Parallel FPGA Routers with Lagrange Relaxation

ROHIT AGRAWAL[1][§], KAPIL AHUJA[2][§], DHAARNA MAHESHWARI[2], MOHD UBAID SHAIKH[2], MOHAMED BOUAZIZ[3], AND AKASH KUMAR[4] (Senior Member, IEEE)

[1]Computer Science & Engineering, Madhav Institute of Technology & Science, Gwalior 474005, MP, India; (e-mail: agarwal0115@gmail.com)

[2]Math of Data Science & Simulation (MODSS) Lab, Computer Science & Engineering, Indian Institute of Technology Indore, Simrol 453552, MP, India; (e-mail: kahuja@iiti.ac.in (K. A.); dhaarna1999@gmail.com (D. M.); shaikhubaid769@gmail.com (M. U. S.))

[3]Tunisia Polytechnic School, La Marsa 1053, Tunisia; mohamed.bouaziz@kaust.edu.sa

[4]Center for Advancing Electronics, Technische Universität Dresden, Dresden 01062, Germany; akash.kumar@tu-dresden.de

[§] Rohit Agrawal and Kapil Ahuja are equal contributors to this work and designated as co-first authors.

Corresponding author: Akash Kumar (e-mail: akash.kumar@tu-dresden.de).

ABSTRACT Routing of the nets in Field Programmable Gate Array (FPGA) design flow is one of the most time consuming steps. Although Versatile Place and Route (VPR), which is a commonly used algorithm for this purpose, routes effectively, it is slow in execution. One way to accelerate this design flow is to use parallelization. Since VPR is intrinsically sequential, a set of parallel algorithms have been recently proposed for this purpose (ParaLaR and ParaLarPD). These algorithms formulate the routing process as a Linear Program (LP) and solve it using the Lagrange relaxation, an adapted sub-gradient method, and a Steiner tree algorithm. When tested on the MCNC benchmark circuits, using underlying VPR 7.0 for packing and placement, ParaLaR and ParaLarPD both outperformed VPR 7.0 for routing, with ParaLarPD being better.

We have three main contributions here. Recently, in 2020, a new variant of VPR, i.e. VPR 8.0, has been proposed. Hence, first, we make ParaLarPD compatible for testing on MCNC benchmark circuits using VPR 8.0. Second, we adapt ParaLarPD for the larger benchmark circuits than MCNC, i.e., VTR, using both VPR 7.0 and VPR 8.0, and perform thorough evaluation. Finally, and third, we improve ParaLarPD further. We design a family of Lagrange heuristics that better the Lagrange relaxation process of ParaLarPD. We term our new algorithm ParaLarH and test it on both the benchmark circuits (MCNC and VTR) and using both the VPRs (VPR 7.0 and VPR 8.0).

When tested on MCNC and VTR benchmark circuits, VPR (VPR 7.0 and VPR 8.0) is outperformed by both ParaLarH and ParaLarPD, with average gains given below. The minimum channel width improvements are 22% and 12%, respectively. The total wire length improvements for both are 45%. Finally, the average critical path delay improvements for both are almost the same (37% and 35%, respectively).

INDEX TERMS FPGA, Lagrangian Heuristics, LP, Optimization, Subgradient Methods.

I. INTRODUCTION

The Electronic Design Automation (EDA) process has been the single biggest factor behind the thriving of the semiconductor industry in the last fifty years. However, it is very time consuming with routing taking a big percentage of this time. In this paper, we focus on a large subset of this problem, i.e. the expensive Field Programmable Gate Array (FPGA) [1], [2] routing process. FPGA routing is computationally expensive because the common standard algorithm to perform routing, i.e., Versatile Place and Route (VPR [3]) is

intrinsically slow. One way to accelerate routing is to exploit parallelization capabilities of the modern High Performance Computing (HPC) machines. Since VPR is fundamentally sequential, new parallel routing algorithms need to be developed.

One of the first attempts in parallelizing this routing process was done in [4]. Here, the authors formulated the problem as a Binary Integer Linear Program (BILP), applied the Lagrange relaxation to eliminate constraints, and then solved the resulting optimization problem using the sub-gradient

method and a Steiner tree algorithm. The final algorithm was termed as ParaLaR. When tested on the MCNC [5] benchmark circuits using underlying VPR*, ParaLaR substantially outperformed VPR.

In one of our recent works [6], we substantially improved the constraints violation drawback of ParaLaR. We achieved this by developing a more problem specific version of the sub-gradient method and fine tuning the size of its iterative step. The final algorithm was termed as ParaLarPD. When again tested on the MCNC [5] benchmark circuits using underlying VPR, ParaLarPD gave bigger gains over VPR as compared to the gain given by ParaLaR over VPR.

We have three fold contribution in this work.

1) Recently (in 2020), a new variant of VPR has been proposed. That is, VPR 8.0. Since earlier, we have tested ParaLarPD on MCNC benchmark circuits using only VPR 7.0. We now take a step further and make ParaLarPD compatible for testing on same circuits but while using VPR 8.0. As mentioned earlier, VPR is used to pack and place before routing as well as is compared against.

2) Earlier, we have experimented with ParaLarPD only on the MCNC benchmark circuits, which are considered small. Hence, we adapt ParaLarPD for larger benchmark circuits of VTR as well. We give thorough results for both VPR 7.0 and 8.0.

3) Although ParaLarPD reduced the constraints violation of ParaLaR, it did not completely eliminate it. Hence, we also design a family of Lagrange heuristics to improve the Lagrange relation process in-turn reducing the constraints violation in ParaLarPD further. We term our new algorithm as ParaLarH. We evaluate ParaLarH on both the benchmark circuits (MCNC and VTR) when using both the VPRs (VPR 7.0 and VPR 8.0).

When experimented on MCNC and VTR benchmarks, the average gains over VPR 7.0 and VPR 8.0 are as follows.

- The minimum channel width: ParaLarH and ParaLarPD achieve 21.72% and 12.24% improvements, respectively.
- The total wire length: ParaLarH and ParaLarPD achieve 44.89% and 44.67% improvements, respectively.
- The average critical path delay: ParaLarH and ParaLarPD achieve 37.37% and 35.18% improvements, respectively.

As evident above, ParaLarH and ParaLarPD both perform well with ParaLarH being better. Extra work done in designing the Lagrange heuristics in ParaLarH leads to slight increase in total running time as compared to ParaLarPD. This can be offsetted by running code in parallel. A parallel code would lead to faster ParaLarPD as well but ParaLarPD would not be able to improve other routing metrics as above.

---

*Besides being a routing algorithm, VPR is also used to pack and place before other routing algorithms are applied

The rest of this paper has four more sections. In Section II, we present the ParaLarPD algorithm from [6]. Our Lagrange heuristic, its variants, and the resulting algorithm of ParaLarH are discussed in Section III. In Section IV, we present the experimental results. Finally, conclusions and future work are given in Section V.

II. BACKGROUND

The routing problem in FPGA or a electronic circuit is formulated as a weighted grid graph $G(V, E)$, where V and E are the sets of certain vertices and edges, respectively, and there is a cost associated with each edge [4], [6]. In this grid graph, we have three types of vertices; the net vertices, a Steiner vertices, and the other vertices. A net is represented as a set $N \subseteq V$ consisting of net vertices with other types of vertices playing a supporting role.

Here, the goal is to find a route for each net such that the union of all the routes will minimize the total path cost of the graph G, which is directly proportional to the total wire length of FPGA. To achieve this objective, the problem of routing of nets is formulated as an LP problem given by [4] (ParaLaR paper).

$$\min_{x_{e,i}} \sum_{i=1}^{N_{nets}} \sum_{e \in E} w_e x_{e,i}, \tag{1}$$

$$\text{Subject to } A_i x_i = b_i, i = 1, 2, ..., N_{nets}, \tag{2a}$$

$$x_{e,i} = 0 \ or \ 1, \text{and} \tag{2b}$$

$$\sum_{i=1}^{N_{nets}} x_{e,i} \leq W, \forall e \in E \tag{2c}$$

with meaning of each variable is given in Table 1. The equality constraints guarantee that a valid route is formed for each net (these are implicitly satisfied by our solution). The inequality constraints are the channel width constraints that restrict the number of nets utilizing an edge to W. These constraints also relate to our other complementary requirement, that is, the minimization of the channel width of each edge (achieved by an iterative reduction in the solution process).

TABLE 1: Summary of the symbols with their meanings as used in LP (1)-(2c).

Symbols	Meaning
$x_{e,i}$	The binary decision variables that can have value either 0 (if net i does not utilize an edge e) or 1 (if net i utilizes an edge e)
N_{nets}	The number of nets
E	The set of edges with e denoting one such edge
w_e	The cost/ time delay associated with the edge e
W	A constant (input and iteratively reduced)
A_i	The node-arch incidence matrix (the constraints matrix of the minimum cost flow problem)
x_i	The vector of all $x_{e,i}$ that represents the route of the ith net
b_i	The demand/ supply vector, which signifies the amount of cost flow to the ith net

The inequality constraints need to be relaxed or eliminated. This is because they introduce dependencies between the

routing of different nets leading to the difficulty in solving the LP in a parallel manner. The Lagrange relaxation [7] is a technique where the constraints can be eliminated by integrating them into the objective function. This introduces Lagrange multipliers λ_e for each constraint, with relaxation carried out by adding λ_e times the corresponding constraint to the objective function. That is, instead of the LP given in (1)-(2c), we have the following [4] (again ParaLaR paper):

$$\min_{x_{e,i},\,\lambda_e} \left(\sum_{i=1}^{N_{nets}} \sum_{e \in E} \left(w_e + \lambda_e \right) x_{e,i} - W \sum_{e \in E} \lambda_e \right), \quad (3)$$

$$\text{Subject to } A_i x_i = b_i, \ i = 1, 2, ..., N_{nets}, \quad (4a)$$

$$x_{e,i} = 0 \ or \ 1 \quad \text{and} \quad (4b)$$

$$\lambda_e \geq 0. \quad (4c)$$

In the above LP, $\left(w_e + \lambda_e \right)$ is the new cost associated with the edge e. As earlier, this LP can be easily solved in a parallel manner.

In (3)–(4c), we have two sets of variables $x_{e,i}$ and λ_e. Since the decision variables $x_{e,i}$ can have values either 0 or 1, and $\lambda_e \in \mathbb{R}$ (or real line), this LP is a Binary Integer Linear Program (BILP) that is non-differentiable [8]–[11]. Hence, the traditional methods such as the Simplex method [12], the interior point method [13], etc. fail here. The sub-gradient based methods [14], [15] are iterative methods for solving optimization problems [16]–[18] without stringent differentiability requirements. In these methods, the variable (say x) is updated as $x^{k+1} = x^k - \alpha^k g^k$, where α^k is the step size, g^k is a sub-gradient of the objective function, and the superscript (k or $k+1$) denotes the iteration number. Since a sub-gradient based algorithm will not give binary solutions, which we need (recall $x_{e,i}$ can be 0 or 1), we use it to compute the Lagrange multipliers λ_e only. For solving $x_{e,i}$, we use a minimum Steiner tree algorithm.

There are many variants of the sub-gradient based methods available such as the projected method [14], the primal–dual method [19], the conditional method [20], the deflected method [20], etc. In our ParaLarPD algorithm [6], which as earlier improved the ParaLaR algorithm [4], we demonstrated the superiority of using the primal–dual method with computation of the Lagrange multipliers done as below.

$$\lambda_e^{k+1} = \lambda_e^k + \alpha^k \max \left(0, \sum_{i=1}^{N_{nets}} x_{e,i} - W \right), \quad (5)$$

where $\sum_{i=1}^{N} x_{e,i} - W$ is a sub-gradient of the objective function at the kth iteration–the partial derivative of the objective function in (3). Also λ_e^0 is taken as zero for all edges.

In our ParaLarPD paper [6], we also proposed a new step size updation strategy that works better than the corresponding technique proposed in the ParaLaR paper [4]. That is,

$$\alpha^k = (1/k) / \left\| T^k \right\|_2, \quad (6)$$

where k is the iteration number, T^k is the Karush–Kuhn–Tucker (KKT) operator of the objective function (3), and $\left\| T^k \right\|_2$ is the 2-norm of T^k.

Next, a minimum Steiner tree algorithm [21] is used to compute $x_{e,i}$. Here, the input is a set S that contains the net vertices. The intermediate goal is to compute the set of Steiner vertices for S, which is initially empty (say U). The algorithm begins by forming a triple of vertices from S. Next, a possible candidate Steiner vertex is found such that the total path cost from the vertices in the triple to the candidate vertex is minimized. This process is repeated for all the sets of triples to find the possible Steiner vertices, out of which U is formed. Finally, the union of S and U is obtained using the minimum spanning tree algorithm leading to a minimum Steiner tree. The edges that are used in this tree have $x_{e,i} = 1$ and all other edges have $x_{e,i} = 0$.

After one complete iteration of the primal–dual sub-gradient algorithm as well as a Steiner tree algorithm, the value of W is reduced and these steps are repeated. This helps us obtain a better local minima both for the total wire length and the channel width. For easy reference the pseudo code of ParaLarPD, as published in [6], is given in **Algorithm 1**.

Algorithm 1 ParaLarPD [6]

Input: Architecture description file and benchmark file.
Output: Route edges.
1: Run VPR with the input architecture and benchmark circuit.
2: $steiner\_points \leftarrow \emptyset$
3: $grid\_graph \leftarrow$ InitGridGraph()
4: $\lambda_e = 0, \forall e \in E$
5: **for** $iter = 1$ **to** max_iter **do**
6:     Calculate the step size α using (6).
7:     $route\_edges \leftarrow \emptyset$
8:     **parallel_for** $i = 1$ **to** N_{nets} **do**
9:         $points \leftarrow \left\{ p : p \in \{\text{source and sinks of } i\text{th net}\} \right\}$
10:         **if** $iter == 1$ **then**
11:             $steiner\_points[i\text{th } net] \leftarrow$
                Min_Span_Tree($grid\_graph, points$)
12:         **end if**
13:         $route\_edges[i\text{th } net] \leftarrow$
            Min_Span_Tree($grid\_graph, steiner\_points[$
            $i\text{th } net] \cup points$)
14:     **end parallel_for**
15:     **while** $e \in E$ **do**
16:         Update Lagrangian relaxation multipliers λ_e using the Equation (5).
17:         Update the edge weight of the $grid\_graph$ on $route\_edge$. New edge weights are $w_e + \lambda_e$.
18:     **end while**
19: **end for**

III. PROPOSED APPROACH

As mentioned earlier, in our proposed work we *first* perform FPGA routing using our ParaLarPD. Since some constraints are often violated by the obtained solution, *second*, we develop a heuristic that converts the infeasible solution to a feasible one (i.e. tackle the issue of the constraints violation), which is discussed next.

This technique has been applied successfully in many domains [22]–[27]. For example, [22] solves a multi-plant
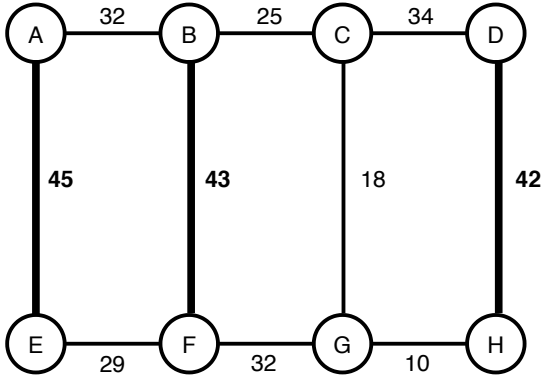
FIGURE 1: A sub-graph to demonstrate working of our heuristic strategy.

lot-sizing problem. Here, the authors formulate an LP to minimize the production costs with the demand and the capacity constraints. The constraints are relaxed by introducing the Lagrangian multipliers. A novel Lagrangian heuristic (in the form of two feasibility stages) is applied to every solution obtained while solving for the multipliers. The first feasibility stage consists of a local search in which the production lots are transferred amongst the time periods to ensure feasible solutions. The second feasibility stage is also a local search based strategy, however, in this, viable solutions are explored by transferring the production batches to not only the different time periods but to the different plants as well.

Another example is [23] where assignment of the students to the classes (based upon their preferences) is formulated as a graph partitioning problem with the capacity constraints. This problem is further modeled as a Quadratic Program (QP), and similar to [22], the constraints are relaxed by introducing Lagrange multipliers, which are solved by the sub-gradient method. As expected, the obtained solutions are not necessarily feasible, and hence, a Lagrange heuristic is built. In this, the constraints violation are assigned probabilities based upon certain characteristics of the solution. The algorithm is again iterated with the new probability based information.

Our basic Lagrangian heuristic to remove the constraints violation in ParaLarPD consists of the five steps as below. Here, we initially explain these steps using the example shown in Figure 1, and then in the form of an algorithm. In this example, the channel widths as computed by ParaLarPD are written next to the corresponding edge in the figure. Since W is taken as forty, we have three edges where the constraints violation occur. That is AE, BF, and DH that are highlighted in bold in Figure 1.

1) Pick an edge with the constraints violation, and find a new alternate path between the nodes of this edge using any path finding algorithm. There may be many alternative paths possible so pick any one. If the new path contains an edge that already has the constraints violation, then drop it and move to the next alternative

path.

For example here, without loss of generality, the edge picked is BF and the first alternate path chosen is $BA \rightarrow AE \rightarrow EF$. Since this path contains the edge AE, which violates the constraints, and hence, we drop it and pick the next possible path ($BC \rightarrow CG \rightarrow GF$) where no such violation occurs.

2) Next, compute the available capacity of each edge in the new path to route more nets without the constraints violation. Minimum of these capacities is termed as Threshold, and used further. Mathematically,

$$\text{Threshold} = \min(W - \sum_{i=1}^{N_{nets}} x_{e_k,i})$$
$$\forall k \in \{\text{edges in the new path}\}.$$

For our example, the value of Threshold is 8.

3) Calculate the amount of violation $d = \sum_{i=1}^{N_{nets}} x_{e,i} - W$ for the edge under consideration e. Further, calculate the number of nets where the constraints violating edge needs to be replaced by the selected new path. This is computed as

$$q = \min(\text{Threshold}, d) \qquad (7)$$

so that no edge in the added new path has the constraints violation.

For the edge under consideration (BF), $d = 43 - 40 = 3$, and hence, $q = \min(8, 3) = 3$.

4) Finally, replace this edge under consideration with the selected path in q number of nets.

In this example, this corresponds to replacing BF with $BC \rightarrow CG \rightarrow CF$ in 3 nets.

5) If in (7) above, Threshold $< d$, then we would have not completely eliminated the constraints violation in the edge under-consideration. In this case, the search for the alternate path needs to resumed from the start until the violation is completely eliminated or no such path exists.

We repeat the above steps for all the edges that are violating the constraints. This violation is directly related to the minimum channel width (discussed earlier), i.e. we improve this requirement as well. **Algorithm 2** describes our heuristic design in an algorithmic form. The points above map to the respective line numbers in the algorithm, which is termed as ParaLarH. For enhanced clarity, we describe ParaLarH via a data flow diagram as well (in Figure 2).

A. OTHER VARIATIONS OF OUR HEURISTIC

Next, we discuss some variants of ParaLarH. As mentioned earlier, these variations are designed to help reduce the constraints violation further, however, they do negligibly increase the computational cost of the overall algorithm.

(i) The first variant is based upon the fact that there may exist multiple paths between any two end points, and in Step 1 above we should pick the one that gives the best results. Hence, instead of picking just one path

---

Algorithm 2 Heuristic Design

Input: Set of nets and edges that are being used; and the decision variables determined by ParaLarPD algorithm.
Output: Updated set of nets and edges that are being used.

    for (each edge $e \in E$) **do**
        while ($d = \sum_{i=1}^{N_{nets}} x_{e,i} - W \geq 0$) **do**
          1) Find a path using any path finding algorithm p : $e_1 e_2 \cdots e_{r-1} e_r$ between the end points of the edge e such that

$$e_1.start = e.start,$$
$$e_j.end = e_{j+1}.start \ \forall j \in \{1, 2, \ldots, r-1\},$$
$$e_r.end = e.end,$$
$$\sum_{i=1}^{N_{nets}} x_{e_j,i} \leq W \ \forall j \in \{1, 2, \ldots, r\},$$

            If there is no such alternative path available for the current constraint violating edge, then break.
          2) Compute

$$\text{Threshold} = \min(W - \sum_{i=1}^{N_{nets}} x_{e_k,i}) \ \forall k \in \{1, 2, \ldots, r\}.$$

          3) Calculate $q = \min(\text{Threshold}, d)$.
          4) If $Net^e = \{N_{nets}^1, N_{nets}^2, \ldots, N_{nets}^t\}$ denotes the t nets where edge e is used. Replace e with path $e_1 e_2 \cdots e_r$ in q such nets. Usually $t > q$.
          // The Point 5 as discussed in text maps to the `while` statement above.
        end while
    end for

---

randomly, we pick β number of paths. Further, we perform Steps 1, 2 and 3 for all these β paths.

(ii) In the second variant, in Step 4 above we begin by sorting the t nets where the edge under consideration e is used. This sorting is done in the increasing order of the number of new edges that get added to each net while eliminating e. Then, we replace e with the new path in the first q nets ensuring minimization of the overall constraints violation.

The results obtained by our basic heuristic and the above variants are approximately the same. Therefore, in the next section, without the loss of generality, we present the results for the second variant.

IV. EXPERIMENTAL RESULTS

We perform experiments on a machine with single Intel(R) Xeon(R) CPU E5-1620 v3 CPU running at 3.50 GHz and 64 GB of RAM. We use the Ubuntu 20.04.1 LTS operating system with kernel version 5.13.0–40. Our code is written in C++11 and compiled using GCC version 9.4.0 with O3 optimization flag. The resulting compiled code is run using a different number of threads.

The architecture parameters used for our experiments are given in Table 2, which are most commonly used [28]–[30]. In Table 2, the values of N and K specify that the CLBs in the architecture contains ten fracturable logic elements (FLEs)
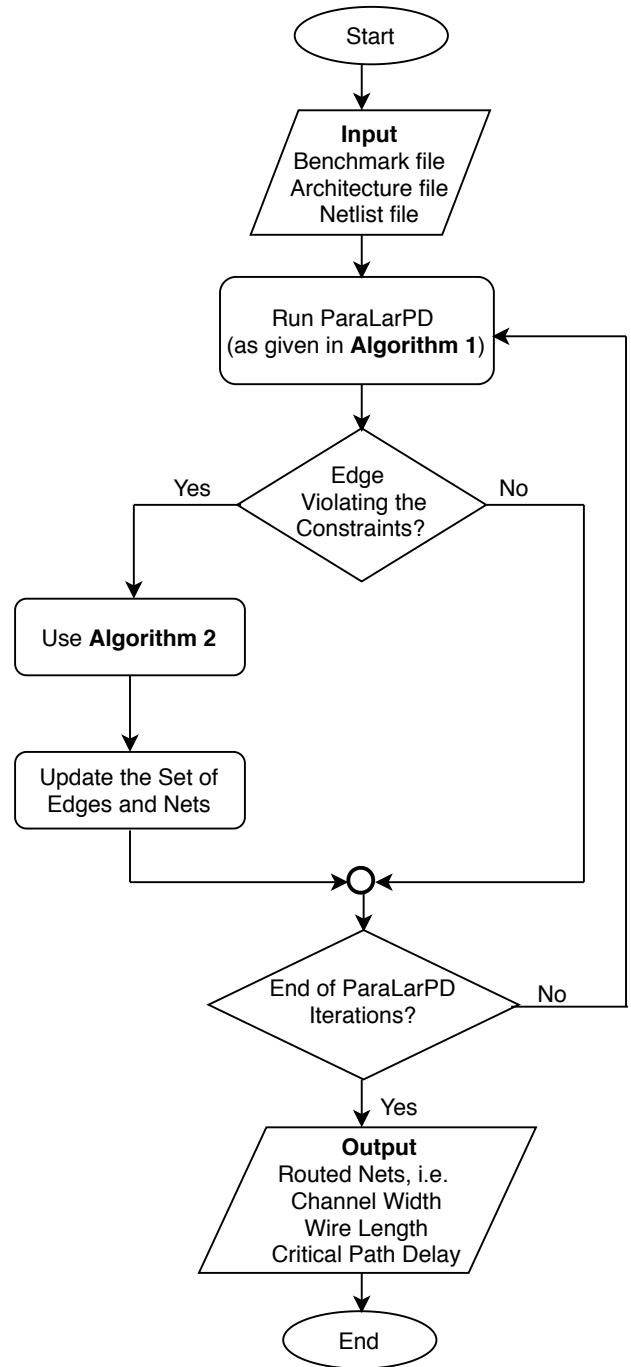
FIGURE 2: Data flow diagram of our ParaLarH.

TABLE 2: FPGA design architecture parameters used in our experiments.

N	K	F_{cin}	F_{cout}	F_s	Length
10	6	0.15	0.10	3	4

and each FLE has six inputs, respectively. The values of F_{cin} and F_{cout} specify that every input and output pin is driven by 15% and 10%, of the tracks in a channel, respectively. In FPGA terminology, the value of F_s specifies the number of

wire segments that can be connected to each wire segment where horizontal and vertical channels intersect. This value can only be a multiple of 3. Here, we perform experiments with $F_s = 3$. The value of length specifies the number of logic blocks spanned by each segment. We took this as 4, although our proposed method can be used for architectures with varying lengths, e.g., length = 1 or a mix of length = 1 and length = 4.

For parallelization, we use Intel threading building blocks (TBB) libraries. Also, in our proposed model, routing of individual nets is independent and we update the cost of utilizing the edges at the end of each routing iteration. Thus, there is no race condition leading to no randomness. Hence, our executions are deterministic. To have a less biased timing data, we perform 100 independent runs of each of the algorithms and report aggregate results.

There is no general rule of choosing the initial value of the channel width for experimental purposes. However, a value of 20% to 40% more than the minimum channel width obtained from VPR is commonly used. For our experiments, all algorithms are initialized with initial channel width (W) as $1.2W_{min}$, where W_{min} is the minimum channel width obtained from VPR. We also do experiments with initial W as $1.4W_{min}$, which does not change the results. We use an upper limit of 50 for the number of iterations for all the methods. The best results out of all these iterations are reported.

We perform four sets of experiments:
- First, on the MCNC benchmarks when VPR 7.0 is used to pack and place.
- Second, again on the MCNC benchmarks when VPR 8.0 is used.
- Third, on the VTR benchmarks when VPR 7.0 is used.
- Fourth and finally, again on the VTR benchmarks when VPR 8.0 is used.

A. EXPERIMENTS ON MCNC BENCHMARKS
The experiments on MCNC benchmark circuits using VPR 7.0 and VPR 8.0 are given in the two subsections below.

1) Using VPR 7.0
We compare ParaLarH with two earlier algorithms of the same family (ParaLarPD [6] and ParaLaR [4]), two other standard algorithms (RVPack [30] and GGAPack2 [30]), and VPR 7.0. Initially circuits are packed and placed using VPR 7.0, and then routing is performed by the respective method.

The algorithms are compared wing the metrics of absolute constraints violation, minimum channel width, the total wire length, average critical path delay, and speed-ups.

In Table 3, we compare the constraints violation of ParaLarH, ParaLarPD and ParaLaR. Note that there is no violation in RVPack, GGAPack2, and VPR 7.0. As evident from this table, ParaLarH has the least constraints violation, and hence, performs the best.

In Table 4, we compare the minimum channel width of all the six algorithms stated earlier (ParaLarH, ParaLarPD,

ParaLaR, RVPack, GGAPack2, and VPR 7.0). As evident from the table, ParaLarH performs the best with 32.68% improvement over VPR 7.0.

In Table 5, we compare the total wire length, again of all the six algorithms. As evident from the table, ParaLarH performs a close third best here with ParaLarPD performing best. All three: ParaLarH, ParaLarPD, and ParaLaR achieve approximately 46% improvement over VPR 7.0.

Finally, in Table 6, we compare the average critical path delay, again of all the six algorithms. As evident from the table, here as well ParaLarH performs the best with 10.01% improvement over VPR 7.0.

TABLE 3: Comparison of the constraints violation between our proposed ParaLarH, ParaLarPD [6] and ParaLaR [4] when experimenting on MCNC benchmarks using VPR 7.0.

Benchmarks Circuits	ParaLarH	ParaLarPD	ParaLaR
Alu4	4.67	5.54	15.27
Apex2	5.12	10.08	28.06
Apex4	1.87	5.58	13.48
Bigkey	5.18	9.04	13.27
Clma	5.32	16.44	31.00
Des	5.40	11.17	20.10
Diffeq	2.67	7.52	19.23
Dsip	4.86	5.63	12.33
Elliptic	5.20	12.42	36.00
Ex5p	4.78	8.83	17.00
Ex1010	4.23	8.31	20.00
Frisc	6.45	13.71	51.38
Misex3	4.56	7.27	13.67
Pdc	4.34	8.67	26.00
S298	4.87	9.00	16.29
S38417	5.34	10.48	32.00
Seq	4.68	8.85	19.00
Spla	4.92	9.41	24.33
Tseng	4.87	9.65	11.67
Average	**4.70**	**9.35**	**22.11**

Next, we calculate the relative speed-ups by the formula given below [6].

$$\text{Speedup} = \frac{\text{Execution time with 1 thread}}{\text{Execution time with n threads}}.$$

The speed-ups obtained in executing the benchmarks via ParaLarH in a parallel setting are given as a bar graph in Figure 3. In this figure, on the x-axis we have the benchmark circuits arranged in the increasing order of their execution time when running them with one thread. This time is directly proportional to the benchmark size. On the y-axis, we have the speed-ups in execution of these benchmark circuits when using 2 threads, 4 threads, and 8 threads in ParaLarH. From this bar graph, we observe that on an average, 2, 4, and 8 threads give speed-ups of 1.70, 2.28, and 2.89, respectively.

Since in ParaLarPD paper [6], the experiments were performed on a machine having different operating system, GCC version, and kernel, hence, for a fair comparison, we also perform ParaLarPD's and ParaLaR's experiment on the same machine and report the speed-ups obtained by a bar graph in Figure 4 and Figure 5, respectively. From this bar graph, we observe that on an average, 2, 4, and 8 threads give speed-ups of 1.84, 2.60, and 3.27, respectively for ParaLarPD and

TABLE 4: Comparison of the minimum channel width between our proposed ParaLarH, ParaLarPD [6], ParaLaR [4], RVPack [30], GGAPack2 [30], and VPR 7.0 [3] when experimenting on MCNC benchmarks using VPR 7.0.

Benchmarks Circuits	ParaLarH	ParaLarPD	ParaLaR	RVPack†	GGAPack2†	VPR
Alu4	34.67	35.54	45.27	49	52	48
Apex2	45.12	50.08	68.06	50	59	64
Apex4	41.87	45.58	53.48	59	60	62
Bigkey	15.18	19.04	23.27	34	38	50
Clma	70.32	81.44	96.00	78	100	94
Des	25.40	31.17	40.10	34	43	40
Diffeq	32.67	37.52	49.23	31	41	54
Dsip	24.86	25.63	32.33	30	38	38
Elliptic	50.20	57.42	81.00	53	60	74
Ex5p	44.78	48.83	57 .00	61	60	70
Ex1010	59.23	63.31	75.00	61	83	82
Frisc	61.45	68.71	106.38	61	78	86
Misex3	39.56	42.27	48.67	49	50	58
Pdc	69.34	73.67	91.00	85	90	92
S298	34.87	39	46.29	62	58	48
S38417	45.34	50.48	72.00	49	78	64
Seq	44.68	48.85	59.00	50	59	70
Spla	54.92	59.41	74.33	70	79	80
Tseng	34.87	39.65	41.67	29	29	58
Average	**43.65**	**48.29**	**61.06**	**52.37**	**60.79**	**64.84**
% improvement over VPR 7.0	**32.68**	**25.52**	**5.84**	**19.24**	**6.25**	**–**

† Note: Original thesis has provided data for these algorithms only in graphical format. Also, the code and pseudo-code for these algorithms is unavailable and so we are unable to generate results for individual circuits. Hence, we report the approximate values as read from the bar graphs.

TABLE 5: Comparison of the total wire length (in nanometers) between our proposed ParaLarH, ParaLarPD [6], ParaLaR [4], RVPack [30], GGAPack2 [30], and VPR 7.0 [3] when experimenting on MCNC benchmarks using VPR 7.0.

Benchmarks Circuits	ParaLarH	ParaLarPD	ParaLaR	RVPack†	GGAPack2†	VPR
Alu4	5030	5030	5029	16500	16700	10480
Apex2	7978	7935	7934	18700	20100	15881
Apex4	5807	5630	5632	15600	16200	10746
Bigkey	3927	3896	3896	19400	20100	7052
Clma	49474	49278	49284	100000	130200	87398
Des	7043	6952	6952	21000	29000	14739
Diffeq	4693	4349	4350	9400	10100	9140
Dsip	4771	4778	4778	18300	19900	9742
Elliptic	15253	15125	15124	29900	39000	28271
Ex5p	4916	4889	4881	10200	10600	10169
Ex1010	23603	23596	23950	49900	70100	43919
Frisc	19713	19484	19484	38200	49800	35664
Misex3	5195	5194	5192	11400	12300	10061
Pdc	30435	30423	30425	61300	78500	53661
S298	5256	5250	5250	17800	19100	10291
S38417	21962	21907	21906	42900	78100	42597
Seq	7685	7654	7653	18900	19900	14203
Spla	20139	20117	20117	42100	53000	37384
Tseng	2491	2484	2484	80200	90100	6148
Average	**12914.26**	**12840.58**	**12859.00**	**32721.05**	**41200.00**	**24081.37**
% Improvements over VPR 7.0	**46.37**	**46.67**	**46.60**	**-35.87**	**-71.08**	**–**

† Note: Original thesis has provided data for these algorithms only in graphical format. Also, the code and pseudo-code for these algorithms is unavailable and so we are unable to generate results for individual circuits. Hence, we report the approximate values as read from the bar graphs.

speed-ups of 1.86, 2.62, and 3.30, respectively for ParaLaR. If we compare the speed-ups obtained from ParaLarH with ParaLarPD and ParaLaR, we observe that there is a slight deterioration in the case of ParaLarH.

ParaLarH's drop in speedups can be fixed by using more number of threads. More threads would improve the speedups of other algorithms as well but these algorithms

would not be able to improve the minimum channel width and critical path delay, which ParaLarH does.

2) Using VPR 8.0

Here, we compare ParaLarH, ParaLarPD and VPR 8.0. Since ParaLaR, RVPack and GGAPack2 have been designed to work only with VPR 7.0, here we do not compare them.

TABLE 6: Comparison of the critical path delay (in nanoseconds) between our proposed ParaLarH, ParaLarPD [6], ParaLaR [4], RVPack [30], GGAPack2 [30], and VPR 7.0 [3] when experimenting on MCNC benchmarks using VPR 7.0.

Benchmarks Circuits	ParaLarH	ParaLarPD	ParaLaR	RVPack	GGAPack2	VPR
Alu4	6.69	7.30	7.01	17.52	13.20	7.50
Apex2	7.30	7.41	7.16	17.81	18.20	7.26
Apex4	6.51	7.08	6.73	13.90	15.05	6.92
Bigkey	3.32	4.01	4.44	10.05	10.10	3.53
Clma	15.42	15.46	16.31	36.80	41.05	15.08
Des	5.47	5.55	5.54	14.10	17.40	5.83
Diffeq	5.84	5.65	5.72	11.50	13.45	7.09
Dsip	3.19	3.62	3.45	8.50	10.02	4.20
Elliptic	7.53	10.83	10.91	27.40	23.10	13.98
Ex5p	6.32	6.94	6.28	14.20	14.50	7.69
Ex1010	12.00	14.57	12.71	23.50	32.40	10.05
Frisc	12.68	13.13	12.84	21.50	25.01	15.38
Misex3	6.19	6.49	6.68	15.05	14.50	6.08
Pdc	12.39	12.63	12.49	25.02	26.02	11.75
S298	11.67	12.71	12.08	20.03	22.50	16.62
S38417	9.11	10.44	10.03	22.04	24.95	8.82
Seq	6.00	6.14	6.18	17.60	17.90	6.09
Spla	10.23	10.43	10.69	24.05	26.50	10.11
Tseng	5.78	5.78	5.78	12.10	12.50	6.75
Average	**8.08**	**8.74**	**8.58**	**18.56**	**19.91**	**8.98**
% Improvements over VPR 7.0	**10.01**	**2.67**	**4.51**	**-106.56**	**-121.61**	**–**

† Note: Original thesis has provided data for these algorithms only in graphical format. Also, the code and pseudo-code for these algorithms is unavailable and so we are unable to generate results for individual circuits. Hence, we report the approximate values as read from the bar graphs.
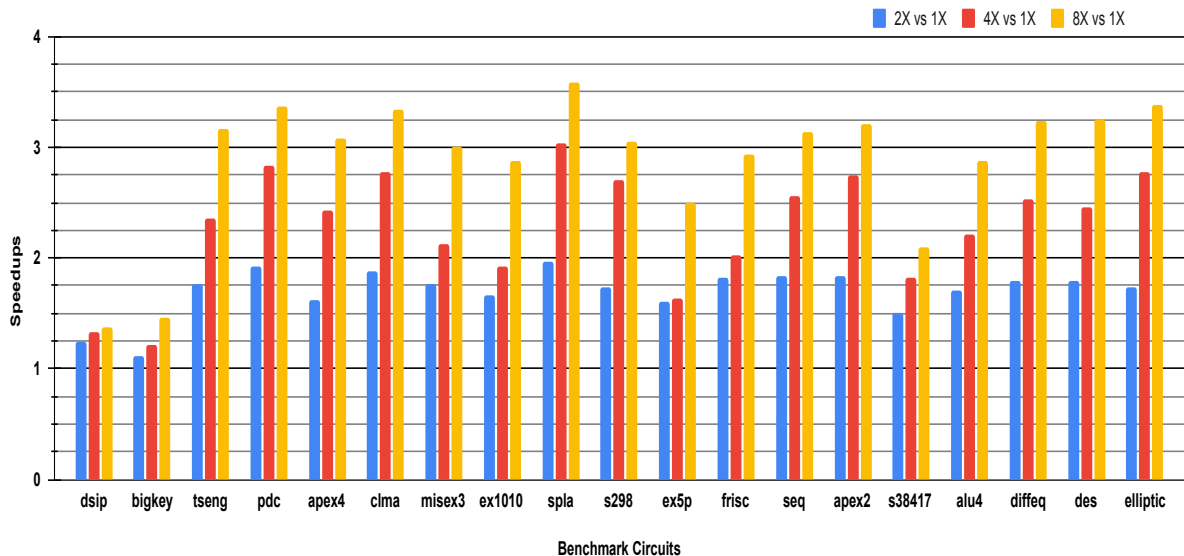

FIGURE 3: Speedups of each benchmark using ParaLarH when running it with 2, 4 and 8 threads.

Initially circuits are packed and placed using VPR 8.0 and then routing is performed with the respective method.

We compare these algorithms using the metrics of constraints violation, the minimum channel width, the total wire length, and the critical path delay. The speed-ups obtained here are nearly the same as those using VPR 7.0. Hence, to avoid repetition, we not discuss them here.

In Table 7, we compare the constraints violation in ParaLarH and ParaLarPD. As earlier, in VPR 8.0, there is no concept of constraints violation because it does not formulate the routing problem as an optimization problem. As evident from this table, ParaLarH has the least constraints violation, and perform the best.

The minimum channel width comparisons are done in Table 8. On an average, ParaLarH and ParaLarPD give 31.98% and 22.62% improvement over VPR 8.0, respectively.

The total wire length is compared in Table 9. On an average, ParaLarH and ParaLarPD give 53.36% and 53.43%
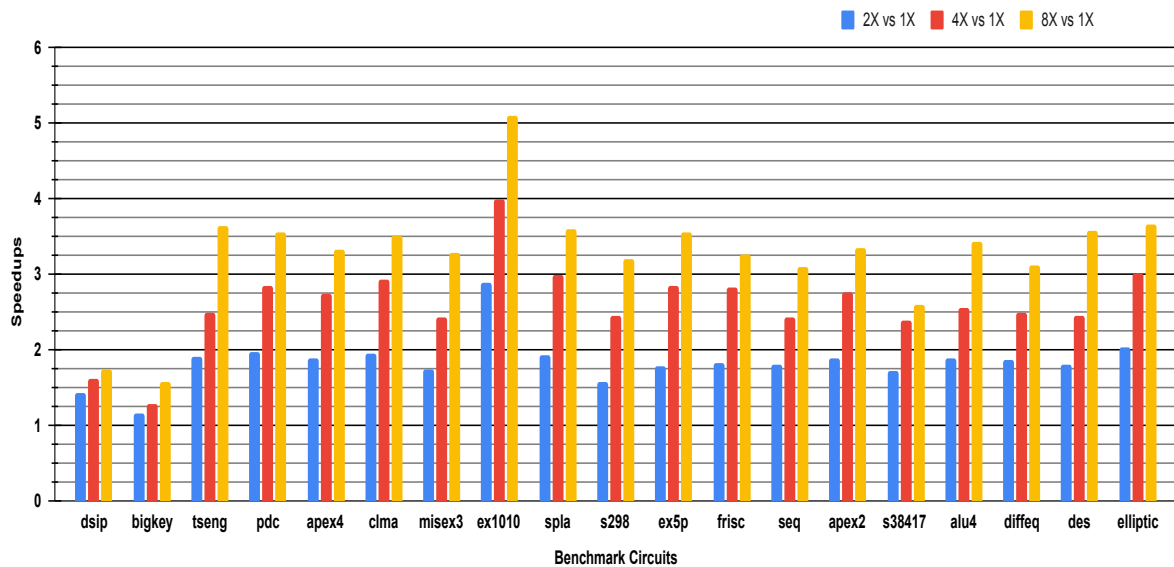
FIGURE 4: Speedups of each benchmark using ParaLarPD when running it with 2, 4 and 8 threads.
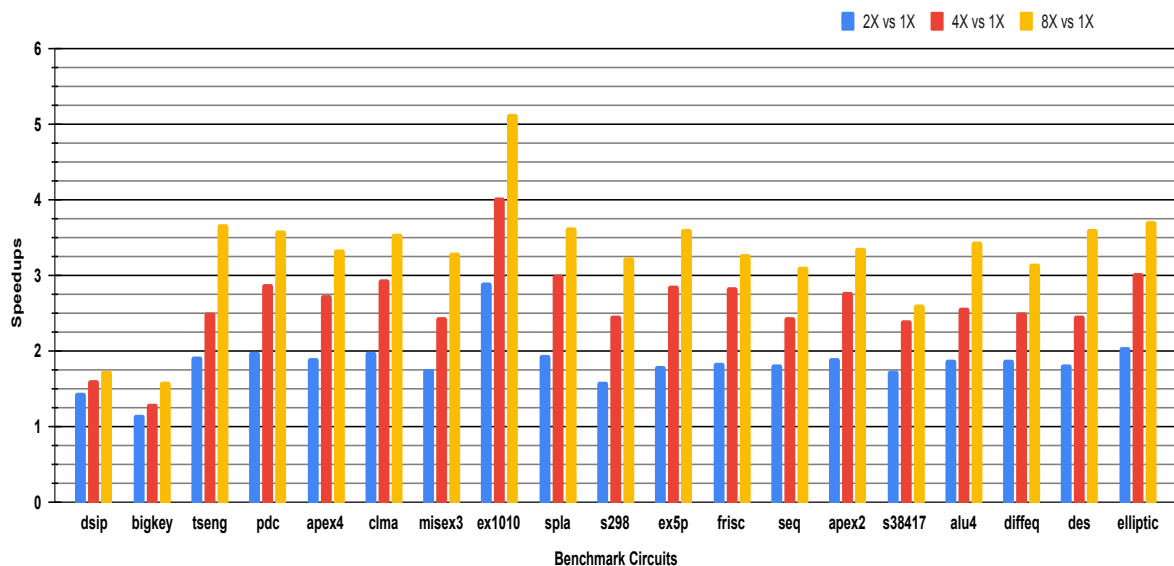

FIGURE 5: Speedups of each benchmark using ParaLaR when running it with 2, 4 and 8 threads.

improvement over VPR 8.0, respectively.

The average critical path delay comparison are done in Table 10. On an average, ParaLarH and ParaLarPD give 13.89% and 9.61% improvement over VPR 8.0, respectively.

Overall, ParaLarH and ParaLarPD both perform substantially better that VPR 8.0, with ParaLarH being the best.

B. EXPERIMENTS ON VTR BENCHMARKS

While experimenting on the VTR benchmarks, here, we perform comparisons between ParaLarH, ParaLarPD, and VPR (VPR 7.0 & VPR 8.0 both) in two respective subsections below. Since the experimental data of ParaLaR, RVPack,

and GGAPack2 for VTR benchmarks is not available, we do not compare against them. Initially, circuits are packed and placed with VPR (VPR 7.0 and VPR 8.0 as the context may be) and then routing is performed with the respective method.

We compare these algorithms using the metrics of constraints violation, the minimum channel width, the total wire length, and the critical path delay. The speed-ups obtained here are nearly the same as those on the MCNC benchmarks. Hence, to avoid repetition, we not discuss them here. The rest of this section has two parts; first, where we use VPR 7.0 and second, where we use VPR 8.0.

TABLE 7: Comparison of the constraints violation between ParaLarH and ParaLarPD when experimenting on MCNC benchmarks using VPR 8.0.

Benchmark Circuits	Absolute Constraints Violation	
	ParaLarH	ParaLarPD
Alu4	4.45	8.43
Apex2	5.23	12.32
Apex4	6.12	13.32
Bigkey	4.45	7.76
Clma	9.34	17.54
Des	5.23	10.43
Diffeq	5.34	11.65
Dsip	4.12	8.45
Elliptic	6.45	11.34
Ex5p	4.44	9.48
Ex1010	4.54	10.67
Frisc	7.53	14.45
Misex	4.43	10.21
Pdc	5.12	13.32
S298	4.45	8.32
S38417	5.12	10.23
Seq	4.23	8.76
Spla	6.23	13.65
Tseng	4.32	10.56
Average	**5.32**	**11.10**

TABLE 8: Comparison of the minimum channel width between ParaLarH, ParaLarPD, and VPR 8.0 when experimenting on MCNC benchmarks using VPR 8.0.

Benchmark Circuits	Minimum Channel Width		
	ParaLarH	ParaLarPD	VPR 8.0
Alu4	32.45	36.43	50
Apex2	45.23	52.32	70
Apex4	40.12	47.32	64
Bigkey	16.45	19.76	36
Clma	71.34	79.54	88
Des	27.23	32.43	40
Diffeq	33.34	39.65	56
Dsip	22.12	26.45	38
Elliptic	52.45	57.34	72
Ex5p	44.44	49.48	72
Ex1010	48.54	54.67	70
Frisc	61.53	68.45	80
Misex	36.43	42.21	58
Pdc	65.12	73.32	88
S298	28.45	32.32	36
S38417	43.12	48.23	56
Seq	44.23	48.76	64
Spla	54.23	61.65	78
Tseng	30.32	36.56	56
Average	**41.95**	**47.73**	**61.68**
% Improvements over VPR 8.0	**31.98**	**22.62**	**–**

1) Using VPR 7.0

In Table 11, we compare the constraints violation in ParaLarH and ParaLarPD. As earlier, in VPR 7.0, there is no concept of constraints violation, and hence, we cannot compare with it. As evident from this table, ParaLarH has the least constraints violation, and hence, performs the best.

The rest of the comparisons are as follows:

- The minimum channel width: From Table 12, we can see that ParaLarH and ParaLarPD give 19.08% and 4.15% improvement over VPR 7.0, respectively.

TABLE 9: Comparison of the total wire length (in nanometers) between ParaLarH, ParaLarPD, and VPR 8.0 when experimenting on MCNC benchmarks using VPR 8.0.

Benchmark Circuits	Total Wire Length		
	ParaLarH	ParaLarPD	VPR 8.0
Alu4	4941	4929	10548
Apex2	7840	7824	17426
Apex4	5502	5469	11307
Bigkey	3529	3529	7759
Clma	42650	42654	85600
Des	6725	6717	16602
Diffeq	4363	4254	9958
Dsip	4064	4060	9866
Elliptic	14459	14453	30358
Ex5p	4854	4807	10667
Ex1010	19114	19100	43938
Frisc	17672	17670	35828
Misex	5228	5218	11987
Pdc	30144	30140	61179
S298	3909	3809	8389
S38417	15302	15303	37201
Seq	7502	7500	16309
Spla	19579	19553	39694
Tseng	2139	2139	6000
Average	**11553**	**11533**	**24769**
% Improvements over VPR 8.0	**53.36**	**53.43**	**–**

TABLE 10: Comparison of the critical path delay (in nanoseconds) between ParaLarH, ParaLarPD, and VPR 8.0 when experimenting on MCNC benchmarks using VPR 8.0.

Benchmark Circuits	Critical Path Delay		
	ParaLarH	ParaLarPD	VPR 8.0
Alu4	6.68	7.68	9.82
Apex2	7.79	7.84	9.70
Apex4	7.30	7.60	7.67
Bigkey	4.37	5.35	4.29
Clma	14.72	15.43	14.63
Des	5.87	6.47	6.86
Diffeq	6.08	6.25	7.74
Dsip	4.09	4.31	3.97
Elliptic	9.79	10.73	12.30
Ex5p	6.23	6.43	7.07
Ex1010	12.36	12.66	12.24
Frisc	11.12	11.49	13.86
Misex	6.00	6.15	8.11
Pdc	13.48	14.17	14.28
S298	11.62	12.30	14.65
S38417	8.32	8.32	11.22
Seq	6.68	7.19	7.88
Spla	10.59	10.59	11.26
Tseng	5.51	5.51	6.63
Average	**8.35**	**8.76**	**9.69**
% Improvements over VPR 8.0	**13.89**	**9.61**	**–**

- The total wire length: From Table 13, we can see that ParaLarH and ParaLarPD give 38.09% and 38.24% improvement over VPR 7.0, respectively.
- The average critical path delay: From Table 14, we can see that ParaLarH and ParaLarPD give 64.58% and 67.74% improvement over VPR 7.0, respectively.

Overall ParaLarH and ParaLarPD both perform substantially better than VPR 7.0, with ParaLarH being better for the

TABLE 11: Comparison of the constraints violation between ParaLarH and ParaLarPD when experimenting on VTR benchmarks using VPR 7.0.

Benchmark Circuits	Absolute Constraints Violation	
	ParaLarH	ParaLarPD
bgm	10.46	21.34
Blob_merge	6.56	14.23
bound_top	4.54	10.32
ch_intrinsic	4.23	8.56
diffeq1	4.12	19.66
diffeq2	4.32	11.21
LU8PEEng	9.68	19.87
LU32PEEng	13.68	19.42
mcml	11.32	36.31
mkDelay	5.21	11.42
mkPktMerge	4.56	12.67
mkSM	4.45	9.32
or1200	4.22	11.12
raygentop	5.89	11.67
sha	4.13	7.56
stereovision0	5.92	10.34
stereovision1	7.65	19.74
stereovision2	13.32	37.57
stereovision3	2.65	9.43
Average	**6.64**	**15.80**

TABLE 12: Comparison of the minimum channel width between ParaLarH, ParaLarPD, and VPR 7.0 when experimenting on VTR benchmarks using VPR 7.0.

Benchmark Circuits	Minimum Channel Width		
	ParaLarH	ParaLarPD	VPR 7.0
bgm	90.46	101.34	116
Blob_merge	52.56	60.23	74
bound_top	40.54	46.32	60
ch_intrinsic	22.23	26.56	50
diffeq1	42.12	57.66	52
diffeq2	36.32	43.21	50
LU8PEEng	91.68	101.87	114
LU32PEEng	165.80	214.00	174
mcml	114.00	149.00	104
mkDelay	55.21	61.42	76
mkPktMerge	26.56	34.67	46
mkSM	36.45	41.32	56
or1200	56.22	63.12	74
raygentop	53.89	59.67	68
sha	42.13	45.56	50
stereovision0	43.92	48.34	62
stereovision1	97.65	109.74	102
stereovision2	149.32	173.57	154
stereovision3	14.65	21.43	34
Average	**64.27**	**76.13**	**79.43**
% Improvements over VPR 7.0	**19.08**	**4.15**	**–**

minimum channel width and ParaLarPD being better for the critical path delay.

2) Using VPR 8.0

In Table 15, we compare the constraints violation in ParaLarH and ParaLarPD. As earlier, in VPR 8.0, there is no concept of constraints violation, and hence, we cannot compared with it. As evident from this table, ParaLarH has the least constraints violation, and hence, performs the best.

The rest of the comparisons are as follows:

TABLE 13: Comparison of the total wire length (in nanometers) between ParaLarH, ParaLarPD, and VPR 7.0 when experimenting on VTR benchmarks using VPR 7.0.

Benchmark Circuits	Total Wire Length		
	ParaLarH	ParaLarPD	VPR 7.0
bgm	354258	354251	624608
Blob_merge	43489	43483	82613
bound_top	15174	15129	30999
ch_intrinsic	1758	1751	4237
diffeq1	4563	4511	12009
diffeq2	3172	3125	8446
LU8PEEng	253039	253001	448756
LU32PEEng	1273984	1272597	2003675
mcml	795213	788607	1331742
mkDelay	72545	72425	121393
mkPktMerge	7639	7624	16019
mkSM	11529	11500	26310
or1200	27300	27291	54911
raygentop	13671	13670	29943
sha	10498	10497	21198
stereovision0	54899	54857	95071
stereovision1	111294	111282	184401
stereovision2	640826	640539	859954
stereovision3	268	229	797
Average	**186356**	**185901**	**301026**
% Improvements over VPR 7.0	**38.09**	**38.24**	**–**

TABLE 14: Comparison of the critical path delay (in nanoseconds) between ParaLarH, ParaLarPD, and VPR 7.0 when experimenting on VTR benchmarks using VPR 7.0.

Benchmark Circuits	Critical Path Delay		
	ParaLarH	ParaLarPD	VPR 7.0
bgm	8.05	8.05	26.52
Blob_merge	6.47	5.12	10.77
bound_top	7.53	4.52	8.00
ch_intrinsic	1.20	1.05	4.53
diffeq1	15.81	15.81	26.08
diffeq2	1.50	1.50	17.91
LU8PEEng	11.74	10.24	115.43
LU32PEng	27.52	27.65	115.14
mcml	30.8	23.43	79.65
mkDelay	8.65	8.96	7.29
mkPktMerge	3.91	3.91	4.57
mkSM	5.87	3.91	8.22
or1200	5.12	5.04	14.45
raygentop	4.44	3.54	6.66
sha	3.46	3.99	16.46
stereovision0	7.07	7.23	4.58
stereovision1	7.60	7.30	6.86
stereovision2	13.17	13.17	19.46
stereovision3	3.76	3.76	2.88
Average	**9.01**	**8.21**	**25.45**
% Improvements over VPR 7.0	**64.58**	**67.74**	**–**

- The minimum channel width: From Table 16, we can see that ParaLarH and ParaLarPD give 3.12% improvement and 3.34% deterioration over VPR 8.0, respectively. Here, negative sign indicate a deterioration.
- The total wire length: From Table 17, we can see that ParaLarH and ParaLarPD give 41.74% and 40.36% improvement over VPR 8.0, respectively.
- The average critical path delay: From Table 18, we

TABLE 15: Comparison of the constraints violation between ParaLarH and ParaLarPD when experimenting on VTR benchmarks using VPR 8.0.

Benchmark Circuits	Absolute Constraints Violation	
	ParaLarH	**ParaLarPD**
bgm	2.01	10.12
blob_merge	1.54	8.32
boundtop	8.1	9.98
ch_intrinsics	2.12	5.02
diffeq1	4.21	6.32
diffeq2	1.65	13.98
LU32PEEng	36.87	40.86
LU8PEEng	0.24	8.14
mcml	2.12	3.21
mkDelayWorker32B	0.96	9.34
mkPktMerge	3.12	11.12
mkSMAdapter4B	1.96	6.1
or1200	6.92	6.78
raygentop	3.86	4.32
sha	8.12	4.12
stereovision0	2.04	2.32
stereovision1	2.98	5.87
stereovision2	0.92	11.79
stereovision3	1.92	3.32
Average	**4.82**	**9.00**

TABLE 16: Comparison of the minimum channel width between ParaLarH, ParaLarPD, and VPR 8.0 when experimenting on VTR benchmarks using VPR 8.0.

Benchmark Circuits	Minimum Channel Width		
	ParaLarH	**ParaLarPD**	**VPR 8.0**
bgm	66.01	74.12	74
blob_merge	55.54	62.32	62
boundtop	44.1	45.98	50
ch_intrinsics	26.12	29.02	40
diffeq1	58.21	60.32	46
diffeq2	57.65	69.98	40
LU32PEEng	178.87	182.86	128
LU8PEEng	74.24	82.14	88
mcml	80.12	81.21	86
mkDelayWorker32B	64.96	73.34	64
mkPktMerge	39.12	47.12	36
mkSMAdapter4B	41.96	46.1	58
or1200	62.92	62.78	74
raygentop	49.86	50.32	62
sha	44.12	40.12	50
stereovision0	46.04	46.32	58
stereovision1	62.98	65.87	80
stereovision2	114.92	125.79	102
stereovision3	21.92	23.32	30
Average	**62.61**	**66.79**	**64.63**
% Improvements over VPR 8.0	**3.12**	**-3.34**	**–**

TABLE 17: Comparison of the total wire length (in nanometers) between ParaLarH, ParaLarPD, and VPR 8.0 when experimenting on VTR benchmarks using VPR 8.0.

Benchmark Circuits	Total Wire Length		
	ParaLarH	**ParaLarPD**	**VPR 8.0**
bgm	227698	231627	436804
blob_merge	37423	38014	73300
boundtop	12344	12498	26980
ch_intrinsics	1859	1921	4693
diffeq1	4074	4289	12840
diffeq2	2796	2943	10166
LU32PEEng	926132	945721	1484514
LU8PEEng	198681	206596	363246
mcml	494368	494368	875701
mkDelayWorker32B	140978	140552	195579
mkPktMerge	7617	7570	16760
mkSMAdapter4B	11666	11810	25590
or1200	27790	27768	57669
raygentop	13383	13531	32114
sha	9147	9627	21268
stereovision0	37172	38956	75570
stereovision1	71348	73945	148262
stereovision2	410669	435651	661327
stereovision3	236	243	942
Average	**138704**	**141980**	**238069**
% Improvements over VPR 8.0	**41.74**	**40.36**	**–**

TABLE 18: Comparison of the critical path delay (in nanoseconds) between ParaLarH, ParaLarPD, and VPR 8.0 when experimenting on VTR benchmarks using VPR 8.0.

Benchmark Circuits	Critical Path Delay		
	ParaLarH	**ParaLarPD**	**VPR 8.0**
bgm	8.58	6.93	26.69
blob_merge	5.42	5.57	10.78
boundtop	3.91	3.24	7.08
ch_intrinsics	1.80	1.43	4.08
diffeq1	2.11	1.50	22.66
diffeq2	1.35	1.73	19.22
LU32PEEng	27.71	33.73	110.81
LU8PEEng	11.89	11.52	108.47
mcml	31.39	31.39	83.87
mkDelayWorker32B	22.28	29.06	9.60
mkPktMerge	4.14	4.512	3.98
mkSMAdapter4B	3.01	3.08	6.67
or1200	4.29	4.97	15.03
raygentop	4.37	3.16	5.74
sha	3.91	4.81	14.14
stereovision0	5.65	5.65	4.41
stereovision1	9.26	8.05	5.84
stereovision2	35.68	27.86	17.98
stereovision3	0.37	0.37	2.96
Average	**9.85**	**9.92**	**25.26**
% Improvements over VPR 8.0	**61.02**	**60.71**	**–**

can see that ParaLarH and ParaLarPD give 61.02% and 60.71% improvement over VPR 8.0, respectively.

Overall ParaLarH and ParaLarPD both perform substantially better than VPR 8.0, with ParaLarH being the best.

V. CONCLUSIONS AND FUTURE WORK

In this work, we have three main contributions. First, we make our earlier proposed ParaLarPD compatible for testing on MCNC benchmark circuits using recently proposed VPR 8.0 (earlier we had experimented with VPR 7.0 only). Sec-

ond, we adapt ParaLarPD for larger benchmark circuits of VTR (earlier we had experimented with MCNC benchmark circuits) using both VPR 7.0 and VPR 8.0, and perform thorough evaluation.

Third, we improve the Lagrange relaxations process of ParaLarPD via new Lagrange heuristics. We term new algorithm ParaLarH. We test ParaLarH on both the benchmark circuits (MCNC and VTR) and using both the VPRs (VPR

7.0 and VPR 8.0).

With experiments on both the benchmarks circuits (MCNC and VTR), we observe that ParaLarH and ParaLarPD both outperform VPR (VPR 7.0 and VPR 8.0) with gains as below.

- The minimum channel width improvements in ParaLarH and ParaLarPD are 22% and 12%, respectively.
- The total wire length improvements in both ParaLarH and ParaLarPD are 45%.
- The average critical path delay improvements in ParaLarH and ParaLarPD are also almost same (37% and 35%, respectively).

We also observe that ParaLarH performs the best.

Next, we discuss the future work. First, in both the routing algorithms (the above discussed ParaLarH and our earlier published ParaLarPD), we use sub-gradient based methods for the solution of BILP. Here, future direction involves finding other more efficient algorithms for solving this BILP. Second, we plan to work towards designing algorithms that would completely remove the constraints violation. Third, we plan to experiment on TITAN benchmarks as well [31]. TITAN benchmarks are larger than MCNC and VTR benchmarks and are also used to evaluate FPGA architectures. These benchmarks make use of heterogeneous resources (RAM blocks and DSP), which is less common in other benchmarks.

ACKNOWLEDGMENT

REFERENCES

[1] Jiang Yue, Chen Hongyi, Yang Xiangrui, Sun Zhigang, and Quan Wei. Design and implementation of CPU & FPGA co-design tester for SDN switches. Electronics, 8(9):950, 2019.

[2] Yu Hoyoung, Lee Hansol, Lee Sangil, Kim Youngmin, and Lee Hyung-Min. Recent advances in FPGA reverse engineering. Electronics, 7(10):246, 2018.

[3] Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, Kenneth B. Kent, Jason Anderson, Rose Jonathan, and Vaughn Betz. VTR 7.0: Next generation architecture and CAD system for FPGAs. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 7(2):1–30, 2014.

[4] Chin H. Hoo, Akash Kumar, and Yajun Ha. ParaLaR: A parallel FPGA router based on Lagrangian relaxation. In Proceeding of the 25th International Workshop on Field-Programmable Logic and Applications (FPL), pages 1–6, London, UK, 2015.

[5] Saeyang Yang. Logic synthesis and optimization benchmarks user guide: version 3.0. In Microelectronics Center of North Carolina (MCNC), Research Triangle Park, NC, 1991.

[6] Rohit Agrawal, Kapil Ahuja, Chin H. Hoo, Taun D. A. Nguyen, and Akash Kumar. ParaLarPD: Parallel FPGA router using primal-dual sub-gradient method. Electronics, 8(12):1439, 2019.

[7] Marshall L. Fisher. The Lagrangian relaxation method for solving integer programming problems. Management Science, 27(1):1–8, 1981.

[8] Yuan Cao, Zixuan Zhang, Fanglin Cheng, and Shuai Su. Trajectory optimization for high-speed trains via a mixed integer linear programming approach. IEEE Transactions on Intelligent Transportation Systems, 23(10):17666–17676, 2022.

[9] David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. Mathematical Programming, pages 1–52, 2022.

[10] Hafiz A. U. Muqeet and Aftab Ahmad. Optimal scheduling for campus prosumer microgrid considering price based demand response. IEEE Access, 8:71378–71394, 2020.

[11] Abdulqader O. Hamadameen and Nasruddin Hassan. A compromise solution for the fully fuzzy multiobjective linear programming problems. IEEE Access, 6:43696–43711, 2018.

[12] Lucas Polo-López, Juan Córcoles, and Jorge A. Ruiz-Cruz. Antenna design by means of the fruit fly optimization algorithm. Electronics, 7(1):3, 2018.

[13] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. Interior point methods for linear programming: Computational state of the art. ORSA Journal on Computing, 6(1):1–14, 1994.

[14] Pham D. Tao and El B. Souad. Duality in D.C. (difference of convex functions) optimization. Subgradient methods. In Trends in Mathematical Optimization: 4th French-German Conference on Optimization, pages 277–293. Springer, 1988.

[15] Qing Yang and Gang Chen. Primal-dual subgradient algorithm for distributed constraint optimization over unbalanced digraphs. IEEE Access, 7:85190–85202, 2019.

[16] Rohit Agrawal and Kapil Ahuja. CSIS: Compressed sensing-based enhanced-embedding capacity image steganography scheme. IET Image Processing, 15(9):1909–1925, 2021.

[17] Rohit Agrawal, Aditya A. Shastri, Kapil Ahuja, Antoine Perreard, and Juniper Gujral. An Apache Giraph implementation of distributed ADMM for solving LASSO problems. In Soft Computing for Problem Solving: Proceedings of SocProS 2020, Volume 2, pages 547–556. Springer, 2021.

[18] Rohit Agrawal, Kapil Ahuja, Marc C. Steinbach, and Thomas Wick. SABMIS: sparse approximation based blind multi-image steganography scheme. PeerJ Computer Science, 8:e1080, 2022.

[19] Yurii Nesterov. Primal-dual subgradient methods for convex problems. Mathematical programming, 120(1):221–259, 2009.

[20] Berhanu Guta. Subgradient optimization methods in integer programming with an application to a radiation therapy problem. PhD thesis, Technische Universität Kaiserslautern, Kaiserlautern, Germany, 2003.

[21] Alexander Z. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. Algorithmica, 9(5):463–470, 1993.

[22] Desiree M. Carvalho and Mariá C. V. Nascimento. Lagrangian heuristics for the capacitated multi-plant lot sizing problem with multiple periods and items. Computers & Operations Research, 71:137–148, 2016.

[23] Oliver G. Czibula, Hanyu Gu, and Yakov Zinder. Lagrangian relaxation-based heuristic to solve large extended graph partitioning problems. In M. Kaykobad and R. Petreschi, editors, WALCOM: Algorithms and computation, Lecture Notes in Computer Science, volume 9627, pages 327–338. Springer, 2016.

[24] Diabat Ali, Battaïa Olga, and Nazzal Dima. An improved Lagrangian relaxation-based heuristic for a joint location-inventory problem. Computers & Operations Research, 61:170–178, 2015.

[25] Kaj Holmberg, Martin Joborn, and Kennet Melin. Lagrangian based heuristics for the multicommodity network flow problem with fixed costs on paths. European Journal of Operational Research, 188(1):101–108, 2008.

[26] Samuel Deleplanque, Kedad S. Safia, and Alain Quilliot. Lagrangean heuristic for a multi-plant lot-sizing problem with transfer and storage capacities. RAIRO-Operations Research, 47(4):429–443, 2013.

[27] Bissan Ghaddar, Joe Naoum-Sawaya, Akihiro Kishimoto, Nicole Taheri, and Bradley Eck. A Lagrangian decomposition approach for the pump scheduling problem in water networks. RAIRO-Operations Research, 241(2):490–501, 2015.

[28] Vaughn Betz and Jonathan Rose. VPR: A new packing, placement and routing tool for FPGA research. In International Workshop on Field Programmable Logic and Applications, pages 213–222, London, UK, 1997.

[29] Yehdhih Moctar, Mirjana Stojilović, and Philip Brisk. Deterministic parallel routing for FPGAs based on Galois parallel execution model. In Proceeding of the 78th International Workshop on Field-Programmable Logic and Applications (FPL), pages 21–25, Dublin, Ireland, 2018.

[30] Wang Yuan. Circuit clustering for cluster-based FPGAs using novel multiobjective genetic algorithms. PhD thesis, University of York, England, 2015.

[31] Kevin E. Murray, Scott Whitty, Suya Liu, Jason Luu, and Vaughn Betz. Timing-driven TITAN: Enabling large benchmarks and exploring the

gap between academic and commercial CAD. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 8(2):1–18, 2015.

ROHIT AGRAWAL received his B.Tech degree in Computer Science and Engineering from BIET Jhansi, India; M.Tech degree in Computer Application from Indian Institute of Technology (Indian School of Mines), Dhanbad, India; and Ph.D. degree in Computer Science and Engineering from Indian Institute of Technology Indore, India. He is currently an Assistant Professor in the Department of Computer Science and Engineering at Madhav Institute of Technology & Science (a government institute), Gwalior, India. In the past, he visited TU Dresden (Germany), Lancaster University (UK), and LU Hannover (Germany) as a guest researcher. His research interests are at the intersection of Computer Science and Mathematical Optimization. Especially, Optimization, Artificial Intelligence, Machine Learning, Data Science, Image Processing, and Circuit Designing.

KAPIL AHUJA received his B.Tech. degree from IIT (BHU) (India), and double M.S. and Ph.D. degrees from Virginia Tech (USA). Subsequently, he was a Postdoctoral Research Fellow at the Max Planck Institute in Magdeburg (Germany). After holding Assistant Professor and Associate Professor positions in Computer Science and Engineering at IIT Indore (India), he is currently a Full Professor here. In the past, he has also been a Visiting Professor at IMT Atlantique (France), LU Hannover (Germany), TU Braunschweig (Germany), TU Dresden (Germany), and Sandia National Labs (USA). At IIT Indore, he heads the Mathematics of Data Science and Simulation (MODSS) research lab, with research interests in Machine Learning, Network Science, Numerical Linear Algebra, and Optimization.

DHAARNA MAHESHWARI received her B.Tech degree in Computer Science from Indian Institute of Technology, Indore, India and she is currently doing her Masters in Computer Science at Columbia University, New York. Her research interests are Optimization and Natural Language Processing.

UBAID SHAIKH received his B.Tech degree in Computer Science and Engineering from the Indian Institute of Technology Indore, India.

He is currently a Compiler Developer at GSI Technology working on LPython and LFortran Compilers. In the past, he has interned at Development Bank of Singapore (DBS Bank) and has been a Google Summer of Code (GSoC) Fellow.

His research interests lie in the domains of Computer Science, Optimization, Programming Languages, Systems and Circuit Designing.

MOHAMED BOUAZIZ received his National Engineering Diploma from École Polytechnique de Tunisie in 2021. During his studies, he was an Erasmus+ visiting student at the University of Trento, Italy, and received a scholarship to undertake his graduation project at the Technical University of Dresden, Germany. He works on electronic design automation for reconfigurable computing.

AKASH KUMAR (Senior Member, IEEE) received the joint Ph.D. degree in electrical engineering and embedded systems from the Eindhoven University of Technology, Eindhoven, The Netherlands, and the National University of Singapore (NUS), Singapore, in 2009. From 2009 to 2015, he was with NUS. He is currently a Professor with Technische Universität Dresden, Dresden, Germany, where he is directing the chair of processor design. His current research interests include the design, analysis, and resource management of low-power and fault-tolerant embedded multiprocessor systems.

• • •