Journal of Circuits, Systems, and Computers
Vol. 22, No. 4 (2013) 1350020 (20 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218126613500205

UNIFICATION OF PR REGION FLOORPLANNING AND FINE-GRAINED PLACEMENT FOR DYNAMIC PARTIALLY RECONFIGURABLE FPGAS*

RUINING HE[†], GUOQIANG LIANG[†], YUCHUN MA[†,§], YU WANG[‡] and JINIAN BIAN[†]

[†]*Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China*

[‡]*Department of Electronic Engineering, Tsinghua University, Beijing, 100084, China*
[§]*myc@tsinghua.edu.cn*

Received 31 March 2012
Accepted 2 November 2012
Published 12 April 2013

Dynamic Partially Reconfiguration (DPR) designs provide additional benefits compared to traditional FPGA application. However, due to the lack of support from automatic design tools in current design flow, designers have to manually define the dimensions and positions of Partially Reconfigurable Regions (PR Regions). The following fine-grained placement for system modules is also limited because it takes the floorplanning result as a rigid region constraint. Therefore, the manual floorplanning is laborious and may lead to inferior fine-grained placement results. In this paper, we propose to integrate PR Region floorplanning with fine-grained placement to achieve the global optimization of the whole DPR system. Effective strategies for tuning PR Region floorplanning and apposite analytical evaluation models are customized for DPR designs to handle the co-optimization for both PR Regions and static region. Not only practical reconfiguration cost and specific reconfiguration constraints for DPR system are considered, but also the congestion estimation can be relaxed by our approach. Especially, we established a two-stage stochastic optimization framework which handles different objectives in different optimization stages so that automated floorplanning and global optimization can be achieved in reasonable time. Experimental results demonstrate that due to the flexibility benefit from the unification of PR Region floorplanning and fine-grained placement, our approach can improve 20.9% on critical path delay, 24% on reconfiguration delay, 12% on congestion, and 8.7% on wire length compared to current DPR design method.

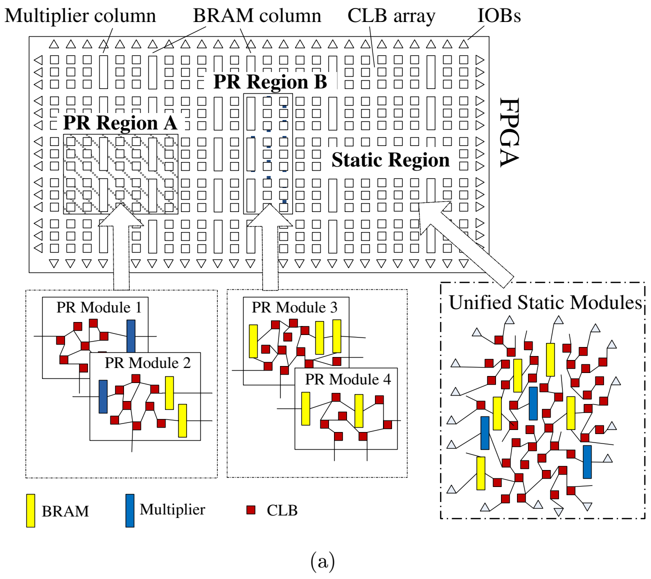
Keywords: Dynamic partial reconfiguration; floorplanning; placement; unification; EAPR.

*This paper was recommended by Regional Editor Eby G. Friedman.

1. Introduction

Modern state-of-the-art FPGA devices like Xilinx Virtex FPGAs[1] support the Dynamic Partially Reconfiguration (DPR) technology. Dynamic partially reconfigurable methodology switches system functionalities online by replacing only certain system modules of the reconfigurable hardware, while the untouched modules continue to operate without interruption. This methodology enhances traditional FPGA applications by reducing size, power, and cost.[21] Figure 1(a) displays the typical layout of Xilinx's FPGAs,[20] with columns of Configurable Logic Blocks (CLBs), Block RAMs (BRAMs), and hardware Multipliers distributed throughout an array of slices. The FPGA fabric is partitioned into two types of regions — static region and Partially Reconfigurable Region (PR Region). Each PR Region is time-multiplexed by a specific group of Partial Reconfiguration Modules (PR Modules), which are swapped in and out this region dynamically.

Though modern technology in the field of FPGA has made DPR available for quite some time, designers are still suffering a great deal from current design methodologies. Commonly used PR designs mostly follow Xilinx's Early-Access PR design flow (EAPR), the state-of-the-art design procedures (as shown in Fig. 1(b)). EAPR requires that PR Regions should be manually defined in terms of shape, size, and physical location. For example, in the case shown in Fig. 1(a), two PR Regions



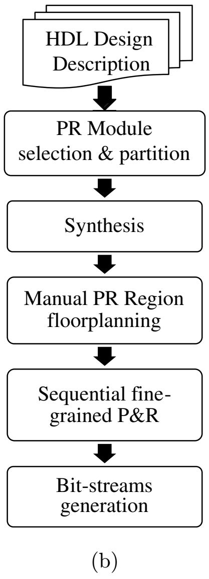

HDL Design Description
PR Module selection & partition
Synthesis
Manual PR Region floorplanning
Sequential fine-grained P&R
Bit-streams generation
(b)

Fig. 1. (a) Diagram of DPR using Xilinx's FPGA. PR Module 1 and PR Module 2 share PR Region A, and PR Module 3 and PR Module 4 share PR Region B. Each PR Module consists of a group of connected components which is represented as a net-list from module synthesis. (b) Diagram of Xilinx's EAPR design flow.

on the FPGA fabric need to be designated for the two groups of PR Modules, respectively. Shape, size, and physical location of each PR Region have to be decided manually. The process of tuning up the above parameters is called floorplanning. This task is challenging due to the following reasons: (1) the designers need to have a good grasp of extensive PR design flow knowledge as well as low-level architectural details of the target FPGA fabric. The manual trial-and-error process is not feasible since the design space grows exponentially with the increase of the number of PR Regions. (2) During the fine-grained placement after PR Region floorplanning, the shape, size, and physical location of each PR Region are taken as fixed parameters. Therefore, the quality of PR Region floorplanning greatly influences the quality of the following fine-grained placement progress. Unfortunately, even the most experienced PR designers usually cannot figure out good PR Region floorplanning solution easily due to the lack of beforehand information about the quality of the following fine-grained placement. In other words, it is hard to guarantee the quality of the PR Region floorplanning. This inspires us to integrate the PR Region floorplanning and the fine-grained placement to build an overall optimization framework.

Besides, the state-of-the-art design flow of DPR systems optimizes the fine-grained placement of PR modules separately. As recommended in Xilinx PR flow,[1] first of all, the most demanding PR Module (in terms of either timing or area) in each PR Region should be chosen to perform the premier placement and routing with static modules. Then the design of static modules will be fixed, which means the placement and routing of static modules are designed according to only a specific subset of PR Modules. Afterwards, the placement and routing of those remaining PR Modules are performed one after another within their corresponding PR Regions. Actually, the sequential design procedures ignore the competition between different PR Modules, and the separated optimization approach can make the design of static module inappropriate to all the other PR Modules. Therefore, the quality of the design's final implementation cannot be guaranteed and the design flow may even fall flat in some cases. For example, as shown in Fig. 2, PR Module A is estimated as the most demanding one. Together with module A, static modules are placed and routed and will be fixed afterwards, as shown in Fig. 2(a). Noticeably, wires from static region are allowed to run across PR Regions.[1] Then the placement and routing of PR Module B are performed with fixed static modules from the previous step. Due to the uneven distribution of available resources, PR Module B may have to be located near to the upper-right corner. Then severe congestion can be generated and the routing of module B has to be detoured which can lead to large delay or even design failure. As a result of the sequential fine-grained placement and routing of PR Modules in existing methods, the degradation of DPR design quality for the whole chip cannot be averted easily. Though the placement for FPGA has been extensively studied,[2–6] none of previous work can handle the simultaneous fine-grained placement for both static modules and various PR Modules in PR Regions.

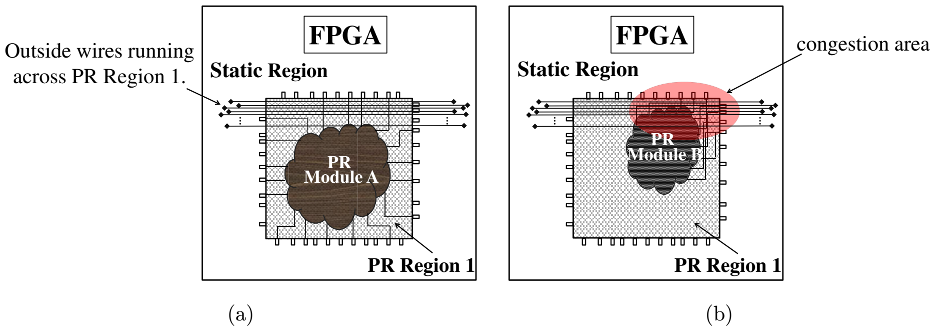


Fig. 2. A sample of infeasible situations when employing EAPR's sequential design method. (a) Fix the static modules and the pin locations of PR Region using the most demanding PR Module A. (b) Congestion occurs due to the highly localized distribution of its available resources and pins when PR Module B is placed within PR Region 1.

In order to overcome the limitation of current design flow, we propose to integrate PR Region floorplanning with fine-grained placement to build a global optimization structure. By our approach, each PR Region can be automatically optimized so that all its corresponding PR Modules can be optimized simultaneously instead of one by one. The key contributions of this paper include:

- **Integrated automatic layout optimization of Partial Reconfiguration Regions and fine-grained placement of system modules.** By performing PR Region floorplanning and fine-grained placement of system modules simultaneously, PR Region floorplanning can be automatically optimized for the whole system with higher quality in terms of size, shape, and physical location. And static modules can be designed for all PR Modules instead of for only a specific subset of PR Modules. This really overcomes the limitation of the separated floorplanning and fine-grained placement used by current design flow and can greatly improve design quality and production efficiency.
- **Effective strategies for tuning up PR Region floorplanning within a two-stage stochastic optimization framework.** Different from traditional FPGA fine-grained placement algorithms, we integrate effective PR Region floorplanning tuning strategies into a two-stage stochastic optimization framework successfully. In this way, PR Regions can be tuned up to explore floorplanning solution space together with the fine-grained placement exploration swiftly and effectively. With different objectives designed in the two separated stages, our framework can ensure design qualities within reasonable run time.
- **Analytical evaluation models customization and multi-objective optimization for DPR designs.** In this paper, we customize some novel analytical models to evaluate not only traditional performance metrics such as wire length and timing delay, but also reconfiguration delay and congestion estimation to

feature DPR designs. The co-optimization of various objectives with additional constraints for DPR designs can be handled simultaneously by our approach.

The remainder of this paper is structured as follows. Section 2 introduces related work. Section 3 formulates the PR Region floorplanning and fine-grained placement problem for dynamic partially reconfigurable FPGA designs. Section 4 introduces our proposed unified PR Region floorplanning and fine-grained placement approach. Experimental results based on 20 largest MCNC FPGA benchmark circuits are given in Sec. 5 and we conclude our current work and discuss future work in Sec. 6.

2. Related Work

Ideas for combined optimizations of floorplanning and placement have been well researched in the ASIC literature.[7,8] But to the best of our knowledge, no such idea has been applied to handle the nontrivial specifics in DPR literature.

For reconfigurable system designs, some relevant work has been done focusing on coarse-grained placement techniques. Early studies[9–12] formulated floorplanning for DPR as a coarse-grained online placement problem of all system modules. They built a three-dimensional template placement model in which each system module was assumed to be an idealized fixed-size block and FPGA fabric was modeled as a homogeneous area. All system modules were PR Modules, and no PR Region needed to be designed (the whole FPGA was regarded as a big PR Region which could accommodate more than one PR Module at the same time). These assumptions make their approach very hard to be applied in practical applications. Similarly, there are also some studies like Refs. 13 and 14 which focus on just-in-time compilation for FPGA designs. But they cannot handle applications with various hardware modules which communicate with each other and share PR Regions fixed during design time on the FPGA fabric. In recent years, Singhal and Bozorgzadeh[15] proposed a multi-layer floorplanner which merges the floorplanning of multiple designs and maximizes the overlap of their common components to achieve benefit from reuse. Banerjee *et al.*[16] introduced a global floorplanning generation method to obtain shared positions for common modules across sub-task instances. Yousuf and Gordon-Ross[17] introduced DAPR, a PR design flow which automated intricate design process of PR designs. DAPR employed a simulated annealing-based floorplanner to create pareto-optimal floorplannings which traded off clock frequency, partial bit-stream size, etc. Although these researches are able to optimize DPR floorplanning with respect to the PR Region aspect ratio, internal fragmentation, routability, etc., the afterward fine-grained placement process is still highly limited by the region constraints generated by floorplanning process. In other words, PR Region floorplanning and fine-grained placement of system modules are still performed sequentially by their approaches. As a consequence, critical performance metrics such as wire length, critical path delay and congestion cannot be optimized for the design as a whole.

Therefore, in this paper, we propose to improve design quality by unifying PR Region floorplanning and fine-grained placement.

3. Problem Formulation

Before describing the problem of PR Region floorplanning and fine-grained placement of system modules, we will first present some concepts that will be used throughout this article.

The target architecture for DPR is a 2D fine-grained heterogeneous FPGA.[1] It consists of an array of computation resources such as Configurable Logic Block (CLB), Block RAM (BRAM), and DSP, as well as I/O pads and routing channels. Each column on FPGA is composed of one kind of computation resource (see Fig. 1(a)). In this paper, we build a 2D Cartesian coordinate system $((L+2) \times (W+2))$ to depict this kind of FPGA as VPR does.[6] IOBs are distributed around the rim of the target FPGA and the remaining area $L \times W$ is occupied by computation resources and routing channels. Each CLB occupies one grid, namely a coordinate which can be represented by a pair of integers. Other computation resource such as BRAM usually occupies more than one grid and we depict its position by the coordinate of the lowest occupied grid.

In Xilinx's DPR technology, reconfiguration bits are grouped into frames of a certain size, called the Reconfigurable Frames (RFs).[1] RF is the minimal reconfiguration granularity supported by DPR FPGA vendors. Correspondingly, the whole FPGA architecture is partitioned into an array of frames with fixed height. For example, a frame on Xilinx's Virtex-4 has a fixed height of 16. To illustrate this structure, Fig. 3 shows an example with the frame height of 6. Note that EAPR requires that each frame should be occupied by one PR Region at most. Therefore, there is an additional geometric constraint (called Reconfiguration Constraint) for

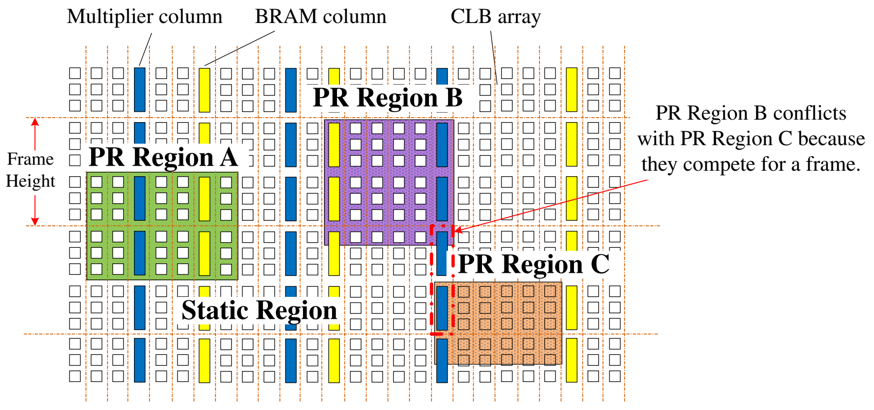


Fig. 3. A portion of the DPR FPGA.

PR Regions that not any two PR Regions are allowed to share a frame, even if only different portions of a frame. Figure 3 shows an infeasible layout of three PR Regions. PR Region B conflicts with PR Region C because there is a frame for which they compete.

In addition to the reconfiguration constraints, reconfiguration delay is a very important factor to consider when designing a DPR system. Actually, reconfiguration delay can be measured in terms of the number of RFs that are partly or fully occupied by PR Regions.[18] In Sec. 4, we will adopt the computation model introduced by Papadimitriou *et al.*[18] to assess the reconfiguration delay of our placement solutions.

Based on the previous introduction, following the state-of-the-art design procedures in DPR, the problem of PR Region floorplanning and fine-grained placement of system modules can be summarized as follows:

After the modular synthesis and packing for a specific application, there will be multiple PR Modules which will share some PR Regions in different time slots on FPGA fabric. There are also some static modules which can be logically unified as a static module since the placement and routing of their logic components are blended together to share the Static Region on the FPGA. Therefore, given a target FPGA architecture ($(L+2) \times (W+2)$) and the detailed computation resources distribution, there are n groups of PR Modules with each group to be mapped to a PR Region on the target FPGA fabric. Assume group i has c_i PR Modules. We need to (1) design each PR Region in terms of shape, size, and physical location, (2) map each PR Module to its corresponding PR Region (specifically, determine the position of each component in the net-list), (3) map the unified static module to the static region (specifically, determine the position of each component in the net-list). (1) corresponds to PR Region floorplanning, (2) and (3) correspond to fine-grained placement introduced earlier. Our objective is not only to minimize wire length and critical path delay, but also to minimize reconfiguration delay and relax congestion of the final solution. During the optimization progress, design constraints which should be handled according to Xilinx's EAPR design flow[1] include:

- **Geometric Constraint.** PR Regions should be rectangular and not overlap with each other. What is more, no frame sharing among PR Regions is allowed, even if they hold its different pieces, as shown in Fig. 3.
- **Resource Constraint.** Each PR Region PR_i must satisfy the maximum resource requirement of every corresponding PR Module.
- **Region Constraint.** Any component of a PR Module must be mapped within the corresponding PR Region. Similarly, any components in the unified static module must be mapped within the static region.

In order to accomplish the automated optimization with the above-mentioned constraints well considered, we propose to integrate PR Region floorplanning with fine-grained placement so that the whole DPR design can be optimized in terms of wire length, critical path delay, reconfiguration delay and congestion at the same time.

4. Unified PR Region Floorplanning and Fine-Grained Placement

In the field of fine-grained placement of FPGA designs, Simulated Annealing (SA)-based algorithms such as VPR[3] have been taking a significant place. In this paper, we propose a novel SA-based approach to handle the nontrivial specifics for DPR designs. Specifically, we need to integrate the designing of PR Region floorplanning into the fine-grained placement process with various constraints considered. Therefore, to obtain the efficient co-optimization, we need not only the effective strategies for tuning up PR Region floorplanning but also the proper analytical evaluation models to evaluate the dynamic partially reconfigurable designs. What is more, an efficient optimization framework is also necessary to speed up the convergence of the whole optimization exploration. In this section, we will introduce the details of our approach in detail.

4.1. *PR region floorplanning tuning*

In this subsection, we introduce our strategies to tune up PR Region floorplanning. Before the layout optimization begins, we calculate the size of each PR Region according to its required computation resources.

4.1.1. *Size calculation of PR region*

Each kind of computation resource's area can be computed by multiplying its width and height (in terms of number of grids). And then we calculate the minimum required area of each PR Region by summing the area of the maximal demand for each resource type:

$$minimum\_area_{PRR} = \sum_{k=1}^{num\_types} (Area_k \times max\_demand_k)\,, \tag{1}$$

where num_types is the number of resource types, $Area_k$ means the area of resource type k, and max_demand_k indicates the maximal resource demand number for resource type k from all the PR Modules in the PR Region. Since the minimal required area might not be large enough to accommodate certain PR Modules due to the uneven distribution of available resources on target FPGA, we enlarge the initial area set for each region by a certain ratio. In this way, routability of the final result can also be improved to a certain degree.[1]

$$area_{PRR} = Enlarge\_ratio \times minimum\_area_{PRR}\,. \tag{2}$$

In this paper, the initial value of $Enlarge\_ratio$ is set to 1.05. If $area_{PRR}$ is so small that no feasible position can be found on the target FPGA, we increase $Enlarge\_ratio$ linearly.

4.1.2. *Reshaping strategy of PR region*

During the stochastic optimization process, each PR Region can be reshaped according to an assigned probability. Since each PR Region must be rectangular according to the Geometric Constraint, we can reshape a PR Region in terms of its aspect ratio. Specifically, for a PR Region (x, y, w, h) which is to be reshaped, we fix its lower-left corner (a, b) as the datum mark. First of all we attempt to obtain a new random value w' for its width in the range of 1 to $area_{PRR}$, and then we set its height h to $h' = \lceil area_{PRR}/w' \rceil$. We randomly reshape the PR Region to guarantee that the annealing process can fully explore the solution space and achieve the optimal solution at length. A sample of reshaping is shown by Fig. 4(a), in which PR Region A has changed from *shape* (a, b, w, h) to *shape*$'$ (a, b, w', h').

It may occur that resources inside the new shape are not enough, or the new shape overlaps with other PR Regions, or illegal frame sharing arises. Figure 4(a) shows a case in which overlap occurs during a reshaping process. In these cases, different random widths w' will be tried until a feasible new shape is found or iteration limit is met. When the new shape satisfies all the constraints, we begin to randomly remap the components of the PR Modules which share the PR Region and the components (from the static module) whose positions are inside the PR Region's new shape. To smooth local perturbation during the stochastic optimization process, as well as to reduce the runtime spent on the random remapping, we minimize the affected regions as much as possible by keeping the mapping of the components inside the overlap region between the old shape and the new shape. We define two regions which are affected by the reshaping process so that the affected area can be minimized:

Definition 1. *Source Region* and *Target Region*: When changing a PR Region's shape or location on the FPGA fabric, the subregion which is no longer occupied by

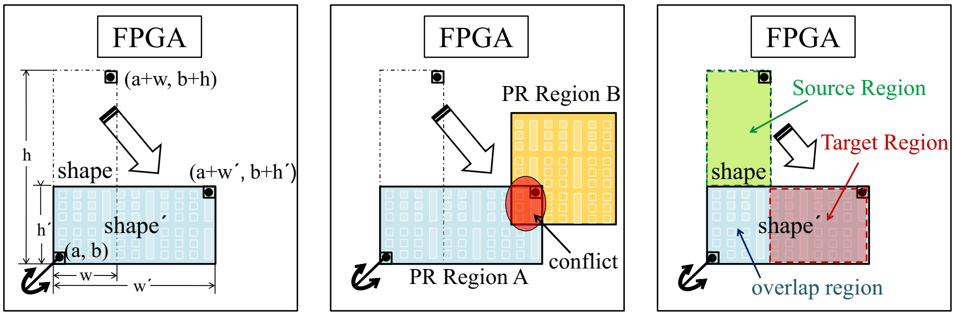


Fig. 4. Reshaping of PR Region A. (a) Reshaping of PR Region A from *shape* to *shape*$'$; (b) The violation of geometric constraint during reshaping; (c) Source Region & Target Region generated after a feasible shape is found.

the new PR Region is called Source Region. The region inside the static region which is affected by this changing is called Target Region.

Figure 4(c) identifies the regions generated by the remapping described by Fig. 4(a). In this case, Source Region is $(a, b + h', w, h - h')$ and Target Region is $(a + w, b, w' - w, h')$. By our approach, we only randomly remap components inside the Source Region and the Target Region. To further smooth local perturbation, components from a same original position will still be remapped to a same new position by our approach. Figure 5(a) describes the whole reshaping process.

4.1.3. *Transferring strategy of PR region*

In addition to the reshaping, each PR Region can be moved on the FPGA fabric according to an assigned probability during the stochastic optimization process. Reshaping and transferring can provide each PR Region with a possibility to go to any possible position on the FPGA with any reasonable shape. When transferring a PR Region (a, b, w, h), we maintain its shape and just randomly change its lower left coordinate to (a', b'). To smooth local perturbation during the stochastic optimization process, we limit the transfer to be local by defining a control parameter

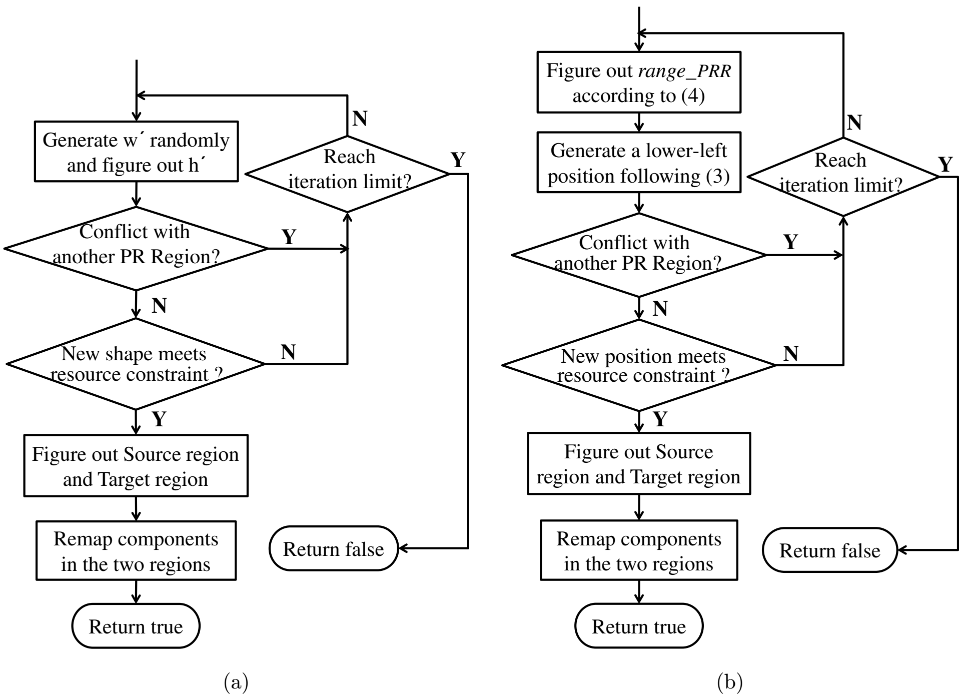

Generate w´ randomly and figure out h´
Reach iteration limit?
N
Y
Conflict with another PR Region?
Y
N
New shape meets resource constraint ?
N
Y
Figure out Source region and Target region
Remap components in the two regions
Return false
Return true
(a)
Figure out range_PRR according to (4)
Generate a lower-left position following (3)
Reach iteration limit?
N
Y
Conflict with another PR Region?
Y
N
New position meets resource constraint ?
N
Y
Figure out Source region and Target region
Remap components in the two regions
Return false
Return true
(b)

Fig. 5. (a) Procedure for reshaping a PR Region; (b) Procedure for transferring a PR Region.

range_PRR. The following constraint must be satisfied when transferring.

$$|a' - a| \leq \mathit{range\_PRR}\,, \quad |b' - b| \leq \mathit{range\_PRR}\,. \tag{3}$$

range_PRR is initially set to be the size of the entire FPGA fabric divided by the average area of all PR Regions. And then we update the value of *range_PRR* dynamically in a similar way VPR updates its range of component swapping.[3]

$$\mathit{range\_PRR}^{\text{new}} = \mathit{range\_PRR}^{\text{old}} * (1 - 0.44 + R^{\text{old}}_{\text{accept}})\,. \tag{4}$$

As introduced in Ref. 3, $R^{\text{old}}_{\text{accept}}$ represents the fraction of attempted moves which were accepted in the previous Markov chain during the stochastic optimization process.

Similar to the discussion of PR Region reshaping, resources within the new PR Region must be enough and the new PR Region must satisfy the geometric constraints. After a new feasible position is found, we also only remap components inside the Source Region and the Target Region to smooth local perturbation and cut down runtime. Figure 5(b) presents the whole process in detail.

4.2. *Analytical evaluation models customization*

Different from the traditional static designs, we need new evaluation models to feature the dynamic partially reconfigurable designs. In this paper, to handle the co-optimization for both PR Regions and static region, novel analytical models to evaluate not only traditional performance metrics such as wire length and timing delay, but also reconfiguration delay and congestion estimation are customized to feature DPR designs.

4.2.1. *Wire length and critical path delay calculation*

Wire length and critical path delay are important metrics to evaluate system performance. But traditional calculation models cannot be directly used to evaluate DPR designs. Since there are multiple PR Modules in each PR Region and at any time only one PR Module can be active in corresponding PR Region, wire length and critical path delay estimation varies depending on different PR Modules.

In our approach, we estimate wire length cost of each PR Region by averaging the bounding box costs of all associated PR Modules. And then we can get the total wire length estimation $\mathit{bb\_cost}_{DPR}$ for the whole design.

$$\mathit{bb\_cost}_{DPR} = \mathit{bb\_cost}_{SR} + \sum_{i=1}^{n} \mathit{bb\_cost}^{\,i}_{PRR}\,, \tag{5}$$

$$\mathit{bb\_cost}^{\,i}_{PRR} = \left(\sum_{k=1}^{c_i} \mathit{bb\_cost}^{\,i,k}_{PRM} \right) \Big/ c_i\,. \tag{6}$$

$bb\_cost^{i,k}_{PRM}$ means the sum of the bounding box costs of PR Module $m_{i,k}$, and bb_cost_{SR} represents the wire length estimation of static region. $bb\_cost^{i,k}_{PRM}$ and bb_cost_{SR} are calculated according to the linear congestion model employed by Ref. 3. c_i is the number of PR Modules in PR Region PR_i. n is the number of PR Regions.

For critical path delay evaluation, we estimate it by the worst case of all the combinations of PR Modules from different PR Regions of the DPR system. Therefore, critical path delay cost td_cost is evaluated by

$$td\_cost_{DPR} = \underset{\substack{1 \le ki \le Ci \\ 1 \le i \le n}}{Max} \left(td(PRM_{1,k1}, PRM_{2,k2}, \ldots, PRM_{n,kn}) \right), \tag{7}$$

where $td(PRM_{1,k1},\ PRM_{2,k2}, \ldots, PRM_{n,kn})$ means the critical path delay when PR Module $PRM_{1,k1}$ occupies the first PR Region, PR Module $PRM_{2,k2}$ occupies the second PR Region, …, PR Module $PRM_{n,kn}$ occupies the nth PR Region. c_i is the number of PR Modules in the ith PR Region. n is the number of PR Regions. This calculation does not take a long time during SA process because we calculate it incrementally as VPR did to update its wire length and critical path delay.[6]

4.2.2. *Reconfiguration delay estimation model*

Reconfiguration efficiency is one of the major issues deserving serious consideration when designing DPR systems. To calculate expected reconfiguration delay, we employ the DPR cost model introduced by Ref. 18. This model takes into account all the physical components that participate in the reconfiguration process. The total reconfiguration delay rd_cost_{DPR} is expressed by the sum of the time spent in each phase of the reconfiguration process:

$$rd\_cost_{DPR} = RD_{SM\text{-}PPC} + RD_{PPC\text{-}ICAP} + RD_{ICAP\text{-}CM}\,, \tag{8}$$

where $RD_{SM\text{-}PPC}$, $RD_{PPC\text{-}ICAP}$, and $RD_{ICAP\text{-}CM}$ represent reconfiguration time spent on three sequential reconfiguration phases separately. *SM-PPC* means the process that PowerPC processor (reconfiguration controller) requests the bit-stream from the external storage and writes it in its local memory; *PPC-ICAP* represents the process that PowerPC transfers the bit-stream word-by-word to the ICAP configuration cache; *ICAP-CM* denotes the process that PowerPC instructs the OPBHWICAP to load the bit-stream to the FPGA configuration memory through the ICAP. In Ref. 18, they introduced detailed procedures to realistically calculate $RD_{SM\text{-}PPC}$, $RD_{PPC\text{-}ICAP}$, and $RD_{ICAP\text{-}CM}$, respectively. Due to limited space, we cannot present their research in much detail but provide the result of their work. The total reconfiguration delay expressed by Eq. (8) is calculated by

$$rd\_cost_{DPR} = fs \times (fn + 1) \times 3.66 \times 10^{-3}\,\text{ms}\,, \tag{9}$$

where *fs* means the size of reconfigurable frame of the target FPGA measured in bytes, while *fn* indicates the number of frames to be reconfigured in DPR design.

In our approach, we calculated reconfiguration delay according to Eq. (9). Specifically, *fn* is computed by adding up the number of frames, whether fully or partially, occupied by all PR Regions. Note that with different aspect ratio or different location which may cause different *fn*, PR Regions even of a same size can bring different reconfiguration delay. *fs* depends on specific FPGA chip. According to Ref. 18, it is equal to 824 bytes for Xilinx XC2VP30, and 164 bytes for all Virtex-4 and Virtex-5 devices. In our model, we assume *fs* to be 164 because we take Virtex-4 board for a case study.

4.2.3. *Congestion evaluation model customization*

In our approach, we also try to relax congestion of DPR designs. This can improve routability of our floorplanning and fine-grained placement result. To accomplish this, we propose a novel congestion evaluation model which is suitable for DPR designs based on the *nonlinear congestion* model adopted by VPR 5.0.2. Our model divides target FPGA into an array of $N \times N$ subregions and records the routing resource demand and supply for each subregion. And in each subregion, two types of routing resource demand are classified: OCC_SR caused by the routes of static region and $OCC\_PR_{i,k}$ caused by the interconnects between components in PR Module $m_{i,k}$. Since routes of different PR Modules sharing a PR Region will not show up simultaneously. Therefore for each PR Region we choose the highest occupation estimation among associated PR Modules as its occupation estimation. For subregion (i, j), occupation is calculated by

$$occupation_{i,j} = OCC\_SR + \sum_{i=1}^{n} max_{k=1}^{c_i} OCC\_PRR_{i,k}\,. \tag{10}$$

And then $congestion_{i,j}$ in subregion (i, j) is calculated according to the penalization rules introduced by VPR 5.0.2. Finally, the *nonlinear congestion* cost of the DPR design nc_cost_{DPR} is calculated by

$$nc\_cost_{DPR} = \sum_{i=1}^{N}\sum_{j=1}^{N} congestion_{i,j}\,. \tag{11}$$

4.3. ***Two-stage stochastic optimization algorithm framework***

In comparison with traditional stochastic optimization FPGA placement algorithms such as VPR, we have extra PR Regions and corresponding groups of PR Modules to handle. Therefore, we need additional operations to generate new solutions during the exploration process. We classify random moves (local perturbations) into the following types:

- **Reshape a PR Region.** There is a specified probability that PR Region can be reshaped in terms of aspect ratio, with constraints considered. When reshaping a

PR Region, we attempt to find a feasible random new shape at most five times. This can improve success rate of reshaping within reasonable runtime.

- **Transfer a PR Region.** There is a specified probability that PR Region's position can also be changed on FPGA fabric, with constraints considered. When transferring a PR Region, we also attempt to find a feasible random new position at most five times to improve success rate of transferring.
- **Swap within static region.** Pairwise exchange between grids, whether empty or not, in static region is performed in a similar way to Ref. 6. We also limit the swap region to the range $[1, \min(W, H)]$ and we update it according to Eq. (12).

$$range^{\text{new}} = range^{\text{old}} * (1 - 0.44 + R_{\text{accept}}^{\text{old}})\,. \tag{12}$$

- **Swap within a PR Region.** Different from the pairwise exchange in static region, we first choose a PR Region PR_i and a PR Module $m_{i,k}$ in PR_i, then choose two grids which have the same resource type in PR_i, and then in the similar way to the pairwise exchange in static region, we exchange the allocation of these two grids for $m_{i,k}$. Swap region is also limited in the same way as the limitation in static region.

The objective of our optimization is to minimize total wire length (WL), timing delay of critical path (TD), reconfiguration delay of the DPR design (RD), and congestion ($Cong$). For the whole SA process, we adopt the temperature schedule employed by VPR 5.0.2, but we increase the number of inner iteration to adapt to DPR design. Our two optimization stages are divided based on temperature.

During the first stage, all types of random moves can be performed according to assigned possibilities. In this stage, floorplanning and fine-grained placement solution is far from the final result. In view of the fact that the *nonlinear congestion* model requires 5× greater CPU time than the bounding box cost function,[19] we do not find much necessity to optimize the routability of DPR design at this early stage. Therefore, three cost components including wire length cost bb_cost_{DPR}, critical path delay cost td_cost_{DPR}, and reconfiguration delay cost rd_cost_{DPR} are modeled and to be optimized in this stage. In the same way VPR 5.0.2 optimizes WL and TD, we optimize the three performance metrics by adding up normalized (5), (7), and (9), and we get our total cost function as follows:

$$cost = \alpha * \frac{bb\_cost_{DPR}}{pre_{bb}} + \beta * \frac{td\_cost_{DPR}}{pre_{td}} + \gamma * \frac{rd\_cost_{DPR}}{pre_{rd}}\,. \tag{13}$$

α, β and γ are coefficients to balance different objectives. We tested different combinations of α, β and γ and found that if they are set to $1/3$ respectively to optimize the three objectives equally, averagely optimal optimization results can be achieved. In the real life, designers may put emphasis on a certain objective by adjusting these coefficients' values. pre_{bb}, pre_{td}, and pre_{rd} are average values in the previous Markov chain in the SA exploration process, as used in VPR 5.0.2.

When the stochastic optimization has been going on for a while, the temperature drops to a threshold and the second stage begins. Because tuning up PR Regions usually means making big changes to current solution, in this stage we speed up the convergence of our approach by fixing PR Region floorplanning and only performing component swappings. Since reconfiguration delay is determined by PR Region floorplanning, reconfiguration delay is a constant in this stage and we do not need to evaluate this item anymore. The optimization in this stage focuses on the fine-grained optimization and the solution is close to the final result, therefore, the congestion in both static region and PR Regions is considered. In this stage, we incorporate *nonlinear congestion* cost *nc_cost* into our evaluation to relax congestion of our final result. By adding up normalized (5), (7), and (11), our total cost function is

$$cost = \alpha' * \frac{bb\_cost_{DPR}}{pre_{bb}} + \beta' * \frac{td\_cost_{DPR}}{pre_{td}} + \gamma' * \frac{nc\_cost_{DPR}}{pre_{nc}} \,. \tag{14}$$

Like we discussed about (13), α', β' and γ' are set to 1/3 respectively in this paper.

5. Experiments

5.1. *Experiment setup*

To test our approach, we construct 20 DPR design test cases meticulously based on the 20 largest MCNC FPGA benchmark circuits. For each MCNC benchmark, we choose around 20%$\sim$35% components as reconfigurable components to compose a set of PR Modules. Each PR Module consists of a group of closely-connected components. And then we divide these PR Modules into several groups with each group to be mapped to a PR Region. During the construction, we try to follow modular design flow and avoid connection between PR Modules in a same group to the best of our ability. Statistics about the 20 DPR test cases are listed in Table 1. "*#components*" and "*#nets*" are the total number of components and nets in the benchmark, respectively. "*#PR*" is the number of PR Regions. "*#PM in each PR*" depicts the number of PR Modules in each PR Region. "*#components in all PMs*" gives the total number of components which are chosen to construct PR Modules. Taking *alu4*, the first test case, for an instance, there are 1544 components and 1536 nets altogether in the original MCNC *alu4* benchmark. 380 components have been chosen carefully following the rules introduced above to construct 9 ([3, 4, 2]) PR Modules to be mapped to 3 PR Regions. All the rest 1164 (= 1544 − 380) components are used to construct the static module. All the test cases are created as the way *alu4* is built. In this paper, our target FPGA fabric is a simulated Xilinx Virtex-4 chip.

To demonstrate the ability of our approach, we simulate the sequential approach employed by Xilinx's EAPR as the control group (EAPR, Ref. 1). Specifically, PR Region floorplanning is created by hand in the first place. Then we select the most

Table 1. Test cases created based on 20 largest MCNC FPGA benchmarks.

Benchmark	#components	#nets	#PR	#PM in each PR	#components in all PMs
alu4	1544	1536	3	[3, 4, 2]	380
apex2	1960	1916	2	[2, 3]	506
apex4	1318	1271	5	[3, 4, 5, 6, 4]	411
bigkey	2559	1936	4	[2, 3, 4, 3]	700
clma	8671	8445	2	[2, 3]	1800
des	2593	1847	4	[3, 3, 4, 3]	600
diffeq	1703	1561	3	[3, 4, 2]	460
disp	2222	1599	4	[3, 5, 3, 4]	420
elliptic	4094	3735	4	[4, 2, 3, 5]	1000
ex5p	1206	1072	5	[2, 3, 4, 3, 4]	333
ex1010	4638	4608	5	[4, 3, 3, 2, 3]	1300
frisc	3828	3576	3	[4, 4, 4]	768
misex3	1453	1411	2	[3, 5]	500
pdc	4687	4591	6	[2, 3, 4, 3, 4, 3]	1050
s298	1951	1935	4	[3, 3, 4, 4]	1000
s38417	6676	6435	3	[2, 3, 4]	1750
s38584.1	7131	6485	4	[4, 3, 2, 2]	1470
seq	1902	1791	3	[4, 2, 3]	550
spla	3814	3706	2	[3, 3]	1110
tseng	1395	1099	3	[3, 2, 3]	420

resource-consuming PR Module in each PR Region and use a slightly-modified VPR 5.0.2 to perform the fine-grained placement for these selected PR Modules and static modules. All models and parameters from the original VPR are retained and we only modify VPR 5.0.2 to support the feature that components from a PR Module can only exchange position within the corresponding PR Region. Finally, we use the original VPR to perform fine-grained placement of the rest PR Modules within their corresponding PR Region one by one. Time consumed by this sequential approach is calculated by adding up the time spent on the sequential fine-grained placement of the whole design. Note that we assume the time consumed by manual floorplanning as 0, which gives sequential approach an unfair edge. The experimental group is the approach proposed by this paper.

All algorithms are compiled and run by a single thread under Red Hat OS on Intel(R) Xeon(R) CPU E5620 @2.40 GHz Server with 12 GB (8G free) of RAM.

5.2. *Experimental results*

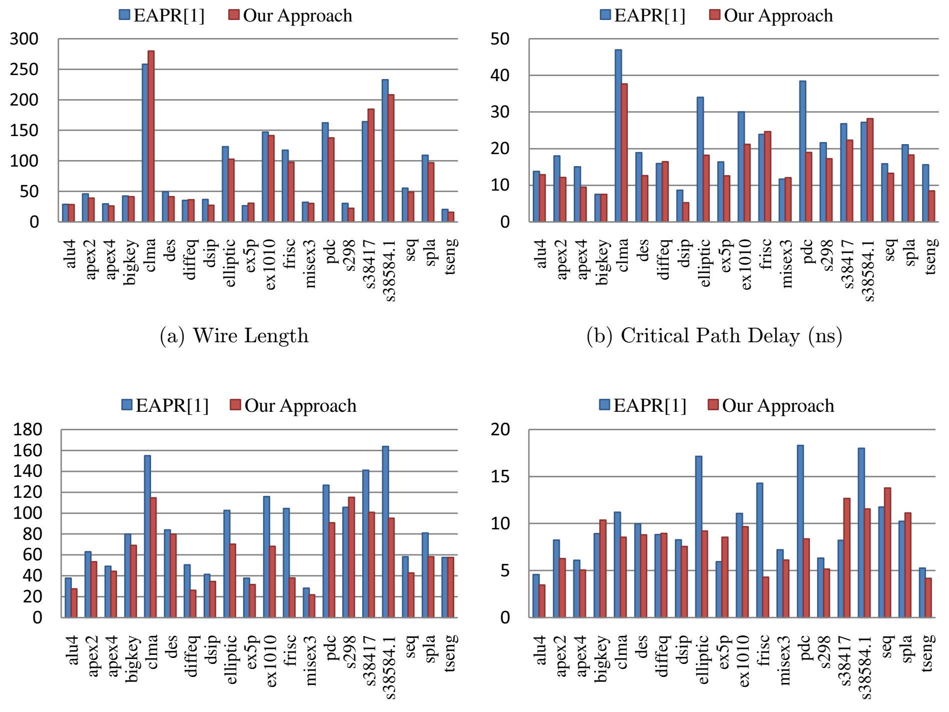
Experimental results of our approach and the sequential method used by Xilinx's EAPR[1] on the 20 largest MCNC FPGA benchmarks are shown in Table 2. *WL* refers to total wire length estimation, *TD* represents timing delay estimation of the critical path, *Cong* means congestion estimation, and *RD* indicates reconfiguration delay. Statistics demonstrate that in addition to the ability to design and optimize PR Regions automatically, our proposed approach can also improve design qualities successfully. Specifically, our approach reduces total wire length by 8.7%, optimizes

Table 2. Experimental results on 20 largest MCNC FPGA benchmark circuits.

	EAPR[1]					Unified Floorplanning and Placement				
Benchmark	*WL*	*TD* (e-8s)	*RD* (ms)	Cong	Time (min)	*WL*	*TD* (e-8s)	*RD* (ms)	Cong	Time (min)
alu4	28834	1.3793	37.815	4.565	0.91	28294	1.2881	27.611	3.466	1.72
apex2	45996	1.8043	63.025	8.240	1.37	39195	1.2132	53.490	6.260	2.86
apex4	29647	1.5026	49.220	6.100	0.81	26058	0.9457	44.418	5.050	1.37
bigkey	42387	0.7536	79.832	8.915	1.92	41526	0.7514	69.134	10.368	3.21
clma	258138	4.6950	154.862	11.213	18.15	279903	3.7652	114.646	8.548	38.52
des	49191	1.8884	84.034	9.974	1.71	41212	1.2652	79.832	8.792	2.87
diffeq	35300	1.5940	50.420	8.818	1.08	36451	1.6450	26.231	8.954	2.10
dsip	36674	0.8651	41.417	8.270	1.34	27192	0.5239	34.521	7.567	2.47
elliptic	123268	3.4035	102.641	17.135	5.48	102766	1.8218	70.207	9.215	9.86
ex5p	26523	1.6352	37.815	5.931	0.65	30576	1.2551	31.635	8.550	1.18
ex1010	147006	3.0044	115.846	11.069	7.12	141304	2.1146	68.180	9.670	14.56
frisc	117747	2.3898	104.441	14.284	4.42	97817	2.4640	37.921	4.298	9.00
misex3	32244	1.1706	28.211	7.2170	0.88	30182	1.2044	21.948	6.130	1.63
pdc	162179	3.8403	126.651	18.316	7.31	137609	1.8965	90.780	8.357	14.12
s298	30367	2.1624	105.642	6.330	1.90	22245	1.7266	115.246	5.165	2.82
s38417	164379	2.6787	141.056	8.213	12.89	184701	2.2343	100.840	12.664	26.62
s38584.1	232802	2.7190	163.866	18.009	14.12	208182	2.8200	95.216	11.565	29.87
seq	55283	1.5865	58.223	11.745	1.477	48900	1.3288	42.736	13.776	2.68
spla	109237	2.1038	81.032	10.232	4.78	96999	1.8244	58.223	11.135	9.81
tseng	20453	1.5622	57.623	5.262	0.71	15965	0.8493	57.623	4.169	0.72
Avg. ratio	1	1	1	1	1	0.913	0.791	0.760	0.880	1.85

critical path delay by 20.9%, lowers reconfiguration delay by 24%, and improves congestion estimation by 12% on average. Figure 6 presents the straightforward comparison between EAPR and our proposed approach for different test cases in terms of wire length, critical path delay, reconfiguration delay and congestion estimation.

Figure 7 presents the scalability analysis of our approach. In this figure, benchmarks are ranked in ascending order of problem size (Note that in Table 2 and Fig. 6 benchmarks are ranked alphabetically). Figure 7(a) shows runtime comparison between current design method and our proposed approach. From Fig. 7(a) we can find that for most of the MCNC benchmarks, our approach can finish the simultaneous optimization of wire length, critical path delay, and reconfiguration delay, and the relaxation of nonlinear congestion within 10 min. For the largest benchmark *clma*, our approach takes less than 40 min. Figure 7(b) shows the ratio of the runtime of our proposed approach to the runtime required by EAPR. On average, we can achieve the co-optimization with reasonable 85% additional runtime. With different number of PR Regions and PR Modules of different sizes, the runtime ratio converges at two approximately. In view of the fact that our approach can finish PR Region designing automatically with higher quality, compared to the laborious, time-consuming manual PR Region designing required by EAPR, our approach is efficient and feasible.



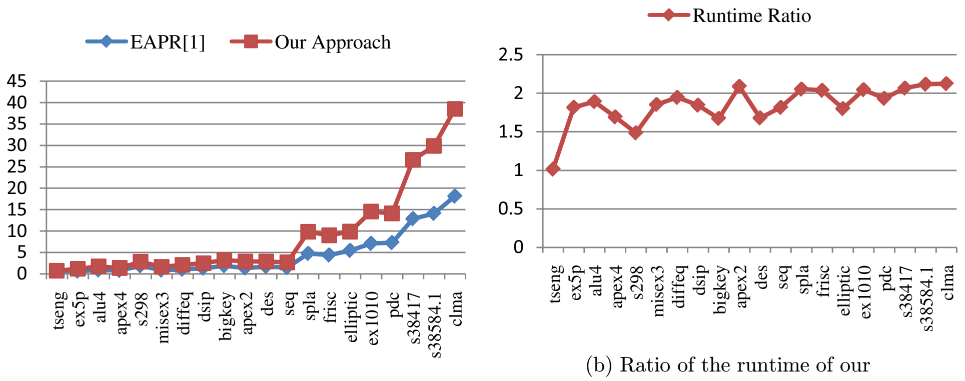
(a) Wire Length

(b) Critical Path Delay (ns)

(c) Reconfiguration Delay (ms)

(d) Nonlinear Congestion

Fig. 6. Improvement in wire length, critical path delay, reconfiguration delay, and nonlinear congestion from the unification of PR Region floorplanning and fine-grained placement.



(a) Runtime (minute).

(b) Ratio of the runtime of our approach to EAPR.

Fig. 7. Evaluation of the scalability of our approach. Benchmarks are ordered in ascending order of problem size. Note that we assume EAPR takes no time for manual PR Region floorplanning, which actually consumes much time. This gives EAPR an unfair edge when comparing runtime.

6. Conclusion and Future Work

Though DPR FPGAs have shown the potential to provide low cost and high performance designs, extensive manual floorplanning and inefficient sequential fine-grained placement are still constraining the application of DPR technology. In this paper, we propose to unify PR Region floorplanning and fine-grained placement to build a global optimization design flow. A novel tow-stage stochastic optimization framework and effective techniques to handle PR Region floorplanning and fine-grained placement are simultaneously presented. Efficient analytical models are also customized to evaluate wire length, critical path delay, congestion, and reconfiguration delay. Based on a simulated Xilinx Virtex-4 FPGA fabric, experimental results on the 20 largest MCNC benchmarks show that our approach can reduce total wire length by 8.7%, optimize critical path delay by 20.9%, lower reconfiguration delay by 24%, and improve congestion estimation by 12% compared to state-of-the-art design method. Since DPR involves many other related issues such as task scheduling and HW/SW division, etc., we will work on more design issues to improve DPR design quality in our future work.

Acknowledgments

This work was supported in part by NSFC 61076035, 60876030 and 61106030.

References

1. Xilinx, Partial Reconfiguration User Guide, 1 March 2011.
2. L. Cheng and M. D. F. Wong, Floorplan design for multi-million gate FPGAs, *Proc. ICCAD* (2004), pp. 292–299.
3. V. Betz and J. Rose, VPR: A new packing, placement and routing tool for FPGA research, *Proc. FPL* (1997), pp. 213–222.
4. Y. Sankar and J. Rose, Trading quality for compile time: Ultra-fast placement for FPGAs, *Proc. FPGA* (1999), pp. 157–166.
5. H. Bian *et al.*, Towards scalable placement for FPGAs, *Proc. FPGA* (2010), pp. 147–156.
6. J. Luu *et al.*, VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling, *Proc. FPGA* (2009), pp. 133–142.
7. S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa and I. L. Markov, Unification of partitioning, placement and floorplanning, *Proc. ICCAD* (2004), pp. 550–557.
8. J. Z. Yan, N. Viswanathan and C. Chu, Handling complexities in modern large-scale mixed-size placement, *Proc. DAC* (2009), pp. 436–441.
9. K. Bazargan *et al.*, Fast template placement for reconfigurable computing systems, *IEEE Design & Test of Computers* (2000), pp. 68–83.
10. A. Ahmadinia and J. Teich, Speeding up online placement for XILINX FPGAs by reducing configuration overhead, *Proc. VLSI-SOC* (2003), pp. 118–122.
11. H. Walder *et al.*, Fast online task placement on FPGAs: Free space partitioning and 2D-hashing, *Proc. IPDPS*, Munich, Germany (2003), pp. 178b.
12. A. Ahmadinia *et al.*, A new approach for on-line placement on reconfigurable devices, *Proc. IPDPS*, Santa Fe, New Mexico, USA (2004), pp. 134–140.

13. H. Sidiropoulos *et al.*, On supporting efficient partial reconfiguration with just-in-time compilation, *19th Reconfigurable Architectures Workshop (RAW)*, May 2012, China.
14. R. Lysecky, F. Vahid and S. Tan, Dynamic FPGA routing for just-in-time FPGA compilation, *Proc. DAC* (2004), pp. 954–959.
15. L. Singhal and E. Bozorgzadeh, Multi-layer floorplanning on a sequence of reconfigurable designs, *Proc. FPL* (2006), pp. 1–8.
16. P. Banerjee *et al.*, Floorplanning for partially reconfigurable FPGAs, *IEEE Trans. CAD Integr. Circuits Syst.* **30** (2011) 8–17.
17. S. Yousuf and A. Gordon-Ross, DAPR: Design automation for partially reconfigurable FPGAs, *Proc. ERSA* (2010), pp. 97–103.
18. K. Papadimitriou *et al.*, Performance of partial reconfiguration in FPGA systems: A survey and a cost model, *ACM Trans. Reconfigurable Technol. Syst.* **4** (2011).
19. V. Betz and J. Rose, Directional bias and non-uniformity in FPGA global routing architectures, *Proc. ICCAD* (1996), pp. 652–659.
20. B. Blodget *et al.*, Partial and dynamically reconfiguration of Xilinx Virtex-II FPGAs, *Proc. FPL* (2004), pp. 801–810.
21. S. Craven and P. Athanas, Dynamic hardware development, *Int. J. Reconfigurable Comput.* **2008** (2008) 10, doi:10.1155/2008/901328.