Lagrangian Relaxation-Based Time-Division Multiplexing Optimization for Multi-FPGA Systems

CHAK-WA PUI and EVANGELINE F. Y. YOUNG, The Chinese University of Hong Kong, Hong Kong

To increase the resource utilization in multi-FPGA (field-programmable gate array) systems, time-division multiplexing (TDM) is a widely used technique to accommodate a large number of inter-FPGA signals. However, with this technique, the delay imposed by the inter-FPGA connections becomes significant. Previous research has shown that the TDM ratios of signals can greatly affect the performance of a system. In this article, to minimize the system clock period and support more practical constraints in modern multi-FPGA systems, we propose an analytical framework to optimize the TDM ratios of inter-FPGA nets. A Lagrangian relaxation-based method first gives a continuous result under relaxed constraints. A binary search–based discretization algorithm is then used to assign the TDM ratio of each net such that the resulting maximum displacement is optimal and all the constraints are satisfied. Finally, a swapping-based post refinement is performed to further optimize the TDM ratios. For comparison, we also solve the problem using linear programming (LP)–based methods, which have guaranteed error bounds to the optimal solutions. Experimental results show that our framework can achieve similar quality with much shorter runtime compared to the LP-based methods. Moreover, our framework scales for designs with over 45,000 inter-FPGA nets while the runtime and memory usage of the LP-based methods will increase dramatically as the design scale becomes larger.

CCS Concepts: • **Hardware → Wire routing**; **Programmable interconnect**;

Additional Key Words and Phrases: EDA, time-division multiplexing, FPGA, routing, Lagrangian relaxation

ACM Reference format:
Chak-Wa Pui and Evangeline F. Y. Young. 2020. Lagrangian Relaxation-Based Time-Division Multiplexing Optimization for Multi-FPGA Systems. *ACM Trans. Des. Autom. Electron. Syst.* 25, 2, Article 21 (January 2020), 23 pages.
https://doi.org/10.1145/3377551

1 INTRODUCTION

In recent years, field-programmable gate array (FPGA) has become very popular in various fields including deep learning [2] and data centers [3]. Although the scale of FPGAs has greatly increased, it is still unlikely to fit the entire design into one FPGA in applications such as logic emulation and rapid prototyping of large designs [4]. Hence, multi-FPGA systems are usually used. A multi-FPGA

The work was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project CUHK14202218. The preliminary version has been presented at the International Conference of Computer Aided Design (ICCAD) in 2019 [1].
Authors' addresses: C.-W. Pui (corresponding author) and E. F. Y. Young, RM913, Ho Sin-Hang Engineering building, the Chinese University of Hong Kong; emails: {cwpui, fyyoung}@cse.cuhk.edu.hk.

1084-4309/2020/01-ART21 $15.00
https://doi.org/10.1145/3377551

system consists of multiple FPGAs which are connected using direct hardwired connections or a programmable interconnection network [5].

In multi-FPGA systems, the available pin count of each FPGA is relatively small compared to the number of inter-FPGA signals, which significantly limits the utilization of the logic resources. Time-division multiplexing (TDM) is a method that multiplexes the use of FPGA pins and inter-FPGA physical wires among multiple inter-FPGA signals [6]. Although the number of logical pins available in each FPGA can be effectively increased, this technique makes the system clock period longer since the inter-FPGA signal delay is lengthened due to time-multiplexing. There are various methods to reduce the negative effect of time-multiplexing on delay during compilation such as minimizing the number of inter-FPGA signals. TDM optimization is still an important step in the compilation flow since different TDM ratios can result in very different system clock periods [7].

In the compilation flow of multi-FPGA systems, TDM ratios are usually determined after inter-FPGA routing [6]. Several methods are proposed to optimize the TDM ratios in recent works. In [8], an integer linear programming (ILP)–based method for 2-FPGA systems is introduced. It tries to put non-critical inter-FPGA nets in TDM wires to improve the utilization without affecting the timing of the systems, which results in a 0-1 decision problem for each net. The works [9–11] extend [8] to support multi-FPGA systems with partially connected FPGAs. Due to the increasing number of inter-FPGA signals in multi-FPGA systems, ILP-based methods are no longer able to produce high-quality results with scalable running time. A recent work [12] proposes a framework that performs TDM assignment and partitioning simultaneously. For signal grouping, a binary search–based method is used to enumerate all the possible groupings. However, the cost they optimize is not the system clock period but an estimated timing criticality metric. The work [13] proposes an analytical framework that performs TDM optimization with minimized system clock ratio. But the continuous solver is slow for convex problems and the discretization method does not scale well for big designs. Moreover, the TDM ratios can be arbitrary integers in previous works except the work [13], which is not true in practice [7]. Therefore, an efficient and effective TDM optimization algorithm for modern multi-FPGA systems is needed.

Lagrangian relaxation is a widely used method in convex optimization due to its effectiveness and efficiency. In physical designs, there are a lot of previous works that apply Lagrangian relaxation in their problems. The works [14, 15] use Lagrangian relaxation to optimize area and wirelength during floorplanning. The works [16–18] solve the gate sizing problem with different Lagrangian relaxation approaches, where circuit delay and area are minimized. In [19], Lagrangian relaxation is applied in timing-driven global placement to improve circuit performance. Lagrangian relaxation can also be applied in our TDM optimization problem since it can be relaxed to a convex and continuous formulation.

In this work, we propose an analytical framework for the TDM optimization problem. The major contributions are summarized as follows:

- An analytical framework is proposed to optimize the system clock period globally in the multi-FPGA TDM optimization problem. It first generates a continuous result of the TDM ratios with minimized system clock period under relaxed constraints. A discretization algorithm is then applied to remove all the constraint violations. Finally, a swapping-based post refinement is performed to further optimize the TDM ratios. Compared to previous works, our approach supports wires with a user given set of TDM ratios, which is more practical in modern multi-FPGA systems.
- A Lagrangian relaxation-based approach is proposed to solve the continuous TDM optimization problem which is effective and efficient. A novel method is proposed to initialize and update the multipliers such that the solver converges faster. In particular, our

Table 1. Notations

T	The set of all FPGA-pairs.
t_i	The set of inter-FPGA nets of the FPGA-pair i in T.
p_i	Wire limit of the FPGA-pair i in T.
G	Timing graph.
$g_{\text{src}}, g_{\text{sink}}$	Source and sink of G.
g_i	Node i in G representing gate i.
$e_{p,q}$	Edge from g_p to g_q in G.
E_i	The set of edges correspond to inter-FPGA net net_i.
e_i	The i^{th} edge in G.
$delay_{p,q}$	Delay of $e_{p,q}$.
at_p	Arrival time of g_p.
$at_{p,q}$	Arrival time along edge $e_{p,q}$ and $at_{p,q} = at_p + delay_{p,q}$.
x_i	TDM ratio of inter-FPGA net net_i.
$x_{p,q}$	TDM ratio of $e_{p,q}$ and $x_i = x_{p,q}$ for all $e_{p,q} \in E_i$.
$b_{p,q}, c_{p,q}$	Architecture related parameters of $e_{p,q}$.
$e^c_{p,q}$ $(e^x_{p,q})$	An edge from g_p to g_q representing an intra-FPGA (inter-FPGA) net.

method not only captures the characteristics of the timing optimization but also considers the Karush-Kuhn-Tucker (KKT) conditions during update.

- An algorithm is proposed to discretize the continuous TDM result which will produce a legal solution with the smallest maximum displacement. A binary search–based method is used to find the minimum feasible maximum displacement. Given the bounded TDM discrete choices, a dynamic programming (DP)–based approach is then used to produce a legal solution with the optimal total displacement under the minimum feasible maximum displacement constraint.
- A post refinement algorithm is proposed to further optimize the results of discretization, which will swap the TDM ratios between critical and non-critical nets.
- Several linear programming (LP)–based methods are proposed as the baselines of our algorithm. A technique is proposed to improve the error rate of our LP methods to about 4% of the optimal solution of the continuous formulation, which further validates the quality of our proposed method.

The remainder of this article is organized as follows. Section 2 gives the preliminaries of the problem. Sections 3–6 first give an overview of our approach and then introduce its details. Section 7 shows the experimental results, and we finally conclude the article in Section 8.

2 PRELIMINARIES

In this section, we first explain our target architecture and its compilation flow. The problem definition will then be introduced. The notations used in the rest of this article are shown in Table 1. Note that, throughout this article, *tdmNet* refers to inter-FPGA net for simplicity.

2.1 Target Architecture

We consider a multi-FPGA system with time-multiplexed hardwired inter-FPGA connections where two FPGAs are adjacent logically if they are directly connected in the system. In our target

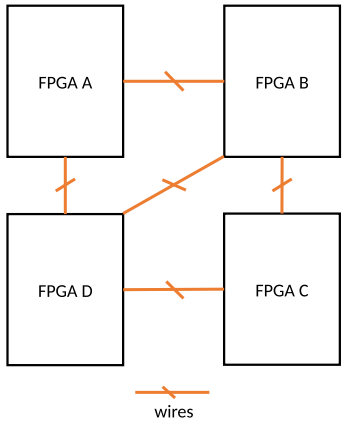


Fig. 1. An example of how FPGAs are connected to each other. Here B is adjacent to C,D while A,B are not adjacent to each other.

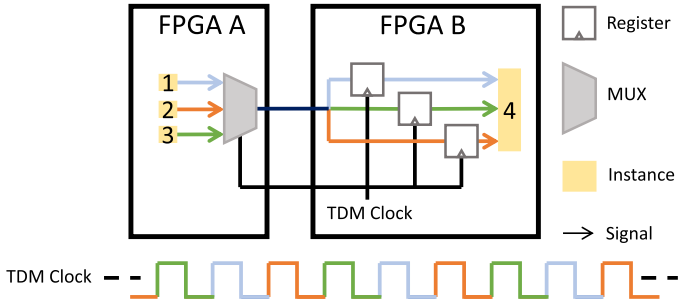


Fig. 2. An example of inter-FPGA time-multiplexed I/Os where three different signals share one wire.

system, two adjacent FPGAs are called an FPGA-pair. Figure 1 gives an example of how FPGAs are connected in such systems. Although only connections between FPGA-pairs are considered in this work, it can be easily extended to architectures where FPGAs can communicate with each other through multiple intermediate hops.

Since the number of tdmNets is much larger than the number of physical wires between FPGAs, time-multiplexed wires are usually used to connect different FPGAs. In such systems, only the tdmNets in the same direction and with the same TDM ratio can be assigned to the same wire. Moreover, there should be no more than n tdmNets assigned to a wire with TDM ratio n. Figure 2 is an example of the inter-FPGA time-multiplexed I/Os in our target architecture. It is implemented with a $3:1$ multiplexer (MUX) placed in the same device as the three signal sources and the same amount of registers placed in the other device. The MUX inputs are selected by a TDM clock, and the MUX input signals are transmitted to FPGA B sequentially. The registers are used to store values of the signals passing the MUX. For each signal going through the routing channel, its value first arrives at the MUX input and will wait until the MUX is open for it to go through. After going through the channel, it is latched in a register which becomes the pseudo source of the signal. When estimating the delay of a signal in our target architecture, I/O TDM implementation assumes the worst case scenario, which happens when the signal needs to wait for an entire TDM cycle to reach its turn. Hence, the transmission delay is a monotonically increasing function of the TDM ratio of its corresponding wire. Given a tdmNet net_i going through a wire with TDM ratio x_i, the delay of edge $e_{p,q}$ can be calculated as $(b_{p,q} \cdot x_i + c_{p,q})$ for all $e_{p,q} \in E_i$, where $b_{p,q}, c_{p,q}$

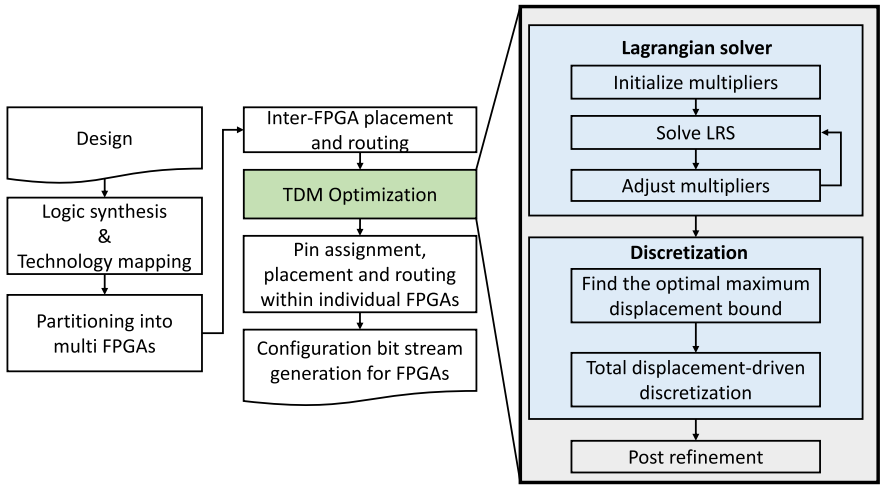


Fig. 3. A typical compilation flow for multi-FPGA systems and the overall flow of our approach.

are architecture related parameters corresponding to $e_{p,q}$. Due to the architecture limitations, the TDM ratio can only be 1 or multiples of 8 instead of an arbitrary integer [7]. And we limit the maximum value of TDM ratio to be 1,600 in this article, which is a parameter that can be changed.

2.2 Compilation Flow

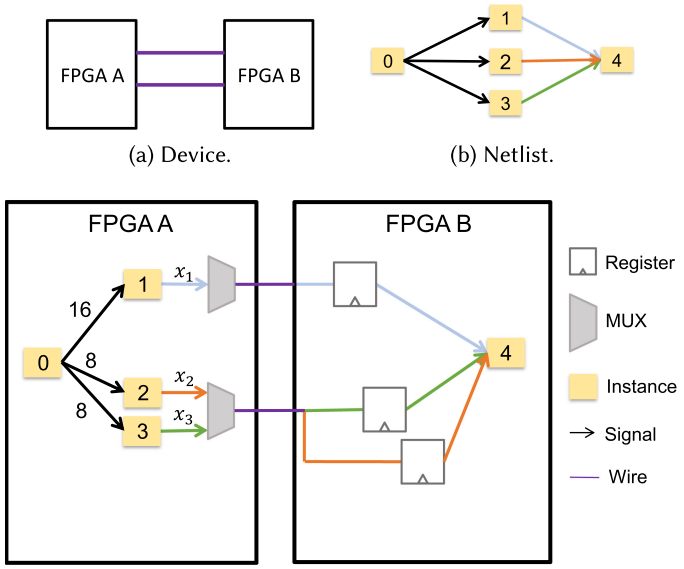
A typical compilation flow for multi-FPGA systems [20] is shown in Figure 3. The first step of the compilation flow is logic synthesis and technology mapping where the given circuit is mapped into a netlist of primitive elements such as lookup tables (LUTs), flip-flops (FFs), RAMs, DSPs, and so forth. The netlist is then divided into partitions such that each partition can fit into a single FPGA and the number of inter-FPGA connections is minimized. Inter-FPGA placement puts each partition into a distinct FPGA on the board. Inter-FPGA routing is then performed which considers both system performance and routing resources. It determines the routing topology and the TDM ratio for each tdmNet. It must ensure that the number of tdmNets between any two adjacent FPGAs will not require more physical wires than available under their TDM specification. Given the routing result, TDM optimization is applied to optimize the TDM ratios regarding the system clock period. After that, pin assignment will choose the physical wire and pins for each tdmNet subject to the TDM constraints from the previous steps. Finally, the placement and routing of individual FPGAs are performed.

2.3 Problem Definition

Definition 1. The **TDM ratio** of a tdmNet represents the maximum number of signals that it can share a physical wire with.

Definition 2. For any two tdmNets $net_a, net_b \in t_A$, they are in the same **direction** if their sources are from the same FPGA.

In the TDM optimization problem, given the system architecture and the timing graph constructed from the inter-FPGA routing result, we need to determine the TDM ratio for each tdmNet. The objective is to minimize the system clock period, which is the arrival time at the sink in the timing graph. The TDM ratios of the tdmNets between the FPGA-pair i should satisfy the following constraints.



(a) Device. (b) Netlist.

(c) The time-multiplexed I/Os when $\{x_1, x_2, x_3\} = \{1, 8, 8\}$.

Fig. 4. An example of TDM optimization problem.

- Each TDM ratio should be either 1 or multiples of 8.
- Only tdmNets with the same TDM ratio and in the same direction can be assigned to the same wire.
- The number of tdmNets in a wire with TDM ratio n should not exceed n.
- The total number of wires used cannot exceed the wire limit p_i.

An example of the TDM optimization problem is shown in Figure 4 where all $c_{p,q}$ and $b_{p,q}$ are assumed to be 0 and 1, respectively. There are two wires between the FPGA-pair and three tdmNets from FPGA A to B. Let the TDM ratios for these three tdmNets be x_1, x_2, x_3 and the arrival times of their driving pins are $16, 8, 8$, respectively. The system clock period is the maximum arrival time at the primary outputs at FPGA B which is $at_{\text{sink}} = \max(16 + x_1, 8 + x_2, 8 + x_3)$. The optimal assignment of the TDM ratios of this system will be $(x_1, x_2, x_3) = (1, 8, 8)$ in which the tdmNets net_2 and net_3 can share one physical wire.

3 TDM OPTIMIZATION FRAMEWORK

The flow of our TDM optimization framework is shown in Figure 3. Given an inter-FPGA routing result, the corresponding timing graph is first constructed. Our continuous solver is then built based on the timing graph, which minimizes the system clock period under relaxed TDM constraints. To be specific, each TDM ratio is within the range $[1, 1{,}600]$ and $\sum_{j \in t_i} \frac{1}{x_j} \leq p_i$ holds for every FPGA-pair i. After that, we can get a continuous result of the TDM ratios with optimized system clock period. Our discretization algorithm is then performed such that all the TDM ratio violations are removed while the continuous result is honored. Finally, a post refinement algorithm is applied to further optimize the system clock period without inducing new constraint violation. Details of our continuous solver, discretization algorithm, and post refinement method will be discussed in Sections 4, 5, and 6, respectively.

4 CONTINUOUS SOLVER

To simplify the notations in this section, we assume (1) $b_{p,q} = 1$ and $c_{p,q} = 0$ for all inter-FPGA edges $e^x_{p,q}$ and (2) $delay_{p,q} = 0$ for all intra-FPGA edges $e^c_{p,q}$. However, these parameters are not restricted to such assumptions during experiments. As discussed in the previous section, our continuous solver will solve the TDM optimization problem under relaxed constraints. In the original problem shown in Equation (1), at_{sink} is a function of the TDM ratios and arrival time.

$$\begin{aligned}
\min \quad & at_{sink}, && \text{(1a)}\\
s.t. \quad & \sum_{j \in t_i} \frac{1}{x_j} \le p_i, \quad \forall i \in T, && \text{(1b)}\\
& at_q = \max(\max_{e^x_{p,q} \in E_i} (at_p + x_i), \max_{e^c_{p,q}} (at_p)), \quad \forall g_q \in (G \setminus g_{src}), && \text{(1c)}\\
& at_{src} = 0, && \text{(1d)}\\
& 1 \le x_i \le 1{,}600, \quad \forall x_i. && \text{(1e)}
\end{aligned}$$

However, in our solver, the function is equivalently transformed into a set of constraints, which is shown in Equation (2).

$$\begin{aligned}
\mathcal{PP}: \quad \min \quad & at_{sink}, && \text{(2a)}\\
s.t. \quad & \sum_{j \in t_i} \frac{1}{x_j} \le p_i, \quad \forall i \in T, && \text{(2b)}\\
& at_q \ge at_p + x_i, \quad \forall e^x_{p,q} \in E_i, && \text{(2c)}\\
& at_q \ge at_p, \quad \forall e^c_{p,q}, && \text{(2d)}\\
& at_q \ge 0, \quad \forall e^c_{src,q}, && \text{(2e)}\\
& 1 \le x_i \le 1{,}600, \quad \forall x_i. && \text{(2f)}
\end{aligned}$$

It is obvious that Equation (2) is convex, we will show how to solve it in the following.

4.1 Lagrangian Relaxation

In the Lagrangian relaxation procedure, the constraints that make the problem hard to solve will be moved to the objective. Here, the constraints shown in Equations (2b)–(2e) are relaxed. For each relaxed constraint, a non-negative variable called Lagrange multiplier is introduced such that any violations of these constraints will be penalized and hence avoided. To be specific, we introduce λ_i for the wire limit constraint of each FPGA-pair i and $\mu_{p,q}$ for the timing constraint imposed by $e_{p,q}$. Note that, $\boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{at}$ denote the sets of variables $\mu_{p,q}, \lambda_i, x_i, at_p$, respectively. Let

$$\begin{aligned}
L_{\boldsymbol{\mu},\boldsymbol{\lambda}}(\boldsymbol{x}, \boldsymbol{at}) = & \; at_{sink} + \sum_{i \in T} \lambda_i \cdot \left(\sum_{j \in t_i} \frac{1}{x_j} - p_i \right) && \text{(3a)}\\
& + \sum_{x_i} \sum_{e^x_{p,q} \in E_i} \mu_{p,q} \cdot (at_p + x_i - at_q) && \text{(3b)}\\
& + \sum_{e^c_{p,q}} \mu_{p,q} \cdot (at_p - at_q) + \sum_{e^c_{src,q}} \mu_{src,q} \cdot (-at_q). && \text{(3c)}
\end{aligned}$$

Then the Lagrangian relaxation subproblems (LRS) associated with the Lagrange multipliers $\boldsymbol{\mu}, \boldsymbol{\lambda}$ will be

$$\begin{aligned}
\mathcal{LRS}/(\boldsymbol{\mu}, \boldsymbol{\lambda}): \quad \min \quad & L_{\boldsymbol{\mu},\boldsymbol{\lambda}}(\boldsymbol{x}, \boldsymbol{at}), && \text{(4a)}\\
s.t. \quad & 1 \le x_i \le 1{,}600, \quad \forall x_i. && \text{(4b)}
\end{aligned}$$

4.2 Solving $\mathcal{LRS}/(\boldsymbol{\mu}, \boldsymbol{\lambda})$

By the KKT conditions, the optimal solutions of $\mathcal{PP}$ also optimize $\mathcal{LRS}/(\boldsymbol{\mu}, \boldsymbol{\lambda})$, so $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ must satisfy the conditions $\frac{\partial L_{\boldsymbol{\mu},\boldsymbol{\lambda}}}{\partial at_p} = 0$ and $\frac{\partial L_{\boldsymbol{\mu},\boldsymbol{\lambda}}}{\partial x_i} = 0$ at the optimum. Therefore, in searching for the $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ to optimize the Lagrangian dual problem (LDP) shown in Equation (9), we only consider those multipliers that satisfy these conditions, which implies the optimality conditions on the multipliers as shown in Equations (5)–(7).

$$\sum_{e_{p,sink}} \mu_{p,sink} = 1, \tag{5}$$

$$\sum_{e_{p,i}} \mu_{p,i} = \sum_{e_{i,q}} \mu_{i,q}, \quad \forall g_i, \tag{6}$$

$$\lambda_i \frac{1}{x_j^2} = \sum_{e^x_{p,q} \in E_j} \mu_{p,q}, \quad \forall j \in t_i; \forall i \in T. \tag{7}$$

Hence, given $(\boldsymbol{\mu}, \boldsymbol{\lambda})$, we can get $\boldsymbol{x}$ as below:

$$x_j = \min\left(1{,}600, \max\left(1, \sqrt{\frac{\lambda_i}{\sum_{e^x_{p,q} \in E_j} \mu_{p,q}}}\right)\right), \quad \forall j \in t_i; \forall i \in T. \tag{8}$$

By combining and rearranging Equations (3), (5), and (6), the coefficients of all at_p become zero. In our method, we forward propagate to calculate at_p after getting $\boldsymbol{x}$ by Equation (8).

4.3 Solving the $\mathcal{LDP}$

Given that $\boldsymbol{\mu}, \boldsymbol{\lambda}$ are two sets of non-negative variables, it is obvious that the optimal value of $\mathcal{LRS}/(\boldsymbol{\mu}, \boldsymbol{\lambda})$ is no greater than that of $\mathcal{PP}$. Hence, if we maximize the minimum value of $L_{\boldsymbol{\mu},\boldsymbol{\lambda}}$, we obtain a tighter lower bound of the $\mathcal{PP}$. By updating $\boldsymbol{\mu}, \boldsymbol{\lambda}$ iteratively, the lower bound imposed by $L_{\boldsymbol{\mu},\boldsymbol{\lambda}}$ will become closer to the optimal solution of $\mathcal{PP}$ and finally converge. Let $Q(\boldsymbol{\mu}, \boldsymbol{\lambda})$ denote the optimal value of $\mathcal{LRS}/(\boldsymbol{\mu}, \boldsymbol{\lambda})$. The LDP is as follows:

$$\mathcal{LDP}: \quad \max \quad Q(\boldsymbol{\mu}, \boldsymbol{\lambda}), \tag{9a}$$

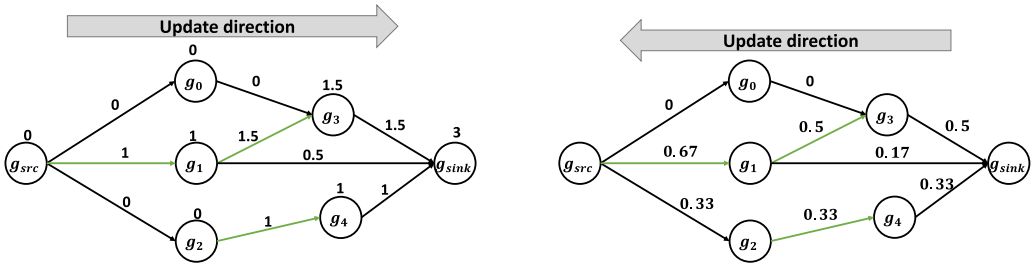
$$s.t. \quad \lambda_i \geq 0, \mu_{p,q} \geq 0, \quad \forall i \in T; \forall e_{p,q}, \tag{9b}$$

$$\sum_{e_{p,sink}} \mu_{p,sink} = 1, \tag{9c}$$

$$\sum_{e_{p,i}} \mu_{p,i} = \sum_{e_{i,q}} \mu_{i,q}, \quad \forall g_i. \tag{9d}$$

Since $\mathcal{PP}$ is convex, we can apply Theorem 6.2.4 of [21], which implies that the optimal solution of $\mathcal{LDP}$ will also optimize $\mathcal{PP}$.

As mentioned before, we only need to consider $(\boldsymbol{\mu}, \boldsymbol{\lambda})$ in the solution space Ω defined by Equations (9b)–(9d) when we optimize the $\mathcal{LDP}$. Previous works usually use an arbitrary assignment of multipliers as a starting point and then iteratively update the multipliers. To update the multipliers, a two-step approach is usually used, which first applies subgradient optimization to update the multipliers and then projects them back to the solution space defined by the KKT conditions. However, we find these methods hard to control and slow due to the projection step and the bad starting point. Hence, unlike previous methods, we start with a good initial assignment of the multipliers in Ω, which is shown in Section 4.3.1. The multipliers are then updated by a novel method shown in Section 4.3.2, which reflects the nature of timing optimization and considers the



(a) PTE calculation. (b) μ initialization.

Fig. 5. An example of how the PTE is computed and the initialization of μ. Note that the green edges represent inter-FPGA nets.

ALGORITHM 1: Solving the $\mathcal{LDP}$.

Input: The timing graph and the architecture information.
Output: A set of continuous TDM ratios with optimized system clock ratio.

```
initialize the multipliers;                                   ▷ see Section 4.3.1
repeat
    solve LRS;                                                ▷ see Section 4.2
    update the best solution if the objective value is better;
    update the multipliers;                                   ▷ see Section 4.3.2
until reach iteration limit or no more improvement
restore the best solution;
```

KKT conditions during the updates. Note that, since the methods are all based on a topological order of the circuit graph, the computations can be easily parallelized among the nodes at the same topological level. Algorithm 1 shows the overall flow of solving $\mathcal{LDP}$.

4.3.1 Multiplier Initialization. As shown in Sections 4.1 and 4.2, $\mu_{p,q}$ indicates the timing criticality of $e_{p,q}$ when solving $\boldsymbol{\mathcal{LRS}/(\mu, \lambda)}$. However, the timing criticality of each edge is unknown during initialization. As mentioned in previous sections, a large part of the system clock period comes from the delay of the time-multiplexed wires; we thus can correlate the initial value of $\mu_{p,q}$ with the number of TDM edges that affect $at_{p,q}$. The numbers of TDM edges encountered on any paths from a source to a node g_p or to an edge $e_{p,q}$, called preceding TDM edges (PTE), are first calculated as shown in Equations (10)–(12).

$$PTE_{e^c_{p,q}} = \frac{PTE_{g_p}}{|\{i|e_{p,i} \in G\}|}, \quad \forall e^c_{p,q}, \tag{10}$$

$$PTE_{e^x_{p,q}} = \frac{PTE_{g_p}}{|\{i|e_{p,i} \in G\}|} + 1, \quad \forall e^x_{p,q}, \tag{11}$$

$$PTE_{g_p} = \begin{cases} 0, & \text{if } g_p = g_{src}, \\ \sum_{e_{i,p}} PTE_{e_{i,p}}, & \text{otherwise.} \end{cases} \tag{12}$$

The constraints shown in Equations (9c) and (9d) are transformed into flow constraints in the timing graph G. Multiplier $\mu_{p,q}$ for each edge $e_{p,q}$ is then initialized as shown in Equation (13), which will ensure that the flow constraints are satisfied since $PTE_{g_p} = \sum_{e_{i,p}} PTE_{e_{i,p}}$ as in Equation (12). Now $\mu_{p,q}$ is proportional to the number of TDM edges that affect $at_{p,q}$.

ALGORITHM 2: Balance flow through g_p.

Input: An unbalance flow of g_p where $\sum_{e_{p,i}} \mu'_{p,i} < \sum_{e_{j,p}} \mu_{j,p}$ and a set of multipliers $M \subset \{\mu_{j,p} | e_{j,p} \in G\}$ to update.

Output: $\sum_{e_{p,i}} \mu'_{p,i} = \sum_{e_{j,p}} \mu'_{j,p}$ is satisfied.

$\Delta \leftarrow \sum_{e_{p,i}} \mu'_{p,i} - \sum_{e_{j,p}} \mu_{j,p}$;
sort the multipliers in M in non-decreasing order of their arrival time gaps $(at_{j,p} - at_p)$; let $s[k]$ be the arrival time gap of the k^{th} multiplier $\mu[k]$ in the sorted sequence;
let $lastIdx$ be the smallest index such that $s[k] \cdot \mu[k] = 0$ holds for all $k \in [lastIdx, |M| - 1]$;
$curIdx \leftarrow 0$;
$\alpha \leftarrow \frac{|\Delta|}{\sum_{k=curIdx}^{|M|-1} (\mu[k] \cdot \frac{s[k]}{s[curIdx]})}$;
while $\alpha > 1$ **do**
  if $curIdx = lastIdx$ **then**
    $muSum \leftarrow \sum_{k=curIdx}^{|M|-1} \mu[k]$;
    $\mu[i] \leftarrow \mu[i] - \Delta \cdot \frac{\mu[i]}{muSum}$ for all $i \in [curIdx, |M| - 1]$;
    quit procedure;
  end if
  $\Delta \leftarrow \Delta + \mu[curIdx]$, $\mu[curIdx] \leftarrow 0$, $curIdx \leftarrow curIdx + 1$;
  $\alpha \leftarrow \frac{|\Delta|}{\sum_{k=curIdx}^{|M|-1} (\mu[k] \cdot \frac{s[k]}{s[curIdx]})}$;
end while
$\mu[i] \leftarrow \mu[i] \cdot (1 - \alpha \cdot \frac{s[i]}{s[curIdx]})$ for all $i \in [curIdx, |M| - 1]$;

$$\mu_{p,q} = \begin{cases} \frac{PTE_{e_{p,q}}}{PTE_{g_q}} \cdot 1, & \text{if } g_q \text{ is the sink,} \\ \frac{PTE_{e_{p,q}}}{PTE_{g_q}} \cdot \sum_{e_{q,i}} \mu_{q,i}, & \text{otherwise.} \end{cases} \tag{13}$$

Figure 5 is an example of how we calculate the PTE and initialize $\boldsymbol{\mu}$.

After getting the initial assignment of $\boldsymbol{\mu}$, we initialize $\boldsymbol{\lambda}$ such that the resources of each FPGA-pair are used as much as possible. From Equation (8) and the constraint that the maximum usage should be no greater than p_j for each FPGA-pair j, we have

$$maxUsage_j = \sum_{i \in t_j} \frac{1}{x_i} = \sum_{i \in t_j} \sqrt{\frac{\sum_{e^x_{p,q} \in E_i} \mu_{p,q}}{\lambda_j}} \leq p_j, \ \forall j \in T. \tag{14}$$

Moreover, we want to make sure that the initial value of each x_i, calculated by Equation (7), is no less than 1. Hence, $\boldsymbol{\lambda}$ is initialized as below:

$$\lambda_j = \max\left(\left(\frac{\sum_{i \in t_j} \sqrt{\sum_{e^x_{p,q} \in E_i} \mu_{p,q}}}{p_j}\right)^2, \max_{i \in t_j}\left(\sum_{e^x_{p,q} \in E_i} \mu_{p,q}\right)\right), \quad \forall j \in T, \tag{15}$$

where the first part maximizes wire usage and the second part ensures $x_i \geq 1$ for all $i \in t_j$.

4.3.2 Multiplier Update. After solving $\mathcal{LRS}/(\boldsymbol{\mu}, \boldsymbol{\lambda})$, we will update $\boldsymbol{\mu}$ according to the timing criticality of each edge. A ratio $r = \alpha \cdot 0.5^{\delta \cdot i}$ is used to control the convergence speed and quality, where i is the current number of iterations and α, δ are set to $0.2, 0.01$, respectively, in our implementation.

We will update $\boldsymbol{\mu}$ in a propagation-based approach similar to greedy timing optimization, which tries to reduce the delay along the critical paths in each iteration. First, we will change $\mu_{i,sink}$ as shown below:

$$\mu'_{i,sink} = \begin{cases} \mu_{i,sink} \cdot (1+r), & \text{if } at_{i,sink} \geq 0.95 \cdot at_{sink}, \\ \text{updated as shown in Algorithm 2}, & \text{otherwise.} \end{cases} \tag{16}$$

To be specific, Algorithm 2 will try to decrease the given set of multipliers in proportion to their values and gradients such that the flow constraints shown in Equation (9c) are satisfied. By doing so, the multipliers associated with the critical edges connected to the sink are increased while the others are decreased, which results in smaller differences between the arrival time along different fan-in edges of the sink since the multipliers associated with the critical (noncritical) paths are increased (decreased) and x_j is proportional to $\sqrt{\frac{1}{\sum_{e^x_{p,q} \in E_j} \mu_{p,q}}}$.

After changing the multipliers associated with the sink, we will propagate the changes to the others such that the flow constraints in Equation (9d) are satisfied. When a node g_p whose flow constraint is violated after the multipliers associated with its fan-out edges are updated, there are two cases: (1) $\sum_{e_{j,p}} \mu_{j,p} < \sum_{e_{p,i}} \mu_{p,i}$ and (2) $\sum_{e_{j,p}} \mu_{j,p} > \sum_{e_{p,i}} \mu_{p,i}$. The first case infers that at_p is on at least one of the critical paths and at_p need to be decreased in order to improve at_{sink}. Hence, we will average out the difference among the edges $\{e_{i,p} | at_{i,p} \geq 0.95 \cdot at_p\}$ as shown in Equation (17) since they are considered to have similar timing criticality.

$$\mu'_{i,p} = \mu_{i,p} + \frac{\mu_{i,p}}{\sum\limits_{\{k | at_{k,p} \geq 0.95 \cdot at_p\}} \mu_{k,p}} \cdot \left(\sum_{e_{p,j}} \mu_{p,j} - \sum_{e_{k,p}} \mu_{k,p} \right). \tag{17}$$

Although gate g_p in case (2) is less timing critical compared to those in case (1), it may still be on one of the critical paths. Algorithm 2 is used to update the multipliers corresponding to the drivers of g_p. To minimize the possibility of increasing at_{sink}, we first try to only decrease the multipliers $\mu_{j,p}$ for all $j \in \{k | at_{k,p} < at_p\}$. When the flow constraint cannot be satisfied by only decreasing these multipliers, we will average out the difference to each multiplier in proportion to its value as shown on lines 7–11. However, there is a special case where the arrival time along each fan-in edge is equal to at_p. For such case, we will average out the difference among all the fan-in edges in the same way as in Equation (17).

After updating $\boldsymbol{\mu}$, $\boldsymbol{\lambda}$ is updated in the same way as in Equation (15).

5 DISCRETIZATION

Since the continuous solver optimizes the TDM ratios in a global manner, its result should be honored during discretization. Instead of minimizing the total displacement, our discretization algorithm will find a legal TDM assignment for each FPGA-pair with minimum total displacement under the optimal maximum displacement bound. The problem formulation of the FPGA-pair A is shown in Equation (18).

$$\min \quad \sum_{i \in t_A} |x_i^{cont} - x_i^{discr}|, \tag{18a}$$

$$s.t. \quad \text{All TDM ratio constraints are satisfied}, \tag{18b}$$

$$|x_i^{cont} - x_i^{discr}| \leq bnd_A, \quad \forall i \in t_A, \tag{18c}$$

where bnd_A is the optimal maximum displacement bound of the FPGA-pair A. Note that, since each FPGA-pair is independent in our formulation, they can be discretized in parallel.

The optimal maximum displacement bound bnd_A is first found by a binary search–based method as shown in Algorithm 3. During the binary search, given a maximum displacement bound $disp_{bnd}$,

ALGORITHM 3: Binary Search–Based Bound Searching.

Input: The continuous TDM ratios of the FPGA-pair A.
Output: The minimum feasible maximum displacement bnd_A.

```
1: get the lower bound lb of maximum displacement;
2: sort the tdmNets in the FPGA-pair A in non-decreasing order of their continuous solutions; let the sorted
   sequence be seq;
3: sort seq stably such that the tdmNets under forward direction are always in front of those under backward
   direction;
4: let x[k], dir[k] be the TDM ratio and direction of the k^th tdmNet in seq;
5: using binary search from lb to find the minimum feasible maximum displacement;

6: function getMinWire(idx)
7:     minWireRequire ← 0;
8:     for i ← idx; i < |t_A|; do
9:         maxChoice ← the largest available choice of x[i];
10:        i ← getEndIdx(idx, maxChoice);
11:        increase minWireRequire by one;
12:    end for
13:    return minWireRequire;
14: end function

15: function getEndIdx(idx, choice)
16:    maxEndIdx ← min(|t_A|, idx + choice);
17:    for i ∈ [idx + 1, |t_A| − 1] do
18:        return i if choice is out of the choice range of x[i] or dir[i] ≠ dir[idx];
19:    end for
20:    return maxEndIdx;
21: end function
```

we will first generate the available discrete TDM choices for each tdmNet such that the distance between the continuous solution and every choice is within $disp_{bnd}$. The feasibility of $disp_{bnd}$ is equivalent to whether the minimum number of required physical wires under the current set of choices, obtained by `getMinWire(0)`, is no greater than the wire limit p_A. According to the feasibility of $disp_{bnd}$, we will continue the binary search until the minimum feasible maximum displacement bound is found.

Definition 3. Given the optimal maximum displacement bound bnd_A of FPGA pair A, for a tdmNet $net_a \in t_A$, the **choice range** of net_a is denoted as $rng_a = [rng_a^s, rng_a^e]$ where $rng_a^s = \max(x_a^{cont} - bnd_A, 1)$ and $rng_a^e = \max(x_a^{cont} + bnd_A, 1{,}600)$.

Given an FPGA pair A and the maximum displacement bound $disp_{bnd}$, Theorem 1 proves that `getMinWire(0)` will return the minimum number of wires needed under the given set of choices. It is easy to see that our binary search will return the optimal maximum displacement bound if Theorem 1 is true. For simplicity, in the proof of Theorem 1, we assume that all the signals of a FPGA-pair are in the same direction. Since signals in different directions cannot be assigned to the same wire, the complete proof can be easily deduced. Given a tdmNet net_a, let its TDM ratios before and after discretization be x_a^c and x_a^d, respectively.

Lemma 1. *Given a solution of TDM assignment which requires the minimum number of wires, it can be transformed into a solution that requires the same number of wires and $x_1^c \le x_2^c$ holds for any two tdmNets net_1, net_2 if $x_1^d \le x_2^d$.*

Proof. Given a solution of TDM assignment which requires the minimum number of wires, we first sort the wires in non-decreasing order of their TDM ratios. Then, within each wire, we sort the tdmNets in non-decreasing order of their continuous solutions. Given two wires whose TDM ratios are r_1, r_2 ($r_1 \leq r_2$) and the two tdmNets net_a, net_b among them, respectively, if $x_a^c > x_b^c$, we can swap the TDM ratios of them without changing r_1, r_2 since r_1, r_2 is in the intersection ($rng_a^s \leq r_1 \leq r_2 \leq rng_b^e$) of the choice ranges of net_a, net_b. After repeatedly performing the swappings mentioned above, the solution still requires the same number of wires and $x_1^c \leq x_2^c$ holds for any two tdmNets net_1, net_2 if $x_1^d \leq x_2^d$. □

Theorem 1. `getMinWire(0)` *will return the minimum number of wires needed under the given set of choices.*

Proof. Given that the tdmNets are sorted in non-decreasing order of their continuous solutions, `getMinWire(0)` is a partitioning algorithm. Since it will maximize the TDM ratio and the number of signals of each partition, the resulting solution will require the minimum number of partitions which is equivalent to the number of wires. As shown in Lemma 1, a solution that requires minimum number of wires can be transformed into a partitioning of a non-decreasing order of the tdmNets' continuous TDM values. Hence, `getMinWire(0)` will return the minimum number of wires needed under the given set of choices. □

After finding the optimal maximum displacement bound, we will use a top-down DP to get a legal TDM assignment of the FPGA-pair A, which is the optimal solution of Equation (18). Details of the DP are shown in Algorithm 4. Note that, the discrete TDM choices are traversed in ascending order on line 12. Procedure `getMinPA`(i, pp) will generate the optimal TDM ratios for the i^{th} to the nth tdmNets in the sequence using pp wires. In the following, we will show an example of how to compute `getMinPA`. Given a set of discrete TDM choices tc^i for the i^{th} tdmNet such that $|tc_q^i - x[i]| \leq bnd_A$ holds for all $tc_q^i \in tc^i$, `getMinPA`(i, pp) is equal to $\min_{tc_q^i \in tc^i} cost_{tc_q^i}$, which is computed as below:

$$cost_{tc_q^i} = \min_{0 \leq j \leq n} \left\{ \texttt{getMinPA}(i + j + 1, pp - 1) + \sum_{k=i}^{i+j} |x[k] - tc_q^i| \right\},$$

where n is the maximum index such that $|x[j] - tc_q^i| \leq bnd_A$ holds for all $j \in [i, n + i]$ and $n < tc_q^i$.

Some pruning techniques are applied to accelerate the process without changing the optimality. As shown on line 12, we skip the discrete TDM choices smaller than the nearest one of $x[i]$ (obtained by `nearestTDM`) because smaller choices will induce more displacement and use more wire resources. $minIdx$ and $prevIdx$ denote the minimum numbers of tdmNets in the pp^{th} wire under the current and next discrete TDM choice, respectively. On line 14, we increase $minIdx$ as long as tc_q is the nearest discrete TDM choice of $x[i + minIdx + 1]$ because smaller $minIdx$ will increase the number of wires used without reducing the total displacement. On line 15, we increase $prevIdx$ as long as $x[i + prevIdx + 1] \leq tc_q$. It will be the starting value of $minIdx$ for the next discrete TDM choice tc_{q+1} since $\sum_{k=i}^{j} |x[k] - tc_q| < \sum_{k=i}^{j} |x[k] - tc_{q+1}|$ holds for any j no greater than $i + prevIdx$. On line 19, the loop ends since a larger j will only increase the total displacement ($curDisp$) of the pp^{th} wire which is already larger than the best cost ($curBest$) of the current discrete TDM choice tc_q.

Theorem 2 proves that Algorithm 4 can obtain the optimal solution for the problem shown in Equation (18).

Lemma 2. *Given a solution of Equation (18) where $x_a^c < x_b^c$ and $x_a^d > x_b^d$, if $x_a^d \in rng_b$ and $x_b^d \in rng_a$, we can swap their TDM ratios without increasing the cost of the solution.*

ALGORITHM 4: Total Displacement-Driven Discretization.

Input: The continuous TDM ratios of the FPGA-pair A and the maximum displacement bound bnd_A.
Output: A legal assignment of TDM ratios of the FPGA-pair A.

1: get the available discrete TDM choices for each tdmNet in the FPGA-pair A such that the distance between its continuous solution and every choice is within bnd_A;
2: sort the tdmNets in the FPGA-pair A in non-decreasing order of their continuous solutions; let the sorted sequence be seq;
3: sort seq stably such that the tdmNets under forward direction are always in front of those under backward direction;
4: let $x[k]$ be the TDM ratio of the k^{th} tdmNet in seq;
5: `getMinPA`$(0, p_A)$;
6: restore the best solutions from the DP;

7: **function** `getMinPA`(i, pp)
8: return the best cost at (i, pp) if computed already;
9: return 0 if $i \geq |t_A|$;
10: return $+\infty$ if `getMinWire`(i) $> pp$; ▷ `getMinWire`(i) is pre-computed for all $i \in [0, |t_A|)$
11: $prevIdx \leftarrow 0$;
12: **for each** choice tc_q in the choice range of $x[i]$ no less than `nearestTDM`$(x[i])$ **do**
13: $minIdx \leftarrow prevIdx + 1$;
14: increase $minIdx$ as long as $tc_q =$ `nearestTDM`$(x[i + minIdx + 1])$ and $minIdx + 1 <$ `getEndIdx`(i, tc_q);
15: increase $prevIdx$ as long as $x[i + prevIdx + 1] \leq tc_q$ and $prevIdx + 1 <$ `getEndIdx`(i, tc_q);
16: $curBest \leftarrow +\infty$;
17: **for** $j \in [i + minIdx,$ `getEndIdx`$(i, tc_q) - 1]$ **do**
18: $curDisp \leftarrow \sum_{k=i}^{j} |x[k] - tc_q|$;
19: jump to line 12 if $curDisp \geq curBest$;
20: $cost \leftarrow curDisp +$ `getMinPA`$(j + 1, pp - 1)$;
21: $curBest \leftarrow \min(cost, curBest)$;
22: update the best solution at (i, pp) if $cost$ is less than the best cost at (i, pp);
23: **end for**
24: **end for**
25: return the best cost at (i, pp);
26: **end function**

Proof. There are three cases: (1) $x_b^d < x_a^d < x_a^c$, (2) $x_b^c < x_b^d < x_a^d$, and (3) $x_b^d < x_a^c < x_a^d$ or $x_b^d < x_b^c < x_a^d$. The difference before and after swapping is $d = |x_a^c - x_a^d| + |x_b^c - x_b^d| - |x_a^c - x_b^d| - |x_b^c - x_a^d|$. For case 1, $d = (x_a^c - x_a^d) + (x_b^c - x_b^d) - (x_a^c - x_b^d) - (x_b^c - x_a^d) = 0$. For case 2, $d = (x_a^d - x_a^c) + (x_b^d - x_b^c) - (x_a^d - x_b^c) - (x_b^d - x_a^c) = 0$. Hence, swapping will not change the cost in the first two cases. An illustration of case (3) is given in Figure 6. We can see that if x_a^c and x_b^c are on different sides of x_m where $|x_m - x_b^d| = |x_m - x_a^d|$, both of their displacements will be reduced after swapping. If they are on the same side of x_m, the total displacement is also reduced. Hence, if $x_a^c < x_b^c$ and $x_a^d > x_b^d$, swapping their TDM ratios will not increase the cost of the solution. □

Corollary 1. *Given an optimal solution of Equation (18) where no swapping as described in Lemma 2 can be performed, it is true that $x_a^d \leq x_b^d$ if $x_a^c \leq x_b^c$.*

Theorem 2. *Algorithm 4 can get the optimal solution of Equation (18).*

Proof. We can see that Algorithm 4 exhausts all possible ways of partitioning the given non-decreasing sequence of continuous TDM ratios and returns the best partition in minimizing the

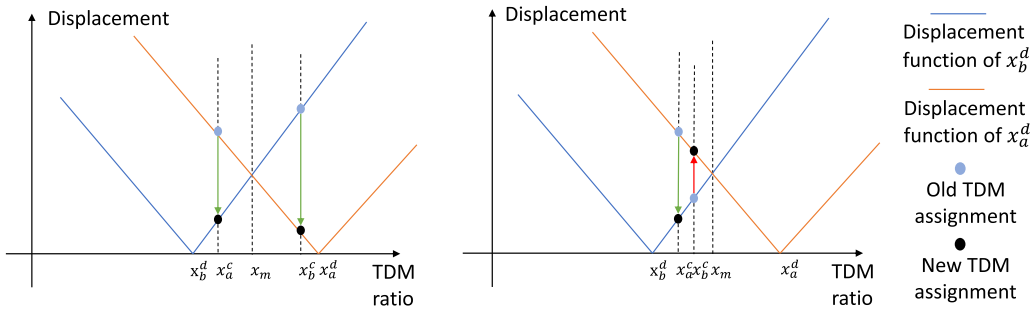


Fig. 6. An illustration of swapping the TDM ratios of two tdmNets.

total displacement. However, the optimality of Algorithm 4 (on the problem shown in Equation (18)) depends on whether there exists an optimal solution of the problem that can be transformed into a partitioning of a non-decreasing sequence of the tdmNets' continuous TDM values.

Given an optimal solution of Equation (18) where no swapping as described in Lemma 2 can be performed, we can first sort the sequence in a non-decreasing order of their discrete TDM ratios. According to Corollary 1, it is true that for all pairs a and b, if $x_a^d \leq x_b^d$, then $x_a^c \leq x_b^c$. For tdmNets that have the same discrete TDM ratios, they can also be sorted in a non-decreasing order of their continuous TDM ratios without changing the objective value. In this way, we can get a partitioning of a non-decreasing sequence of the tdmNets' continuous TDM ratios while the objective value is still optimal. □

6 POST REFINEMENT

In discretization, our algorithm is not aware of the timing criticality of each edge and assumes that smaller disturbance to the continuous solution will result in better timing. However, this might result in a solution that a critical edge in the timing graph is assigned to a larger TDM ratio. An example is shown below. Given a continuous solution $at_p = 8$, $at_m = 16$, $delay_{p,q} = 16$, and $delay_{m,n} = 8$, it is clear that the optimal results of these two edges have 0 displacement. But when we considering the discretization results of other edges in the timing graph, the values of at_p and at_q become 15 and 9. If we swap the TDM ratios of $e_{p,q}$ and $e_{m,n}$, $max(at_n, at_q)$ can be improved by 6. To avoid this situation, we propose a swapping-based post refinement process after discretization. To be specific, for the signals between each FPGA pair, we will swap their TDM ratios to improve the system clock period. Details of the algorithm are shown in Algorithm 5. On line 1, the system clock period is considered to have room for improvement if there is a successful swapping in the previous iteration and the result has been improved compared to the one 10 iteration earlier. On line 4, a tdmNet net_j is considered to be the candidate of net_i if it satisfies the following three constraints:

- net_i and net_j are in the same direction.
- The TDM ratio of net_j is smaller than that of net_i.
- Given $ResSlack_j = \min_{e_{p,q} \in E_j} slack_q - b_{p,q} \cdot (x_i - x_j)$, $ResSlack_j$ should be larger than 0.

Note that, $ResSlack_j$ is the largest possible remaining slack of a net_j after its TDM ratio is swapped with the one of net_i. On line 6, we first try those nets with larger $ResSlack$ since such swappings are more likely to improve the system clock period. Moreover, we also maintain a history of swapping and forbid repleting swappings, which can effectively prevent our algorithm from sticking in local optimal.

Table 2. The Information of the Generated Benchmark

Design	#Nets	#Inter-FPGA Net	#Nodes	#Edges
FPGA01	105,885	1,375	160,392	408,577
FPGA02	168,611	2,290	232,594	654,978
FPGA03	432,546	6,190	592,492	1,735,711
FPGA04	445,423	17,201	595,724	1,740,531
FPGA05	479,772	45,690	601,264	1,748,834
FPGA06	723,141	10,398	1,058,384	2,826,183
FPGA07	735,797	21,017	1,064,220	2,834,945
FPGA08	733,577	15,792	935,142	2,909,648
FPGA09	901,012	26,467	1,235,304	3,487,570
FPGA10	967,932	7,379	1,554,168	3,569,619
FPGA11	878,740	29,707	1,209,846	3,285,312
FPGA12	1,120,458	9,892	1,707,398	4,068,022

ALGORITHM 5: Swapping-Based Post Refinement.

Input: A legal TDM assignment.
Output: A legal TDM assignment with improved system clock ratio.

```
1:  while the system clock period can be improved do
2:      find the critical path and all the tdmNets N on it;
3:      for each net_i ∈ N do
4:          find all the candidate tdmNets N_c in the FPGA-pair that contains net_i;
5:          sort N_c in non-increasing order of ResSlack;
6:          for each net_j ∈ N_c do
7:              swap the TDM ratios if the resulting system clock ratio is not worsened;
8:              go to line 2 if the swapping is a success;
9:          end for
10:     end for
11: end while
```

7 EXPERIMENTAL RESULT

In this work, all algorithms are implemented in C++ and tested on a Linux workstation with an Intel Xeon 2.2 GHz CPU with 40 cores and 256GB memory. Note that all experiments are conducted under the limitation of eight threads. In the following, we will first introduce how we generate the benchmarks. Details of our baseline algorithms will then be shown. Finally, we will analyze the performance of our proposed framework.

7.1 Benchmark Generation

Our benchmarks are generated from the ISPD 2016 contest benchmarks [22] in three steps: (1) partitioning, (2) intra-FPGA global placement, and (3) timing graph construction. The results are shown in Table 2. The number of FPGAs is set to five and they are all directly connected, which means that they are adjacent to each other and there are 10 FPGA pairs. The number of physical wires between two FPGAs is set to 20. Patoh [23] is used for partitioning and RippleFPGA [24, 25] is used as our global placer.

Given the number of devices n, we first partition the netlist into n partitions such that the number of cuts is minimized. Note that, since the netlist is formulated as a hypergraph during partitioning, if a multi-pin net has terminals in k FPGAs, the cut size induced by this hyperedge is

Table 3. Notations of the LP Shown in Section 7.2

$x_i^{tc_j}$	Variable indicates whether tdmNet net_i is assigned to discrete TDM ratio tc_j. It is a continuous (binary) variable in Section 7.2.1 (7.2.2).
$x_{p,q}^{tc_j}$	$x_i^{tc_j} = x_{p,q}^{tc_j}$ for all $e_{p,q} \in E_i$.
n_{k,tc_j}^f (n_{k,tc_j}^b)	Integer variable indicates the number of forward (backward) wires used in the FPGA-pair k, whose TDM ratios are tc_j.
t_k^f (t_k^b)	The set of forward (backward) tdmNets of the FPGA-pair k in T.
tc	The set of all discrete TDM choices, which are $\{1, 8, 16, 25, \ldots, 1{,}600\}$.

considered to be $(k-1)$. For a net spanning multiple devices, we will divide it into several nets. For example, given a net spanning four devices, the intra-FPGA parts remain to be multi-pin nets while the inter-FPGA parts are decomposed into three nets such that each of them spans between two devices only. Note that these three nets are considered as different tdmNets in the optimization steps.

After partitioning the circuit, we will have the netlist for each FPGA. We can then perform global placement for each FPGA considering only intra-FPGA connections.

Finally, the timing graph is constructed, where the netlist must be acyclic. We will first decompose each FF, BRAM, and DSP into two nodes such that one is associated with the incoming signals and the other is with the outgoing signals. Besides these three types of logic elements, each LUT/IO corresponds to a node in G. Each n-pin net is represented by $(n-1)$ edges such that each edge connects the driver and one of the fan-out pins. There are two kinds of edges in the graph: one is for the inter-FPGA connections while the other one is for the intra-FPGA connections. For each intra-FPGA edge, we will use the global placement result to estimate its delay, which is proportional to the Manhattan distance between the logic elements it connects. For those inter-FPGA edges, each of them is associated with a TDM ratio, which is the variable we will optimize later. In our experiments, $b_{p,q}, c_{p,q}$ is set to 5 and 0 for all inter-FPGA edges. There is also a delay for each LUT, which is 2ps.

7.2 LP Baseline

Since there is no prior work that directly works on this problem, two LP-based methods are proposed as our baselines for comparison, where Gurobi [26] is used as our LP solver. The notations are shown in Table 3.

7.2.1 Continuous LP Baseline. As shown in Equation (19), a continuous LP-based method is proposed as a baseline for comparison with the continuous solver in our method.

$$\min \quad at_{\text{sink}}, \tag{19a}$$

$$s.t. \quad \sum_{tc_j \in tc} x_i^{tc_j} = 1, \quad \forall i, \tag{19b}$$

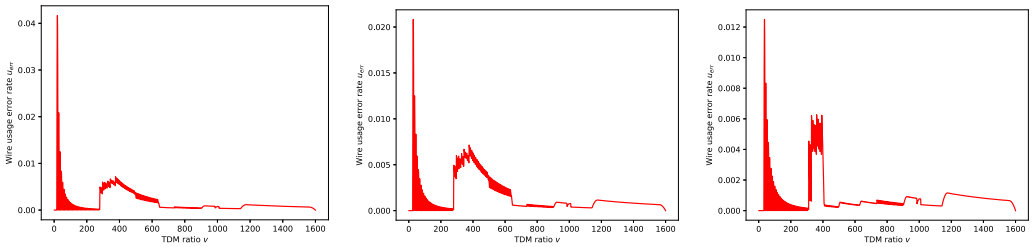
$$\sum_{i \in t_k} \sum_{tc_j \in tc} x_i^{tc_j} \cdot \frac{1}{tc_j} \le p_k, \quad \forall k \in T, \tag{19c}$$

$$at_q \ge at_p + b_{p,q} \cdot \sum_{tc_j \in tc} x_{p,q}^{tc_j} \cdot tc_j + c_{p,q}, \quad \forall e_{p,q}^x, \tag{19d}$$

$$at_q \ge at_p + delay_{p,q}, \quad \forall e_{p,q}^c, \tag{19e}$$

$$0 \le x_i^{tc_j} \le 1, \tag{19f}$$

$$at_{src} = 0. \tag{19g}$$



(a) adding integers under 16 (b) adding integers under 24 (c) adding integers under 32

Fig. 7. The error ratio calculated by Equation (20) after adding integers to tc in Equation (19).

Note that the TDM ratio x_i is represented by $\sum x_i^{tc_j} \cdot tc_j$, whose wire usage is $\frac{1}{\sum x_i^{tc_j} \cdot tc_j}$. Since this cannot be expressed in LP, we approximate it by $\sum x_i^{tc_j} \cdot \frac{1}{tc_j}$. We call the error of this approximation u_{err}. The reason we choose this method to represent the TDM ratio and wire usage of net_i is that the error is zero whenever x_i is equal to any $tc_j \in tc$. To reduce the error caused by the approximation, we add some additional small TDM choices in the LP. To be specific, in addition to the original discrete TDM choices, we add the integers under 16 to tc in our implementation. To verify the effect of this method, the LP shown in Equation (20) is performed for all $v \in [1, 1{,}600]$, whose results are shown in Figure 7(a).

$$\min \quad u_{err} = \frac{\sum_{tc_j \in tc} x^{tc_j} \cdot \frac{1}{tc_j} - \frac{1}{v}}{\frac{1}{v}}, \tag{20a}$$

$$s.t. \quad \sum_{tc_j \in tc} x^{tc_j} = 1, \tag{20b}$$

$$\sum_{tc_j \in tc} x^{tc_j} \cdot tc_j = v, \tag{20c}$$

$$0 \le x^{tc_j} \le 1, \tag{20d}$$

where x^{tc_j} is a continuous variable and v is the TDM ratio to be measured. We can see that, for any $v \in [1, 1{,}600]$, a linear combination of x^{tc_j} can always be found such that the estimated wire usage is about 4% larger than the actual wire usage.

It is clear that the method we use to reduce the error rate is a tradeoff between runtime and quality since we can further reduce the error by adding more small choices to tc, which is shown in Figure 7(b) and (c).

Since Equation (19) has the same constraints and objective as Equation (2), we can prove that there is a guaranteed error bound to the optimal solution of $\mathcal{PP}$ by approximating it with the optimal LP solution.

Theorem 3. *Given that the wire usage error rate u_{err} is no greater than r for all $v \in [1, 1{,}600]$, the optimal solution of Equation (19) has a guaranteed error bound of r to the optimal solution of $\mathcal{PP}$.*

Proof. Let $x_i = \sum x_i^{tc_j} \cdot tc_j$ and the optimal solution of $\mathcal{PP}$ be $\boldsymbol{x}^*$. Consider $x_i = (1 + r) \cdot x_i^*$. From Figure 7, we can see that, for any $x_i \in [1, 1{,}600]$, there is always a linear combination of $x_i^{tc_j}$ satisfying Equations (19b) and (19f) such that $\frac{1+r}{x_i} \ge \sum x_i^{tc_j} \cdot \frac{1}{tc_j} \ge \frac{1}{x_i}$. Since $\boldsymbol{x}^*$ is the optimal solution of $\mathcal{PP}$, we have

$$\sum_{i \in t_k} \sum x_i^{tc_j} \cdot \frac{1}{tc_j} \le \sum_{i \in t_k} \frac{1+r}{x_i} = \sum_{i \in t_k} \frac{1+r}{x_i^* \cdot (1+r)} = \sum_{i \in t_k} \frac{1}{x_i^*} \le p_k, \forall k \in T.$$

Hence, we can conclude that $(1 + r) \cdot x_i^*$ is in the solution space of Equation (19). Given that the objective value of Equation (19) is a linear function of x_i which equals $(1 + r) \cdot x_i^*$, the ratio between the optimal solution of Equation (19) and the optimal solution of $\mathcal{PP}$ is at most $(1 + r)$. □

Moreover, to improve the runtime of the baseline, we limit the number of available discrete TDM choices used to represent the continuous TDM ratio. To be specific, for each tdmNet, the difference between the discrete choices and its continuous solution is set to at most 64.

7.2.2 Mixed Integer Linear Programming Baseline. A mixed integer linear programming (MILP) is proposed as the baseline for comparison with the original TDM optimization problem shown in Section 2.3. The formulation is the same as Equation (19) except that $x_i^{tc_j}$ is a binary variable and Equation (19c) is substituted by Equations (21) and (22) to model the exact wire usage.

$$n_{k,tc_j}^{dir} \geq \frac{\sum_{i \in t_k^{dir}} x_i^{tc_j}}{tc_j}, \quad \forall k \in T; \forall tc_j \in tc; dir \in \{f, b\}, \tag{21}$$

$$\sum_{tc_j \in tc} n_{k,tc_j}^{b} + n_{k,tc_j}^{f} \leq p_k, \quad \forall k \in T. \tag{22}$$

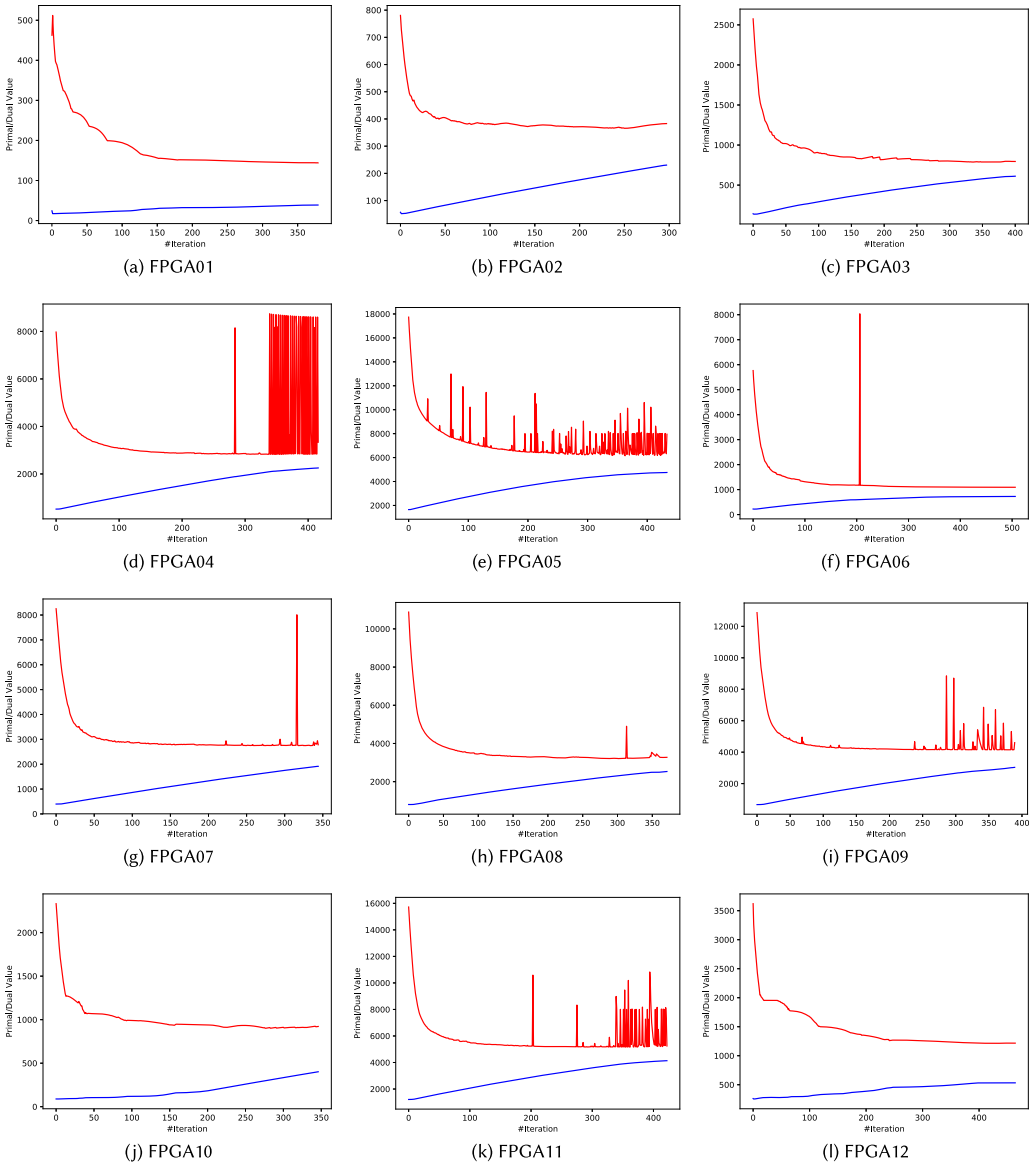
7.3 Result Analysis

In this section, we will analyze the results of our method. First we will discuss the performance of our continuous solver in terms of quality and scalability compared with the LP-based methods. Then we will compare the performance of our discretization method with a total displacement-driven discretization.

7.3.1 Result of Lagrangian-Based Continuous Solver. As shown in Section 4.3, our continuous solver will stop once the results converge or the iteration limit is reached. In our experiment, the iteration limit is set to 1,000. Figure 8 shows the convergence graph of our Lagrangian-based continuous solver, where the red and blue lines represent the primal and dual values, respectively. Our solver converges around 400 iterations in most of the test cases but there are some glitches on the convergence curves, especially for those iterations that are near the end of our solver. The reason for the glitches is that our multiplier updater may change some of the multipliers too much, which results in some very big TDM ratios. However, as shown in the figures, our solver recovers from these kinds of glitches very quickly and manages to converge in terms of primal and dual values.

To evaluate the result quality, we have run the LP-based methods with two sets of settings, which are the original LP and the LP with reduced choices. Note that the LP with reduced choices is based on the continuous results of the Lag solver (by limiting the choices to the surrounding discrete choices of the continuous solution). Table 6 shows a comparison of the system clock period (SCP) and the runtime between different methods. Based on our Lag method, the LP method with reduced choices can achieve about 5% improvement in SCP with 3× runtime. If we run the original LP, it cannot finish in large scale test cases due to the large number of variables and constraints. Note that the memory usage of our method for large scale designs is less than 2 GB, while the original LP may use up to 30 GB.

7.3.2 Result of Max-Displacement Bounded Discretization. Table 4 shows the performance comparison between our discretization methods and the one with total displacement minimized. Our proposed method can achieve 18% improvement in system clock period with significantly faster runtime. It also infers that maximum displacement is a better metric compared with total displacement since it measures the maximum disturbance to the continuous solution, which is more



(a) FPGA01 (b) FPGA02 (c) FPGA03

(d) FPGA04 (e) FPGA05 (f) FPGA06

(g) FPGA07 (h) FPGA08 (i) FPGA09

(j) FPGA10 (k) FPGA11 (l) FPGA12

Fig. 8. The convergence curves of different test cases. Red line is the value of primal while the blue line represents the dual value.

related to how the system clock period is calculated. With the pruning techniques we propose, the runtime of our discretization method can be further improved by up to 50% on large test cases like FPGA5 and FPGA11. Note that, since our discretization is very fast on small test cases, the runtime variations due to operating system may significantly affect the performance evaluation of our pruning techniques on small cases.

7.3.3 Result of Swapping-Based Post Refinement. Table 5 shows the performance of our post refinement method. Based on the results generated by our proposed Lagrangian solver and

Table 4. System Clock Period and Runtime Comparison between Different Discretization Methods

Design	System Clock Period (ps)						Runtime (sec)					
	Disp-Flow		MaxDisp-Flow		MaxDisp-Flow (Prune)		Disp-Flow		MaxDisp-Flow		MaxDisp-Flow (Prune)	
FPGA01	183	1	181	0.989	181	0.989	0.29	1	0.20	0.710	0.17	0.601
FPGA02	512	1	424	0.829	424	0.829	1.03	1	0.25	0.242	0.31	0.305
FPGA03	956	1	886	0.927	886	0.927	6.75	1	0.88	0.130	0.98	0.146
FPGA04	3,653	1	3,439	0.942	3,439	0.942	146.61	1	1.92	0.013	1.53	0.010
FPGA05	9,885	1	8,518	0.862	8,518	0.862	844.86	1	20.81	0.025	9.45	0.011
FPGA06	1,496	1	1,259	0.842	1,259	0.842	18.30	1	1.29	0.070	1.10	0.060
FPGA07	3,715	1	3,172	0.854	3,172	0.854	154.85	1	2.12	0.014	1.81	0.012
FPGA08	6,694	1	4,214	0.630	4,214	0.630	192.68	1	3.17	0.016	1.67	0.009
FPGA09	7,133	1	5,090	0.714	5,090	0.714	328.64	1	2.74	0.008	2.14	0.007
FPGA10	1,421	1	1,078	0.758	1,078	0.758	18.08	1	1.86	0.103	1.30	0.072
FPGA11	9,508	1	7,268	0.764	7,268	0.764	513.47	1	12.07	0.024	6.37	0.012
FPGA12	1,891	1	1,488	0.787	1,488	0.787	23.14	1	1.60	0.069	2.18	0.094
Average		1		0.825		0.825		1		0.119		0.112

[1] Both discretization methods are based on the results of our Lagrangian solver.

Table 5. System Clock Period and Runtime of Our Post Refinement Method

Design	System Clock Period (ps)				Runtime (sec)			
	w/o Refinement		w/ Refinement		w/o Refinement		w/ Refinement	
FPGA01	181	1	174	0.963	29	1	31	1.053
FPGA02	424	1	415	0.977	33	1	38	1.156
FPGA03	886	1	863	0.974	129	1	151	1.169
FPGA04	3,439	1	3,153	0.917	148	1	190	1.282
FPGA05	8,518	1	8,325	0.977	190	1	192	1.013
FPGA06	1,259	1	1,151	0.914	275	1	324	1.178
FPGA07	3,172	1	2,960	0.933	198	1	358	1.809
FPGA08	4,214	1	4,214	1.000	160	1	162	1.012
FPGA09	5,090	1	4,695	0.922	236	1	358	1.517
FPGA10	1,078	1	1,066	0.989	252	1	257	1.021
FPGA11	7,268	1	7,268	1.000	260	1	262	1.008
FPGA12	1,488	1	1,288	0.866	434	1	520	1.198
Average		1		0.953		1		1.201

[1] The post refinement process is performed on the results of our proposed Lagrangian solver and discretization method.

discretization method, it can improve the system clock ratio 5% on average with 20% runtime overhead.

7.3.4 Result of Our Proposed Framework. Table 6 shows the results after discretization of different continuous methods. As one can see, although our Lag method has worse SCP in the continuous domain compared to the LP methods, it results in similar or even better SCP after discretization. This is because there is a gap between the continuous and final results due to the discrete TDM choices and wire limit constraints. The continuous result with better SCP might be far from the discrete choices, which will be severely disturbed in discretization. Note that we also run the MILP

Table 6. System Clock Period and Runtime Comparison between Different Continuous Methods

Design	System Clock Period (ps)																		Runtime (sec)					
	Continuous						Discretized						Post-Refinement											
	Lag		Lag+LP_64		LP		Lag		Lag+LP_64		LP		Lag		Lag+LP_64		LP		Lag		Lag+LP_64		LP	
FPGA01	144	1	134	0.933	134	0.933	181	1	177	0.978	177	0.979	174	1	174	0.999	174	0.999	31	1	36	1.170	20	0.660
FPGA02	366	1	361	0.986	361	0.986	424	1	433	1.021	433	1.021	415	1	418	1.008	418	1.008	38	1	49	1.295	37	0.964
FPGA03	787	1	737	0.936	737	0.936	886	1	875	0.988	875	0.988	863	1	857	0.993	857	0.993	151	1	224	1.485	223	1.484
FPGA04	2,827	1	2,724	0.963	2,721	0.962	3,439	1	3,371	0.980	3,299	0.959	3,153	1	3,116	0.988	3,220	1.021	190	1	744	3.908	8,105	42.587
FPGA05	6,140	1	5,837	0.951	-	-	8,518	1	8,799	1.033	-	-	8,325	1	8,485	1.019	-	-	192	1	1,529	7.945	-	-
FPGA06	1,093	1	1,003	0.918	1,003	0.918	1,259	1	1,267	1.006	1,322	1.050	1,151	1	1,267	1.100	1,267	1.100	324	1	681	2.099	1,613	4.974
FPGA07	2,747	1	2,648	0.964	-	-	3,172	1	3,165	0.998	-	-	2,960	1	3,005	1.015	-	-	358	1	1,220	3.411	-	-
FPGA08	3,217	1	3,101	0.964	3,098	0.963	4,214	1	4,294	1.019	4,214	1.000	4,214	1	4,294	1.019	4,214	1.000	162	1	918	5.651	3,339	20.553
FPGA09	4,146	1	-	-	-	-	5,090	1	-	-	-	-	4,695	1	-	-	-	-	358	1	-	-	-	-
FPGA10	901	1	872	0.967	872	0.967	1,078	1	1,132	1.050	1,092	1.013	1,066	1	1,070	1.003	1,066	1.000	257	1	347	1.348	570	2.214
FPGA11	5,163	1	-	-	-	-	7,268	1	-	-	-	-	7,268	1	-	-	-	-	262	1	-	-	-	-
FPGA12	1,217	1	1,056	0.868	1,054	0.866	1,488	1	1,411	0.948	1,331	0.895	1,288	1	1,411	1.095	1,331	1.033	520	1	485	0.932	905	1.739
Average		1		0.945		0.941		1		1.002		0.988		1		1.024		1.019		1		2.924		9.397

[1] All ofthe continuous methods are discretized by our proposed discretization method.
[2] The time limit of LP is setto 10 ,000 seconds in our experiments. LP_64 represents the LP method with reduced choices while LP indicates the original LP method.

shown in Section 7.2.2 for up to 10,000 seconds, which fails to find a feasible solution for any cases. Even if starting with a very "bad" feasible solution, it cannot further improve the result.

8 CONCLUSION

In this work, we propose a novel method for the TDM optimization problem in a modern multi-FPGA system. To be specific, an analytical framework is used to minimize the system clock period, which consists of a Lagrangian relaxation-based continuous solver, a discretization considering both maximum and total displacement, and a swapping-based post refinement that optimizes the TDM ratios on critical paths. Experimental results show that our approach is scalable and effective in large scale multi-FPGA–based designs compared with LP-based methods. Future works may consider discretization effect in the continuous solver, and so forth.

REFERENCES

[1] Chak-Wa Pui and Evangeling F. Y. Young. 2019. Lagrangian relaxation-based time-division multiplexing optimization for multi-FPGA systems. In *Proc. ICCAD*.
[2] Andrew Ling and Jason Anderson. 2017. The role of FPGAs in deep learning. In *Proc. FPGA*. 3–3.
[3] George A. Constantinides. 2017. FPGAs in the cloud. In *Proc. FPGA*. 167–167.
[4] Scott Hauck. 1998. The roles of FPGAs in reprogrammable systems. *Proceedings of the IEEE* 86, 4 (1998), 615–638.
[5] Wan-Sin Kuo, Shi-Han Zhang, Wai-Kei Mak, Richard Sun, and Yoon Kah Leow. 2018. Pin assignment optimization for Multi-2.5 D FPGA-based systems. In *Proceedings of the 2018 International Symposium on Physical Design*. ACM, 106–113.
[6] Jonathan Babb, Russell Tessier, Matthew Dahl, Silvina Zimi Hanono, David M. Hoki, and Anant Agarwal. 1997. Logic emulation with virtual wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16, 6 (1997), 609–626.
[7] William N. N. Hung and Richard Sun. 2018. Challenges in large FPGA-based logic emulation systems. In *Proceedings of the 2018 International Symposium on Physical Design*. ACM, 26–33.
[8] Masato Inagi, Yasuhiro Takashima, Yuichi Nakamura, and Atsushi Takahashi. 2008. Optimal time-multiplexing in inter-FPGA connections for accelerating multi-FPGA prototyping systems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 91, 12 (2008), 3539–3547.

[9] Masato Inagi, Yasuhiro Takashima, and Yuichi Nakamura. 2009. Globally optimal time-multiplexing in inter-FPGA connections for accelerating multi-FPGA systems. In *International Conference on Field Programmable Logic and Applications (FPL'09)*. IEEE, 212–217.

[10] Masato Inagi, Yasuhiro Takashima, and Yuichi Nakamura. 2010. Globally optimal time-multiplexing of inter-FPGA connections for multi-FPGA prototyping systems. *IPSJ Transactions on System LSI Design Methodology* 3 (2010), 81–90.

[11] Masato Inagi, Yuichi Nakamura, Yasuhiro Takashima, and Shin'ichi Wakabayashi. 2015. Inter-FPGA routing for partially time-multiplexing inter-FPGA signals on multi-FPGA systems with various topologies. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 98, 12 (2015), 2572–2583.

[12] Shih-Chun Chen, Richard Sun, and Yao-Wen Chang. 2018. Simultaneous partitioning and signals grouping for time-division multiplexing in 2.5D FPGA-based systems. In *Proc. ICCAD*. 4.

[13] Chak-Wa Pui, Gang Wu, Freddy Y. C. Mang, and Evangeling F. Y. Young. 2019. An analytical approach for time-division multiplexing optimization in multi-FPGA based systems. In *Proc. SLIP*.

[14] Fung Yu Young, Chris C. N. Chu, W. S. Luk, and Y. C. Wong. 2000. Floorplan area minimization using Lagrangian relaxation. In *Proceedings of the 2000 International Symposium on Physical Design*. Citeseer, 174–179.

[15] Chuan Lin, Hai Zhou, and Chris Chu. 2006. A revisit to floorplan optimization by Lagrangian relaxation. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*. ACM, 164–171.

[16] Chung-Ping Chen, Chris C. N. Chu, and D. F. Wong. 1999. Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18, 7 (1999), 1014–1025.

[17] Hiran Tennakoon and Carl Sechen. 2002. Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design*. ACM, 395–402.

[18] Jia Wang, Debasish Das, and Hai Zhou. 2009. Gate sizing by Lagrangian relaxation revisited. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 7 (2009), 1071–1084.

[19] Gang Wu and Chris Chu. 2017. Two approaches for timing-driven placement by Lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 12 (2017), 2093–2105.

[20] Mohammed A. S. Khalid. 1999. *Routing Architecture and Layout Synthesis for Multi-FPGA Systems*. Ph. D. dissertation, Department of ECE, University of Toronto.

[21] Mokhtar S. Bazaraa, Hanif D. Sherali, and Chitharanjan M. Shetty. 2013. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons.

[22] Stephen Yang, Aman Gayasen, Chandra Mulpuri, Sainath Reddy, and Rajat Aggarwal. 2016. Routability-driven FPGA placement contest. In *Proc. ISPD*. 139–143.

[23] Ümit Çatalyürek and Cevdet Aykanat. 2011. Patoh (partitioning tool for hypergraphs). *Encyclopedia of Parallel Computing* (2011), 1479–1487.

[24] Chak-Wa Pui, Gengjie Chen, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Peishan Tu, Hang Zhang, Evangeline F. Y. Young, and Bei Yu. 2016. RippleFPGA: A routability-driven placement for large-scale heterogeneous FPGAs. In *Proc. ICCAD*. 67:1–67:8.

[25] Gengjie Chen, Chak-Wa Pui, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Evangeline F. Y. Young, and Bei Yu. 2018. RippleFPGA: Routability-driven simultaneous packing and placement for modern FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 10 (Oct. 2018), 2022–2035. DOI: http://dx.doi.org/10.1109/TCAD.2017.2778058

[26] Gurobi Optimization Inc. 2016. Gurobi Optimizer Reference Manual. Retrieved on 01 November, 2019 from http://www.gurobi.com.

Received July 2019; revised October 2019; accepted December 2019