpdateLB(p)
21: $cnt \leftarrow 0$
22: **end if**
23: $iter \leftarrow iter - 1$
24: $cnt \leftarrow cnt + 1$
25: **end while**
26: **return** $bestSA, bcostSA$
27: **endFunction**
28:
29: $bests,\ bcosts \leftarrow$ Init()
30: **WHILE** $bests$**is**$empty$**or**$\boldsymbol{t}$**expires do**
31: $\boldsymbol{S_p} \leftarrow$ StackStateFrames$(\boldsymbol{t})$
32: launchSAs(SALoop,$bests, bcosts$)
33: **IF** $bests$**is**$empty$ **then**
34: $\boldsymbol{t} \leftarrow$ UpdateTime$(\boldsymbol{t}, SFs)$
35: **end if**
36: **end while**
37: **return** $bests.best$

C. Analysis of Time Complexity

Each SA worker will iterate at most $(\log T_0 - \log T)/\log \alpha$ times. And the PSO loop will iterate at most $iter$ times. Therefore, for a given task, the overall time complexity is:

$$O(iter \cdot (\log T_0 - \log T)/\log \alpha) \tag{55}$$

Without partitioning and parallelism, SA needs to increase the value of T_0 to maintain the same number of iterations while PSO needs to calculate each particle one-by-one. Supposing there are $thdsSA$ SA workers and $particles$ PSO workers, the number of SA iterations is $thdsSA$ times more than the nested one while the computations of particles need to be serialized. Approximately, the time complexity is:

$$O((iter \cdot particles) \cdot (thdsSA \cdot (\log T_0 - \log T)/\log \alpha))) \tag{56}$$

The complexity of calculating metrics for each candidate region can be determined using (25), (42) and (53). Internal fragments can be obtained via a functional operation in $O(1)$

TABLE II
SIMULATION FPGA MODELS

Board type	Zynq	Virtex	Cyclone
Tile Rows	5	7	5
Tile columns	97	146	87

TABLE III
INFORMATION OF TASK SETS

	Small	Middle	Big	Hybrid
clb	[20, 60]	[60, 100]	[100, 140]	[20, 140]
ram	[10, 30]	[30, 50]	[50, 70]	[10, 70]
dsp	[5, 15]	[15, 25]	[25, 35]	[5, 35]
tin	[1, 100]	[1, 100]	[1, 100]	[1, 100]
exe	[1, 3]	[1, 10]	[1, 20]	[1, 20]
ddl	[1, 5]	[5, 15]	[15, 25]	[1, 25]
tasks	500	500	500	500
sets	10	10	10	10

time. A structure is developed to maintain idle spaces enabling the calculation of external fragments through list lookup in $O(1)$ time. Compactness is computed based on a state frame in $O(n'^2)$ time. Parameter n' is the portion of split search space and is much smaller than the original length and width of FPGA boards. Therefore, the overall time complexity is $O(n'^2)$.

Since both parallel and sequential versions need to calculate metrics for each candidate region, the accelerating rate can disregard this portion. Therefore, our algorithm theoretically executes faster than normal nested algorithms by a factor of approximately around $(paritcles \cdot thdsSA)$.

VII. EVALUATION

FEditor is evaluated in three aspects in a simulated environment. The effectiveness of the three metrics and the nested heuristic algorithm is also verified. Additionally, the suitability of FEditor for datacenters is proved.

A. Environment Setup

The simulation is performed on an Intel Core i5-13600 K CPU (5.10 GHz) with 32 GB memory. FEditor and other compared methods are evaluated using Xilinx's mainstream FPGA families (Virtex and Zynq) and Intel's Altera Cyclone FPGAs as detailed in Table II.

Four types of task sets are used to evaluate FEditor as shown in Table III. Tasks are generated based on an uniform distributed engine. To avoid the influence of scattered task sets, the result of each type is averaged over 10 unique task sets.

As FEditor is the first consecutive task placement algorithm with adjustable shapes, we can only select three most related and most advanced algorithms to compare, including P2MC [30], 3D and LA-3D [36]. P2MC can handle FPGAs with multiple resource types based on location weights, but assumes a static set of tasks. Tasks in the set arrives simultaneously and are placed immediately. As the whole FPGA remains occupied until all loaded tasks are completed, this immediate placement strategy leads to many task expirations. Differently, 3D and LA-3D consider the time dimension to support consecutive tasks based on vertex links. But they are established on FPGAs with a single resource type. As the vertex links only record the resource states at latest timestamp, the two algorithms can only attempt to place one task each time. Therefore, tasks cannot be delayed based on their deadlines to optimize the overall resource utilization and acceptance rate.

We expand P2MC with a lazy placement strategy. The enhanced P2MC collects arrived tasks and places them at their earliest t_{ddl}. Also, we developed H3D and LA-H3D to adapt different resources on FPGAs. Starting from each vertex, they exhaustively search for the minimal length and width of each task. Then they choose the best placement. To support deadline and compilation prediction, H3D and LA-H3D record the vertex links with each special timestamp as *SFs* do. The loading timestamps can be calculated as FEditor does.

To compare with the conventional slot-based placement algorithm, we implemented a straightforward allocation strategy named FIFO-slot. This strategy partitions the FPGA board into multiple static slots. Each task will either fit into one slot or be rejected. Tasks are placed into idle slots in FIFO order.

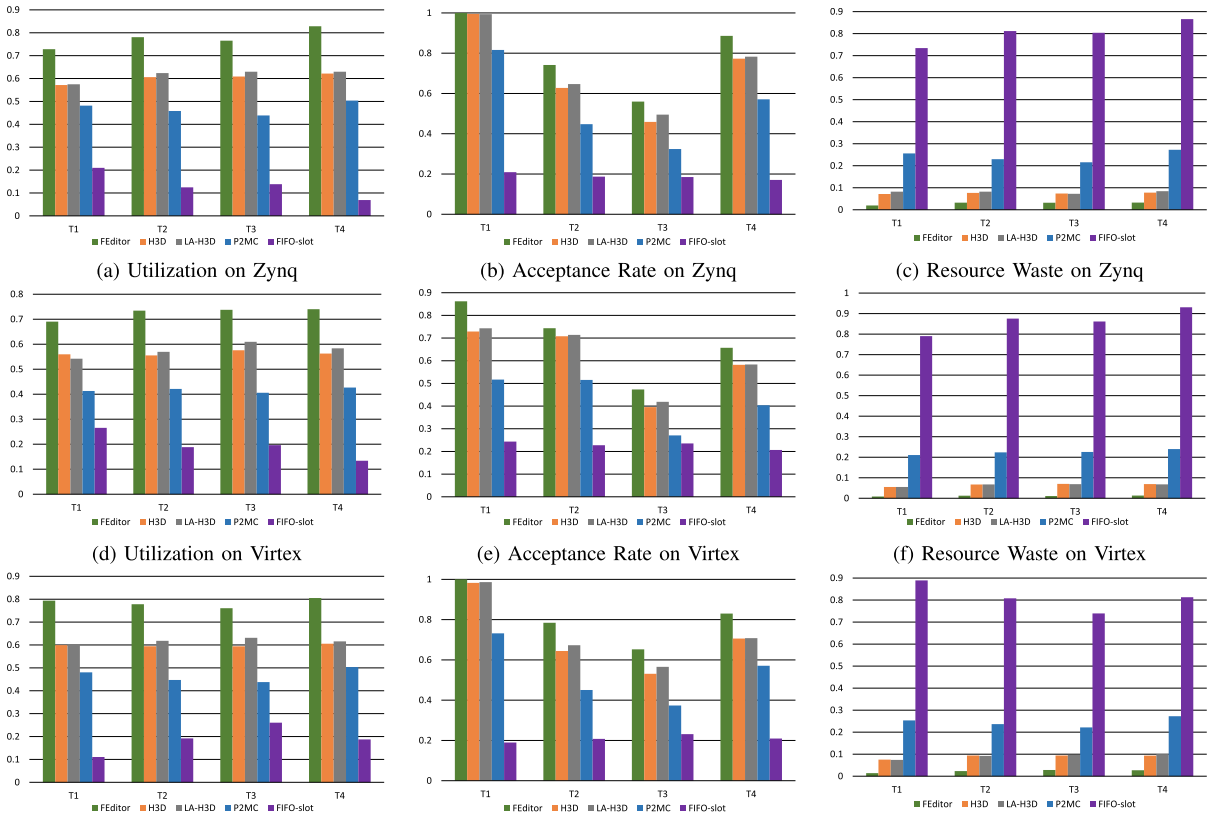
B. Performance

Three rates, including utilization, acceptance rate, and resource waste, are calculated to evaluate the performance of FEditor and other algorithms as shown in Fig. 13.

Utilization rate denotes how many resources are effectively used. It is the sum of *SFs* weighted by their durations. Acceptance rate indicates how many tasks in the task set can be successfully placed. Resource waste denotes the degree of internal fragmentation. Therefore, we define an optimal algorithm as the one with the highest utilization rate and acceptance rates as well as the lowest resource waste rate.

FEditor and other algorithms all reach relatively stable levels of utilization and resource waste across all types of task sets. However, their acceptance rates decrease as the resource requirements increase. This is because the number of tasks in different types of task sets is identical. Larger tasks occupy more resources over longer periods. As a result, FEditor and other algorithms must reject tasks due to resource shortages.

The experimental results demonstrate that FEditor significantly outperforms existing algorithms across the three rates. Compared with H3D and LA-H3D, FEditor constitutes an enhanced evolution of the LA-H3D algorithm, addressing its inherent limitations. While LA-H3D employs a one-step look-ahead strategy constrained by computational complexity, FEditor generates placement strategies across the entire life-cycle of each task. This comprehensive approach yields substantial performance gains. On the T4-Zynq suite, FEditor achieves a 19.8% increase in resource utilization and a 10% improvement in task acceptance rate compared to LA-H3D. In terms of optimization extent relative to H3D, FEditor achieves $25.6\times$ and $7.6\times$ higher optimization on T4-Zynq and T4-Virtex suites respectively.



(a) Utilization on Zynq (b) Acceptance Rate on Zynq (c) Resource Waste on Zynq

(d) Utilization on Virtex (e) Acceptance Rate on Virtex (f) Resource Waste on Virtex

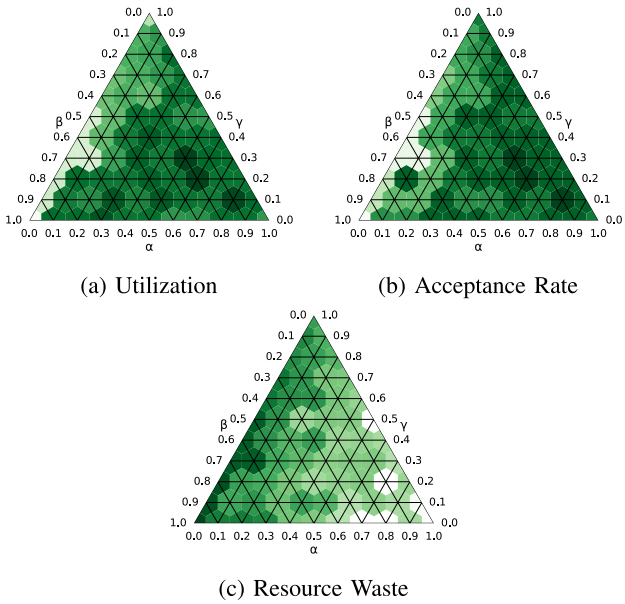
(g) Utilization on Cyclone (h) Acceptance Rate on Cyclone (i) Resource Waste on Cyclone

Fig. 13. Performance of Algorithms on Zynq, Virtex and Cyclone FPGAs.

Furthermore, FEditor reduces relative resource waste by 60% and 80% compared to LA-H3D on both suites. As for the T4-Cyclone suite, FEditor achieves an increase of 18.9% and 12.2% on resource utilization and acceptance rates. Also, resource waste is reduced by 7.3% compared with LA-H3D.

When contrasted with the P2MC, FEditor exhibits even more pronounced advantages. FEditor reduces relative resource waste by 88% and 94.5% relative to P2MC on T4-Zynq and T4-Virtex suites, while simultaneously achieving 32.5% and 33.2% higher utilization rates respectively. And FEditor increases overall task acceptance rate by around 30% on the T4 task set. Moreover, in Altera's FPGAs, FEditor achieves 59.8%, 45.4%, and 90% relative optimization in three aspects for the T4 task set. Notably, even though P2MC is enhanced by the lazy placement strategy, its performance remains constrained by the inability to address straggler tasks in one placed group.

As shown in Fig. 13, the slot-based placement algorithm, FIFO-slot, causes significant resource waste and task rejection by forcing all accepted tasks to occupy an entire slot. As small tasks incur fragmentation and resource waste, large ones cannot use resources across slots, which increases the task rejection rate.



(a) Utilization (b) Acceptance Rate

(c) Resource Waste

Fig. 14. Rate values based on different hyperparameters.

C. Metrics Effectiveness

Different ratios of α, β and γ lead to different performance. To evaluate their effectiveness, we adjust their values and test FEditor on T4-Zynq. The results are presented as ternary phase diagrams in Fig. 14. Each point in this figure represents a combination, which can be interpreted by drawing parallel lines to the axes. We shade the area around each point with one rate

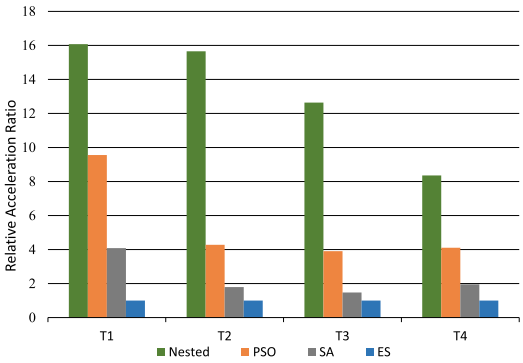


Fig. 15. The relative acceleration rates of different heuristic algorithms.

value defined above. We can identify the optimal combination from the deepest shaded area in Fig. 14(a) and (b), while the lightest is optimal in Fig. 14(c).

The figures demonstrate that the optimal combination consistently lies within the interior of the triangles rather than along their edges. This geometric observation strongly suggests that all three metrics—internal fragmentation, external fragmentation, and placement compactness—contribute positively to the overall performance. The absence of edge dominance indicates a collaborative relationship among the metrics, where each contributes unique value that cannot be fully compensated by the others. It is important to note that while pure α-maximization (with $\beta = \gamma = 0$) would theoretically minimize internal fragmentation, the extreme weight configurations cannot reach the overall optimum.

Further analysis of parametric optimization under fixed weights reveals a symmetry in the optimal configuration. When α is held constant, the optimal solutions cluster around the point where β equals γ. This symmetry arises from the complementary nature of external fragmentation control and placement compactness optimization. Metric on external fragmentation remains in rectangular idle spaces, while placement compactness prevents fragmented hollows within these spaces. When fixing β or γ, experimental results show that the optimal α:(β/γ) ratio stabilizes near 3:1 across different task sets. The darkest-shaded area in the figures corresponds to this combination (0.6,0.2,0.2), which proves the 3:1:1 regularity.

The aforementioned findings highlight the importance of the collaborative relationship among the three metrics. Scattered values in the results arise from limitations in the task sets.

D. Nested Algorithm Acceleration

To evaluate proposed nested heuristic algorithm, we compare it with Exhaustive Search (ES), Simulated Annealing (SA), and Particle Swarm Optimization (PSO). For fairness, we develop their parallel versions for comparison. We measure their relative time consumption on the Zynq FPGA model, as shown in Fig. 15. We fine-tune their hyperparameters to achieve results comparable to the nested algorithm. Compared with other heuristic algorithms, the nested algorithm accelerates the search by up to 16× and 13× on average, while SA achieves 4× and 2.3× speedups and PSO obtains 9.5× and 5.5× speedups. This is because the nested heuristic algorithm splits the search space and explores multiple sub-spaces in parallel. In contrast, PSO and SA only accelerate computations. SA maintains a stable acceleration rate because it traverses all temperature levels in its annealing schedule.

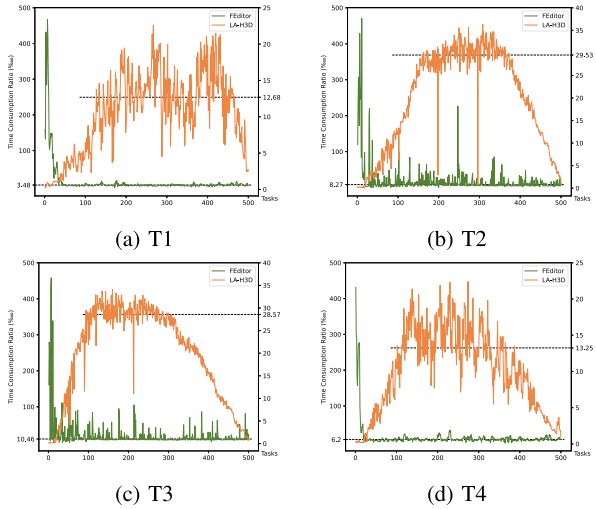


Fig. 16. Trends of time efficiency for task sets on Zynq.

E. Trends of Time Efficiency

FEditor uses the stacked *SF* to suggest placement strategies. As tasks are consecutively placed onto the FPGA, many candidate placements overlap with existing ones. When checking the legality of candidates, a direction to idle spaces in a *SF* is returned if the candidate is illegal. FEditor modifies its solutions by following the given direction. Based on this mechanism, FEditor can decide whether to accept or reject a task rapidly. As more tasks keep arriving, its advantage will become even more obvious.

LA-H3D uses vertex links of multiple idle spaces to explore the solution space. It generates candidate solutions with minimal internal fragmentation for each vertex. After traversing all vertices, LA-H3D selects the solution with the highest 3D adjacency score as the final placement. Unlike FEditor, vertex links management becomes complicated with arrivals of tasks, as the number of vertices becomes huge.

As shown in Fig. 16, we can clearly observe the time consumption ratio of each task. The x-axis represents the task sequence, and the two y-axes indicate the proportion of each task's time consumption to the overall time. The left one corresponds to FEditor, and the right one corresponds to LA-H3D. The horizontal dashed line denotes the most concentrated ratios of FEditor and LA-H3D.

FEditor experiences a cold-start period and then maintains its highest time efficiency. This is because FEditor gradually fills up the FPGA, reducing the number of idle regions and minimizing the valid search space. The concentrated time ratio maintains a low value, around 8 textpertenthousand.

The curve of LA-H3D presents a mountain-like shape. As tasks accumulate, the time consumption gradually rises and

TABLE IV
REAL TIME CONSUMPTION (MS) ON ZYNQ

Algorithm	T1	T2	T3	T4
FEditor	2558.55	4730.89	3812.56	4881.68
LA-H3D	5324.26	14733.52	15966.29	12077.68
Relative	2.08	3.11	**4.19**	2.47

TABLE V
REAL TIME CONSUMPTION (MS) ON VIRTEX

Algorithm	T1	T2	T3	T4
FEditor	19409.61	35424.75	22027.08	11835.48
LA-H3D	19858.13	42730.30	41121.90	27545.67
Relative	1.02	1.21	1.87	**2.33**

TABLE VI
REAL TIME CONSUMPTION (MS) ON CYCLONE

Algorithm	T1	T2	T3	T4
FEditor	16795.29	20033.95	22842.89	19539.58
LA-H3D	91023.25	205538.6	205652.6	178109.7
Relative	5.42	**10.26**	9.00	9.12

stabilizes. As tasks are drained, the time consumption gradually decreases. This is because the number of vertex links is smaller during the filling phase. After repeated placement and release operations, vertex links become increasingly complex and numerous. Ratios in the stable phase are highest and most dense, around 20 textpertenthousand. As shown in Fig. 13(b), most rejections by LA-H3D occur during the stable phase. Therefore, the arrival rate during the draining phase is lower than the removal rate, which simplifies the vertex link structure. Task placement in this phase thus consumes less time.

Since tasks in datacenters arrive consecutively, FEditor can generate placements for arriving tasks efficiently when FPGAs are under heavy load, whereas LA-H3D tends to reject them. This trend makes FEditor suitable for datacenter environments while renders LA-H3D unsuitable.

The real-time consumption results of FEditor and LA-H3D are shown in Tables IV, V, and VI. The data in tables are the summary of decision-making, not including structure maintenance. FEditor runs faster than LA-H3D by achieving a speedup of up to $10.26\times$. From the time complexity analysis, we can get the complexity of FEditor is $O(iter \cdot (\log T_0 - \log T)/\log \alpha \cdot n'^2)$. Since LA-H3D needs to traverse the vertex links to search for the optimal region, the complexity is $O(mn)$, while parameters m and n are length and width of the original board. Due to the effective parallelization and space splitting, FEditor runs faster than LA-H3D.

VIII. CONCLUSIONS AND FUTURE WORK

This paper proposes FEditor to address the placement problem for consecutive tasks with adjustable shapes on FPGAs with multiple resource types. *SFs* are introduced to simplify the resource management and transform the 3D problem into finite 2D sub-problems. Based on this approach, we introduce three complementary metrics. The problem is formally formulated and proved to be NP-hard by reduction from 3DBP, followed by the design of a nested heuristic algorithm. Extensive experimental evaluations demonstrate that FEditor significantly outperforms the most advanced methods. However, FEditor relies on the accuracy of the compilation time predictor. We plan to optimize the predictor and implement FEditor on a real FPGA board in the future.

REFERENCES

[1] AWS, "Amazon EC2 F2 instances." Accessed: Oct. 2025. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f2/

[2] Aliyun, "New generation FPGA instances." Accessed: Oct. 2025. [Online]. Available: https://promotion.aliyun.com/ntms/act/fpgaf3.html

[3] Azure, "FPGA attestation for Azure NP-Series VMs." 2024. Accessed: Oct. 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/field-programmable-gate-arrays-attestation

[4] AMD, "AMD virtex UltraScale+ FPGAs." 2025. Accessed: Oct. 2025. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus.html

[5] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Des. Test Comput.*, vol. 17, no. 1, pp. 68–83, Jan.–Mar. 2000.

[6] C. Bobda et al., "The future of FPGA acceleration in datacenters and the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15 no. 3, pp. 1–42, Feb. 2022.

[7] A. Boutros and V. Betz, "FPGA architecture: Principles and progression," *IEEE Circuits Syst. Mag.*, vol. 21, no. 2, pp. 4–29, Second Quarter 2021.

[8] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu, "B*-Trees: A new representation for non-slicing floorplans," in *Proc. 37th Des. Automat. Conf.*, 2000, pp. 458–463.

[9] E. Chung et al., "Serving DNNs in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar./Apr. 2018.

[10] G. Dai, Y. Chi, Y. Wang, and H. Yang, "FPGP: Graph processing framework on FPGA a case study of breadth-first search," in *Proc. 2016 ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 105–110.

[11] A. Dhar et al., "DML: Dynamic partial reconfiguration with scalable task scheduling for multi-applications on FPGAs," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2577–2591, Oct. 2022.

[12] A. Eiche, D. Chillet, S. Pillement, and O. Sentieys, "Task placement for dynamic and partial reconfigurable architecture," in *Proc. 2010 Conf. Des. Architectures Signal Image Process.*, 2010, pp. 228–234.

[13] S. Fekete, B. Fiethe, S. Friedrichs, H. Michalik, and C. Orlis, "Efficient reconfiguration of processing modules on FPGAs for space instruments," in *Proc. 2014 NASA/ESA Conf. Adaptive Hardware Syst.*, 2014, pp. 15–22.

[14] Z. Guettatfi, P. Kaufmann, and M. Platzner, "Optimal and greedy heuristic approaches for scheduling and mapping of hardware tasks to reconfigurable computing devices," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Berlin, Germany: Springer, 2020, pp. 108–117.

[15] X. Iturbe et al., "R3TOS: A novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on FPGAs," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1542–1556, Aug. 2013.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[17] Q. H. Khuat, D. Chillet, and M. Hübner, "Considering reconfiguration overhead in scheduling of dependent tasks on 2D reconfigurable FPGA," in *Proc. 2014 NASA/ESA Conf. Adaptive Hardware Syst.*, 2014, pp. 1–8.

[18] M. Koester, M. Porrmann, and H. Kalte, "Task placement for heterogeneous reconfigurable architectures," in *Proc. 2005 IEEE Int. Conf. Field-Program. Technol.*, 2005, pp. 43–50.

[19] D. Korolija, T. Roscoe, and G. Alonso, "Do OS abstractions make sense on FPGAs?," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 991–1010.

[20] Q. H. Le, E. Casseau, and A. Courtay, "Place reservation technique for online task placement on a multi-context heterogeneous reconfigurable architecture," in *Proc. 2014 Int. Conf. Reconfigurable Comput. FPGAs*, 2014, pp. 1–6.

[21] K. Li and K. H. Cheng, "Generalized first-fit algorithms in two and three dimensions," *Int. J. Found. Comput. Sci.*, vol. 1, no. 02, pp. 131–150, 1990.
[22] K. Li and K. H. Cheng, "On three-dimensional packing," *SIAM J. Comput.*, vol. 19, no. 5, pp. 847–867, 1990.
[23] K. Li and K. H. Cheng, "Static job scheduling in partitionable mesh connected systems," *J. Parallel Distrib. Comput.*, vol. 10, no. 2, pp. 152–159, 1990.
[24] K. Li and K. H. Cheng, "Job scheduling in a partitionable mesh using a two-dimensional buddy system partitioning scheme," *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, no. 4, pp. 413–422, Oct. 1991.
[25] K. Li and K. H. Cheng, "A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system," *J. Parallel Distrib. Comput.*, vol. 12, no. 1, pp. 79–83, 1991.
[26] K. Li and K. H. Cheng, "Heuristic algorithms for on-line packing in three dimensions," *J. Algorithms*, vol. 13, no. 4, pp. 589–605, 1992.
[27] H. Liao et al, "TurboHE: Accelerating fully homomorphic encryption using FPGA clusters," in *Proc. 2023 IEEE Int. Parallel Distrib. Process. Symp.*, 2023, pp. 788–797.
[28] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.
[29] X. Peng et al, "Capuchin:Tensor-based GPU memory management for deep learning," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 891–905.
[30] J. Tabero, J. Septién, H. Mecha, and D. Mozos, "Task placement heuristic based on 3D-adjacency and look-ahead in reconfigurable systems," in *Proc. 2006 Asia South Pacific Des. Automat. Conf.*, 2006, pp. 396–401.
[31] B. L. Tan, K. M. Mok, J. J. Chang, W. K. Lee, and S. O. Hwang, "RISC32-LP: Low-power FPGA-based IoT sensor nodes with energy reduction program analyzer," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4214–4228, Mar. 2022.
[32] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: Free space partitioning and 2D-hashing," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003.
[33] G. Wang, S. Liu, J. Nie, F. Wang, and T. Arslan, "An online task placement algorithm based on maximum empty rectangles in dynamic partial reconfigurable systems," in *Proc. 2017 NASA/ESA Conf. Adaptive Hardware Syst.*, 2017, pp. 180–185.
[34] J. Whangbo et al., "FireAxe: Partitioned FPGA-accelerated simulation of large-scale RTL designs," in *Proc. ACM/IEEE 51st Annu. Int. Symp. Comput. Archit.*, 2024, pp. 501–515.
[35] Y. Xiao, D. Park, Z. J. Niu, A. Hota, and A. Dehon, "ExHiPR: Extended high-level partial reconfiguration for fast incremental FPGA compilation," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, no. 2, pp. 1–28, 2024.
[36] R. Yao, Y. Zhao, Y. Yu, Y. Zhao, and X. Zhong, "Fast search and efficient placement algorithm for reconfigurable tasks on modern heterogeneous FPGAs," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 30, no. 4, pp. 474–487, Apr. 2022.
[37] P. H. Yuh, C. L. Yang, and Y. W. Chang, "Temporal floorplanning using the T-tree formulation," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, IEEE, 2004, pp. 300–305.
[38] S. Zeng et al., "DF-GAS: A distributed FPGA-as-a-service architecture towards billion-scale graph-based approximate nearest neighbor search," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2023, pp. 283–296.
[39] Y. Zha and J. Li, "Virtualizing FPGAs in the cloud," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 845–858.
[40] J. Zhang et al., "WIC: Hiding producer-consumer synchronization delays with warp-level interrupt-based GPU communications," in *Proc. 2025 USENIX Annu. Tech. Conf.*, 2025, pp. 889–904.
[41] W. Zhang, J. Zhao, G. Shen, Q. Chen, C. Chen, and M. Guo, "An optimizing framework on MLIR for efficient FPGA-based accelerator generation," in *Proc. 2024 IEEE Int. Symp. High-Perform. Comput. Archit.*, 2024, pp. 75–90.
[42] Z. Zhu, A. X. Liu, F. Zhang, and F. Chen, "FPGA resource pooling in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 610–626, Apr.–Jun. 2021.

Yanyan Li received the BS degree in computer science and technology in 2024 from the Beijing University of Posts and Telecommunications, Beijing, China, where he is currently working toward the PhD degree with the State Key Laboratory of Networking and Switching Technology. His research interests include distributed software in datacenters, and computer theory.

Yu Chen received the BS degree in computer science and technology in 2023 from the Beijing University of Posts and Telecommunications, Beijing, China, where she is currently working toward the master's degree with the State Key Laboratory of Networking and Switching Technology. Her research interests include distributed software in FPGA resource management and scheduling.

Zhiqian Xu received the master's degree in computer science from Wayne State University, USA, in 1995, and the PhD degree in information security from Royal Holloway, University of London, London, U.K., in 2019. She is currently a research associate with Beijing University of Posts and Telecommunications, Beijing, China. Her research interests include system security, high performance computing, and quantum computing.

Yawen Wang received the PhD degree in communication and information systems from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2010. She is currently an associate professor with the State Key Laboratory of Networking and Switching Technology, BUPT. Her research interests include high-performance computing, compilation systems, program analysis, and software testing. She is also a member of the China Computer Federation.

Hai Jiang (Member, IEEE) received the master's and PhD degrees in computer science from Wayne State University, USA, in 1995 and 2003, respectively. He is currently a professor with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing, China. His research interests include high performance computing, system security, and quantum computing. He is a member of ACM and IEEE Computer Society.

Keqin Li (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, Beijing, China, in 1985, and the PhD degree in computer science from the University of Houston, Houston, TX, USA, in 1990. He is also a SUNY distinguished professor with the State University of New York, USA, and a National Distinguished professor with Hunan University, China. He has authored or coauthored more than 1130 journal articles, book chapters, and refereed conference papers. He also holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked 2) and career-long impact (ranked 4) based on a composite indicator of the Scopus citation database. He was the recipient of the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022, the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023, IEEE Region 1 Technological Innovation Award (Academic) in 2023, the 2022–2023 International Science and Technology Cooperation Award, and the 2023 Xiaoxiang Friendship Award of Hunan Province, China. From 2023 to 2024, he was listed in Scilit Top Cited Scholars and is also among the top 0.02% out of more than 20 million scholars worldwide based on top-cited publications. From 2022 to 2024, he was listed in ScholarGPS Highly Ranked Scholars and is among the top 0.002% out of more than 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He is also a member of the European Academy of Sciences and Arts, Academia Europaea (Academician of the Academy of Europe), and SUNY Distinguished Academy. He is also fellow of AAAS, AAIA, ACIS, and AIIA.