Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2015, Article ID 756014, 10 pages
http://dx.doi.org/10.1155/2015/756014

Research Article

An Improved Diffusion Based Placement Algorithm for Reducing Interconnect Demand in Congested Regions of FPGAs

Ali Asghar and Husain Parvez

Karachi Institute of Economics and Technology, Korangi Creek, Karachi 75190, Pakistan

Correspondence should be addressed to Ali Asghar; ali.asghar@pafkiet.edu.pk

Received 30 June 2015; Accepted 18 October 2015

Academic Editor: Miriam Leeser

An FPGA has a finite routing capacity due to which a fair number of highly dense circuits fail to map on slightly underresourced architecture. The high-interconnect demand in the congested regions is not met by the available resources as a result of which the circuit becomes unroutable for that particular architecture. In this paper, we present a new placement approach which is based on a natural process called diffusion. Our placer attempts to minimize the routing congestion by evenly disseminating the interconnect demand across an FPGA chip. For the 20 MCNC benchmark circuits, our algorithm reduced the channel width for 15 circuits. The results showed on average ~33% reduction in standard deviation of interconnect usage at an expense of an average ~13% penalty on critical path delay. Maximum channel width gain of ~33% was also observed.

1. Introduction

With every passing year, the logic density inside a chip increases approximately following Moores Law [1], which makes more and more complex problems solvable and applications realizable. But with these advancements, the efficiency of CAD tools to map and optimize these new applications is decreasing. These highly dense circuits have introduced new kind of challenge for CAD tool designers, such as longer run-time of algorithms, higher power dissipation, and inhomogeneous heat distribution.

The logic capacity and capability of FPGAs have increased by leaps and bounds over the past two decades but the routing resources have not increased in a commensurate manner. The work of [2] showed that nearly 80–90% area of an FPGA chip is consumed by the programmable interconnect/routing resources, while the remaining 10–20% area goes to the configurable logic blocks (CLBs). CLBs are the principal logic components in an FPGA device; they are formed by the clustering of more simple logic blocks known as Basic Logic Elements (BLEs). To execute computations, the CLBs need to communicate with each other which is made possible by the interconnect fabric/routing network, spread across an FPGA chip. As the logic density inside a chip increases, more and more CLBs will get crammed into a small chip area and they will require more routing resources to communicate.

Therefore, the distribution of CLBs across an FPGA is of pivotal importance as it hugely impacts the utilization of routing resources along with other design metrics. For Example, if several CLBs with high routing demand are brought in close proximity to each other, they may exhaust the routing resources in that particular region and as a result the circuit would become unroutable for that particular architecture.

In this paper, we present a new congestion driven placement approach that attempts to reduce the variations in interconnect usage by diffusion. The major contribution of this work is a new placement cost function which pays attention to both wirelength and congestion.

Diffusion is a natural transport mechanism under the influence of which particles from a region of higher concentration move towards a region of lower concentration. The diffusion of a dye in water, the distribution of heat in a metal plate, and the spreading of gas molecules in a room are all examples of diffusion. In our case, the regions with high-interconnect demand correspond to high concentration regions while the surrounding regions with relatively low-interconnect usage act as low concentration regions. Hence, a concentration gradient exists between the two regions due to which diffusion takes place. During the diffusion process, logic blocks with high-interconnect demand are moved towards regions of relatively low-interconnect demand. After the completion of this diffusion based placement, logic blocks with high-interconnect demand spread out uniformly across an FPGA as a result of which the variations in interconnect demand are reduced and a solution with better routability is produced. The proposed placement approach is ideally suited for circuits which fail to map on slightly underresourced architecture, for which migration to resource-rich architecture is not a viable option. This approach can be extremely beneficial in situations where the size of an FPGA cannot be increased due to high logic utilization ~100%, or in the case of dynamic partial reconfiguration [3, 4] which involves reconfiguration of the specific regions of an FPGA to implement different functions, while the rest of the device continues to operate in the same manner as before. Implementing a new function or modifying an existing function may result in a design whose routing requirements exceed the resources available in the designated reconfigurable section of an FPGA.

The remainder of this paper is organized as follows. Section 2 provides a background to the problem of routability in an FPGA along with some prior research work in the domain of congestion driven placement algorithms and some other techniques which have been used to reduce high-interconnect demand. In Section 3, we propose our diffusion based placement algorithm and introduce the cost function used in this placer along with some techniques used to improve the run-time of our placer and Section 4 presents the experimental details and also some methods to improve the selection procedure of CLBs which are considered for diffusion. In Section 5, we present the experimental results, while Section 6 covers the conclusion and future work.

2. Background and Previous Work

In the standard CAD flow for FPGA, synthesis and technology mapping are followed by clustering. In clustering, LEs are grouped together on the basis of timing and connectivity of mapped netlist without much attention being given to the routability metrics which can lead to congestion. The packing/clustering algorithm used by VPR called T-VPACK [5] is an example where timing optimization and packing each cluster to its capacity are the primary goals for the objective function. The clustering stage is followed by placement in which the clusters are placed onto the fixed array of CLBs. The placement stage can further exacerbate the problem, if the available architectural resources are not taken into account.

The cost function used by the VPR's placement algorithm [6] is wirelength driven; that is, it attempts to optimize the total wirelength of the current placement. The total wirelength is estimated on the basis of a semiperimeter bounding box metric using the following:

$$Wiring\_Cost = \sum_{i=1}^{N} q(i) \cdot \left(bb_x(i) + bb_y(i)\right), \qquad (1)$$

where N is the total number of nets, $bb_x(i)$ and $bb_y(i)$ are the horizontal and vertical span of the net i, and $q(i)$ is a compensation factor for nets with more than three terminals [7].

To reduce high-interconnect variation or congestion, a placement algorithm must pay attention to the routability of the final design. The semiperimeter bounding box metric used by VPR's placement algorithm does not address this issue because a semiperimeter based cost function brings the CLBs as close as possible unaware of the routing resources available in the target architecture. Such a placement could result in a large number of nets getting restricted to a relatively small area of the chip generating regions with high-interconnect demand which in some cases exceeds the resources available in the architecture. To address these issues, the authors of [8] have presented a congestion driven placer which considers the effect of overlapping bounding boxes. A congestion coefficient is generated from a congestion map which indicates the number of bounding boxes or nets which overlap each CLB. This coefficient is then multiplied with the cost function in (1) which penalizes the moves resulting in congestion. Another congestion driven placement approach was presented in [9] which uses the well-known Rent rule to estimate the routing requirements of the design. A novel approach for the reduction of high-interconnect demand [10] is to iteratively perform reclustering until a target channel width constraint is met. If the standard CAD flow fails to route the given netlist on the available channel width, the iterative portion of un/do pack is invoked. This portion consists of 4 stages: it identifies the congested regions and fully unpacks CLBs in that region. Secondly, it reclusters the unpacked LEs (belonging to congested regions) with a smaller cluster size. To accommodate the increased number of clusters, the size of FPGA is incrementally increased. Then, placement is performed using an incremental placer called RePlace [11]. Finally, routing is performed to determine if the given circuit successfully maps or not. If it fails to route, then the CAD flow starts all over again until the given channel width constraint is met.

The approach used in this paper for reducing congestion, that is, diffusion based placement, has been applied in placement algorithms for ASICs. The authors of [12] used diffusion to overcome a very critical postplacement design closure issue called legalization. Generally, CAD for ASICs uses analytic or force-directed algorithms for placement solution but these methods produce an illegal solution due to the overlapping of logic blocks. The work in [12] addresses this issue by using diffusion to produce a legal solution. In addition, this approach also improves other design metrics like heat distribution, routing congestion, and signal integrity.

In our earlier implementation of this algorithm [13], we used a *fixed* approach (discussed in Section 4.2.2), to select the congested CLBs for diffusion. In this work, we have implemented two new techniques for the better selection of CLBs, which show much better results compared to our previous *fixed* approach, while the results for metrics like critical delay and run-time of algorithm which were not reported in [13] are also included. Otherwise, there is no published work in FPGA domain which utilizes diffusion except [14] which utilizes diffusion based placement solution for reducing high temperature spots over an FPGA chip.

Jaffari and Anis [14] applied the concept of diffusion to address the uneven heat distribution problem in an FPGA by targeting thermal uniformity as the main objective function. The proposed placer tool uses a simulated annealing engine with a weighted common driver cost function to account for the issues of wirelength and performance requirements. The CLBs with high temperature are moved away from each other in such a way that the overall thermal profile of the FPGA smooths out. The cost function used in this placer is similar to the wirelength cost function of simulated annealing in which a negative cost is desirable for the move to get accepted. This work shows a significant improvement in standard deviation (up to 51%) and average reduction (up to 73%) in temperature with a 4% penalty in wirelength and delay.

3. Implementation

Our proposed idea focuses on improving (i.e., reducing) the standard deviation of interconnect usage, which attempts to reduce the peak channel occupancy of a congested region; this may later reduce the overall channel width of the architecture. Our congestion driven placer strives to reduce the interconnect variations by evenly spreading the routing demand across the entire chip. We have adopted the same diffusion based placement technique as proposed in [14]. However, instead of using the technique for reducing high temperature spots, as done in [14], we have used the technique for reducing high-interconnect usage. Moreover, unlike the work done in [10], size of FPGA is not increased.

Our placer uses a weighted common driver cost function similar to the one used in [14] except for the fact that the common driver weight in [14] corresponds to thermal cost extracted from the temperature profile of an FPGA while in our case it corresponds to the occupancy (defined in Section 3.1) of each CLB which we call the congestion cost.

3.1. The Cost Function. The equation for calculating the congestion cost is as follows:

$$\mathit{Congestion\_Cost} = \sum_{i=0}^{n} \left(\sum_{j=1}^{n} \frac{c_i \cdot c_j}{r_{ij}} \right), \tag{2}$$

where n is the number of CLBs used in the FPGA, c_i and c_j are the occupancy of the ith and jth CLB while r_{ij} is the distance between the ith and jth CLB, and c_i and c_j are obtained from a preplaced and routed file which we call *congestion file*; this file contains the occupancy of all the used CLBs. The term *occupancy* refers to the number of connections a CLB has with the adjacent routing channels.

The computational complexity of congestion cost function is $O(n^2)$. For large circuits where the utilization of CLBs is very high, this function can be expensive in terms of run-time. To improve the run-time of our placer, several techniques have been implemented which are discussed in Section 3.3.

The cost function of our placer incorporates both the wirelength cost obtained from (1) and the congestion cost:

$$\Delta \mathit{Cost} = \alpha \left(\frac{\Delta \mathit{Wiring\_Cost}}{\mathit{Wiring\_Cost}} \right) + \alpha \left(\frac{\Delta \mathit{Timing\_Cost}}{\mathit{Timing\_Cost}} \right) + (\beta) \left(\frac{\Delta \mathit{Congestion\_Cost}}{\mathit{Congestion\_Cost}} \right), \tag{3}$$

where α is the weight assigned to the wiring and timing costs and

$$\beta = 1 - 2\alpha \tag{4}$$

is the common driver weight assigned to congestion cost. The formula for calculating $\Delta \mathit{Wiring\_Cost}$ is the same as the one implemented in VPR framework [6]. The calculation details for $\Delta \mathit{Congestion\_Cost}$ are discussed in Section 3.3.

3.2. Flow Overview. Our placement flow has three stages as follows:

(1) First, a routine placement is performed using the conventional wirelength-driven cost function (1). Routing is performed using PathFinder routing algorithm [15].

(2) After the completion of placement and routing, a congestion file is generated which contains the occupancy of all used CLBs.

(3) Now, the placement is performed again using the weighted common driver cost function (3); the values for c_i and c_j are obtained from the generated congestion file. Routing is again performed.

3.3. Improvements in Run-Time. The complexity of the congestion cost function is quadratic in nature which makes the run-time of our placer unscalable for large designs. To improve the run-time of our placer, we have made the following attempts.

3.3.1. Calculating the Incremental Change in Congestion Cost. Instead of calculating the overall (global) congestion cost, we calculate the change in congestion cost. Change in congestion cost which results when a block is moved or swapped is added to the global congestion cost which should produce the same result as computing the overall congestion cost.

The equation for calculating $\Delta \mathit{Congestion\_Cost}$ is as follows:

$$\Delta \mathit{Congestion\_Cost} = \sum_{i=0}^{n} \left(c_i \cdot c_j \right) \left(\frac{1}{r_{ij(new)}} - \frac{1}{r_{ij(old)}} \right), \tag{5}$$

where n is the number of used CLBs in the FPGA, c_i is the occupancy of the logic block which is moved, c_j represents the occupancy of all the other logic blocks, and $r_{ij(old)}$ and $r_{ij(new)}$ represent the distance between the ith and jth CLBs before and after the move. For $\Delta \mathit{Congestion\_Cost}$ to be negative in (3), the distance between the congested logic blocks should increase.

3.3.2. Creating a Priority Queue. The incremental Δ*Congestion_Cost* improves the run-time considerably but still suffers from long run-time penalty when the number of used logic elements is very high. To counter this problem, we first sorted the CLBs according to their occupancies and then a certain percentage of CLBs was inserted in the priority queue (discussed in Section 4.2.2). Now, instead of computing Δ*Congestion_Cost* for all used CLBs at each move, only the CLBs present in the priority queue are considered. In our cost function (5), there is a greater probability of acceptance for the moves which involve logic elements having high values of occupancy. When such blocks are pulled away from each other, high negative values of Δ*Congestion_Cost* are achieved. Hence, the idea of considering only a percentage of highly congested elements is justifiable.

4. Experimental Details and Results

4.1. Experimental Setup. The proposed diffusion based placement algorithm has been implemented in the state-of-the-art Versatile Place and Route (VPR) framework. The functions for *Congestion_Cost* and Δ*Congestion_Cost* along with some other modifications have been incorporated in the VPR framework by modifying the *place.c* file (available in the VPR 6.0 release), which contains most of the functions used to execute the placement stage of the FPGA CAD flow, while slight modifications have been performed in several other files to make VPR capable of accepting some new command line arguments, like *common driver weight* and *percentage of CLBs to be inserted in the priority queue*.

4.2. Experimental Details. To fully check the functionality and performance of our proposed placer, we have performed rigorous testing on all the 20 MCNC benchmarks circuits [16] by varying different key parameters which include

(1) common driver weight,

(2) percentage of CLBs inserted in the priority queue (discussed in Section 3.3.1),

(3) varying the cluster size (for $N = 1$ and $N = 4$).

4.2.1. Common Driver Weight. As discussed in Section 3.1, a percentage of the overall change in cost (Δ*Cost*) is assigned to Δ*Congestion_Cost* by a factor known as the common driver weight β; see (3). From our experiments, we observed that increasing the common driver weight too much penalizes the key performance parameters such as area and speed. Since β is the emphasis given to the Δ*Congestion_Cost*, the greater the value of β, the lesser the influence of Δ*Wiring_Cost* on Δ*Cost* or the acceptance (or rejection) of a move. Hence, values of β should be chosen with great care because large values of β can have an adverse effect on the wiring and delay of the circuit.

During experimentation, we empirically tested different values of the common driver weight for the MCNC benchmark suite [16]. Our results revealed that setting a driver weight between 0.1 and 0.25 (10%–25%) generates optimum results.

4.2.2. Percentage of CLBs Inserted in the Priority Queue. As mentioned earlier in Section 3.3, the run-time of our placement algorithm is a strong function of the number of CLBs. To counter this issue, the idea of a priority queue was presented in Section 3.3.2. So now the run-time of the algorithm will depend upon the number of CLBs inserted in the priority queue. To explore the effect of the number of CLBs (inserted in the queue) on the run-time and the placement quality of the solution, three different CLB insertion approaches were used:

(i) Fixed: a fixed number of CLBs like top 5%, 10%, or 20%, and so forth (in terms of occupancy) are inserted in the priority queue.

(ii) Average: we first calculate the average (μ) of all CLB occupancies. The CLBs with occupancies greater than the average value (μ) are inserted in the queue.

(iii) One standard deviation (1σ): after the average occupancy, we calculate the standard deviation (σ) in occupancy values. CLBs with occupancy values greater than $\mu + \sigma$ are inserted in the priority queue.

The rationale for introducing these three different approaches for inserting CLBs in the priority queue will be discussed in Section 5.2.6.

4.2.3. Varying the Cluster Size. To check the performance of our algorithm, we carried out the set of experiments for two cluster sizes: $N = 1$ and $N = 4$. As mentioned earlier in Section 2, a direct consequence of clustering is the increase in congestion because more BLEs inside a CLB means more connections with the adjacent routing channels. For cluster size $N = 1$, we completely eliminate the effect of clustering on the performance of our algorithm. Results reveal that our algorithm manages to reduce the routing channel width not only for cluster size $N = 4$, but also for the less favorable case of netlists with no clustering (i.e., $N = 1$).

5. Results

5.1. Architecture Details. The architecture file used for the simulation of benchmark circuits (with cluster size $N = 4$) is $k4\_N4$, provided with the release of VPR 6.0. For the netlists with cluster size $N = 1$, we have used an older architecture file from the VPR 4.30 release. The parameter values for the $k4\_N4$ architecture are as follows:

(i) The size of CLBs (number of BLEs) is 1, for netlists with cluster size $N = 1$, while its value is 4, for netlists with cluster size $N = 4$.

(ii) The LUT size is $k = 4$, for both $N = 1$ and $N = 4$ netlists.

(iii) For all the benchmark circuits, the size of FPGA is adjusted to such a value which results in logic utilization close to 100%. Logic utilization refers to the number of logic elements in architecture which are utilized when a specific netlist is mapped on it. For example, for a netlist with 896 CLBs (logic elements), to achieve logic utilization close to 100%, the FPGA grid size should be 30 × 30 (900 CLBs).

(iv) The architecture is based on directional wires which are prevalent in the modern commercial architecture due to their better area and delay performance [17].

(v) The type of switch block used in the architecture is Wilton [18] with $Fs = 3$.

(vi) The values for (Fc_{in}) and (Fc_{out}) are 1.0 and 0.25, respectively.

5.2. Simulation Results. We compare our results with the VPRs placement algorithm; the results from VPR are obtained by running it in the wirelength plus timing driven mode (only for cluster size $N = 4$) for placement and routing. The key parameters used for the comparison of results include

(i) wirelength,

(ii) critical path delay (only for cluster size $N = 4$),

(iii) channel width,

(iv) standard deviation (σ) in interconnect demand/occupancy values.

We measure the standard deviation in the occupancy values for all the CLBs. As mentioned earlier, the focus of our proposed placement approach is to reduce the variations/standard deviation in interconnect demand, as a consequence of which the overall channel width of an FPGA decreases. Since, the standard deviation is a measure of the spread of values for a particular group of numbers (in our case, *occupancy* values for all CLBs) or how far away the data values of a set are from mean μ. For a high value of standard deviation, there will be some CLBs with very high *occupancy* values (much greater than μ); on the other hand, some CLBs will have extremely low *occupancy* values (much smaller than μ). The proposed diffusion based placement algorithm attempts to reduce this gap in CLB occupancies by evenly spreading the CLBs with high occupancies across the entire FPGA chip.

5.2.1. Cluster Size $N = 1$. We first consider the CLBs with size $N = 1$; that is, only one BLE is present inside a cluster. As mentioned earlier, this completely eliminates the effect of clustering on the final placement and routing results. Table 1 shows the results produced by VPR (in the wirelength-driven mode) and our proposed diffusion based placement approach for the 20 MCNC benchmark netlists. The first column of Table 1 shows the MCNC benchmark netlists used for the simulation. The second column contains the optimal FPGA sizes which result in 100% logic utilization for each netlist. Column 3 presents the wirelength cost for each netlist, which is the sum of half-perimeter of all the bounding boxes, while the standard deviation (σ), in column 4, shows the standard deviation in interconnect demand, which has been computed by calculating the value of σ for the occupancies of all the CLBs in a particular netlist. Column 5 depicts the final routed channel width achieved for each netlist. The results produced by our placement approach for the 20 MCNC benchmark netlists have been presented in columns 6, 7, and 8 of Table 1. All the key parameters (wirelength, standard deviation, and channel width) for each netlist in Table 1 have been normalized to the results obtained from VPR.

The results produced by our proposed placement approach for cluster size $N = 1$ show on average ~11% reduction in standard deviation of interconnect demand at an expense of an average ~5% penalty on wirelength. For all the benchmark circuits, bounding box cost slightly increased, which is due to the opposing (to the VPR cost function) nature of our proposed algorithm which attempts to pull the logic elements with high occupancy away from each other.

For the two benchmarks *misex3* and *apex4*, channel width was reduced by ~17% and ~14%, respectively. As mentioned earlier, the objective of our placement approach is to reduce the standard deviation in interconnect demand which in turns reduces peak channel occupancy as a result of which channel width decreases. The highest gain (reduction) in standard deviation was observed for *elliptic* (~32%) and *misex3* (~24%). For *misex3*, a gain of (~17%) was also achieved in channel width, which further justifies the idea that significant gains can be achieved in channel width by reducing standard deviation. On the other hand, there was no gain in channel width for *elliptic*, the primary reason for which is the failure of reduction in peak channel occupancy. As mentioned earlier, gains in channel width are possible if reduction in standard deviation decreases the peak interconnect usage in congested regions. The peaks in *elliptic* are all very uniformly distributed; hence, trying to reduce congestion in one region produces congestion in another.

5.2.2. Cluster Size $N = 4$. Now, the effect of clustering is taken into account by simulating the circuits packed with a cluster size $N = 4$. We first present the results obtained for the 20 MCNC benchmark netlists (cluster size $N = 4$) by running VPR in the weighted wirelength plus timing driven mode, with equal emphasis given to the wirelength and timing; that is, equal driver weight is assigned to the two functions. The results shown in Table 2 will be used as a reference for comparing results of our proposed algorithm with VPR. The results of Table 2 show a significant increase in the final routed channel width of the benchmark circuits; this effect is a direct consequence of clustering. Since more logic elements are now crammed inside a cluster (in this case, 4), the CLBs would now require more routing tracks in their adjacent channels to communicate with each other; this will produce congestion in various regions of an FPGA chip. Hence, our placement approach which attempts to minimize the congestion by evenly spreading the routing demand is more suited for the circuits with clustered logic elements.

5.2.3. Fixed. As mentioned earlier in Section 4.2.2, for the cluster size $N = 4$, three different approaches for inserting CLBs in the priority queue have been proposed. This section covers the *fixed* approach, in which a fixed predetermined number of CLBs are inserted in the priority queue. For the results presented in this section, the top 10% CLBs (in terms of occupancy) are inserted in the priority queue.

Table 1: Results produced by VPR for cluster size $N = 1$.

Netlist	FPGA size	VPR			Diffusion based placement		
		Wirelength	σ	Channel width	Wirelength	σ	Channel width
alu4	40 × 40	21826	5.72	10	23269	5.44	10
apex2	44 × 44	33355	5.8	12	35428	5.75	12
apex4	36 × 36	24052	7.56	14	25166	6.78	12
bigkey	42 × 42	20502	3.55	8	20939	3.54	8
des	40 × 40	25365	3.78	10	26137	3.48	10
diffeq	39 × 39	18204	3.82	8	19830	3.61	8
dsip	38 × 38	16891	3.53	8	19830	3.61	8
ex5p	33 × 33	22717	8.9	14	24040	8.83	14
misex3	38 × 38	23585	6.67	12	24534	5.08	10
seq	42 × 42	31026	6.6	12	32226	5.71	12
tseng	33 × 33	12773	3.61	8	14093	3.46	8
s298	44 × 44	16728	3.49	8	17425	3.18	8
elliptic	61 × 61	47603	7	10	52001	4.71	10
frisc	60 × 60	59959	7.14	12	62569	5.78	12
spla	61 × 61	67065	7.72	14	72017	6.57	14
s38417	81 × 81	83355	4.3	8	86291	4.22	8
ex1010	68 × 68	78798	4.88	12	81029	4.62	12
Pdc	68 × 68	98470	9.86	16	104287	7.84	16
s38584.1	81 × 81	79180	4.5	8	87642	3.99	8
clma	92 × 92	140105	6.06	12	151582	5.01	12

Table 2: Results produced by VPR for cluster size $N = 4$.

Netlist	FPGA size	Critical path (ns)	σ	Channel width
alu4	20 × 20	7.3962	17.002	32
apex2	23 × 23	10.5786	17.1328	36
apex4	19 × 19	7.9168	20.638	38
bigkey	36 × 36	4.8433	8.082	20
clma	47 × 47	16.3227	29.769	50
des	42 × 42	7.7112	10.283	20
diffeq	20 × 20	7.39792	16.43	26
dsip	36 × 36	4.76876	7.735	22
elliptic	31 × 31	10.716	26.405	40
ex5p	17 × 17	8.00032	19.165	38
ex1010	35 × 35	12.4845	21.412	38
frisc	30 × 30	12.3652	26.615	44
misex3	19 × 19	9.45672	17.506	34
pdc	35 × 35	14.9399	25.374	54
s298	23 × 23	11.8196	18.741	32
s38417	40 × 40	9.15104	16.148	30
s38584.1	40 × 40	8.11126	20.906	32
seq	22 × 22	8.31452	17.723	34
spla	31 × 31	10.037	30.291	48
tseng	17 × 17	5.99362	13.987	20

The results produced by the *fixed* approach for the 20 MCNC benchmark netlists have been presented in Figures 1, 2, and 3. All the key parameters (channel width, critical path delay, wirelength, and standard deviation) for each netlist have been normalized to the results obtained from VPR shown in Table 2.

The results for the *fixed* approach show on average ~ 14% reduction in standard deviation at an expense of ~5% penalty on the critical path delay. The channel width has been reduced for 4 benchmark circuits, with the highest gain achieved for *diffeq* ~8% (see Figure 1), which also registered the highest reduction in standard deviation with a value of ~ 35% (Figure 2), at an expense of (~9%) penalty on the critical path delay (Figure 3).

5.2.4. Average. The second approach for inserting CLBs in the priority queue is based on the calculation of average of (μ) CLB occupancies. After calculating the average (μ) occupancy for all CLBs, logic blocks with occupancies greater than μ are inserted in the priority queue. Table 3 shows the percentages of CLBs inserted in the priority queue for the 20 MCNC benchmark circuits, having occupancies greater than μ.

The results for the *average* approach show on average ~21% reduction in standard deviation at an expense of ~4% penalty on the critical path delay. The channel width has been reduced for 9 benchmark circuits, with the highest gains achieved for *dsip* ~9% (see Figure 4), which also showed a reduction in standard deviation of interconnect demand by ~39% (see Figure 5). On the other hand, the critical path delay for *dsip* deteriorated by ~28% (see Figure 6).

5.2.5. One Sigma (1-σ). The third approach for inserting CLBs in the priority queue involves the calculation of standard

Table 3: Percentage of CLBs inserted in the priority queue for *average* approach.

Netlist	% CLBs inserted in the priority queue
alu4	58.418
apex2	58.436
apex4	62.31
bigkey	66.823
diffeq	64.986
dsip	63.929
elliptic	61.878
ex5p	60
s298	54.141
ex1010	58.652
frisc	62.597
s38417	60.45
s38584.1	56.065
pdc	62.794
spla	57.471264
seq	61.160714
clma	57.787076
tseng	60.076046
des	62.043796
misex3	59.833795

Table 4: Percentage of CLBs inserted in the priority queue for *1-sigma* approach.

Netlist	% CLBs inserted in the priority queue
alu4	13.520408
apex2	14.403292
apex4	14.285714
bigkey	13.882353
diffeq	10.079576
dsip	14.662757
elliptic	12.486188
ex5p	12.857143
s298	18.787879
ex1010	14.725458
frisc	13.325868
s38417	13.579474
pdc	7.912458
seq	13.616071
spla	16.9279
s38584.1	18.416091
misex	16.620499
clma	16.643422
tseng	13.987093

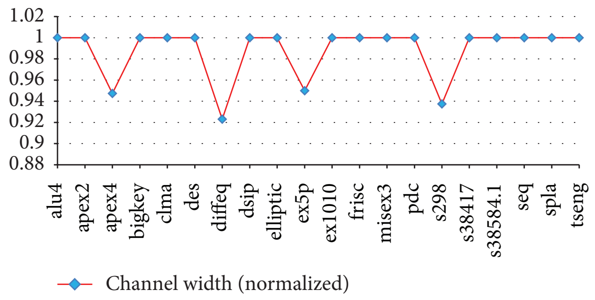

Figure 1: Comparative results of channel width for diffusion based placement algorithm using *fixed* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

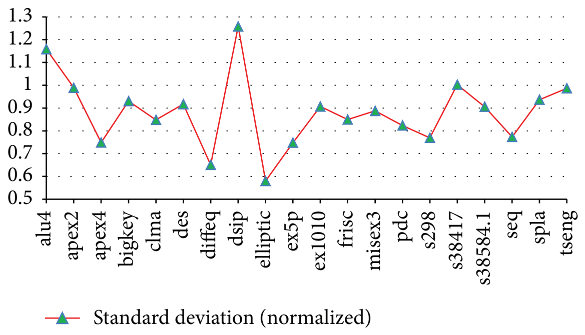

Figure 2: Comparative results of standard deviation in interconnect demand for diffusion based placement algorithm *fixed* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

deviation (σ) in CLB occupancies. After calculating the average (μ) occupancy for all CLBs, logic blocks with occupancies greater than $\mu + \sigma$ are inserted in the priority queue. Table 4 shows the percentages of CLBs inserted in the priority queue for the 20 MCNC benchmark circuits, having occupancies greater than $\mu + \sigma$.

The results for the *1-sigma* approach show on average ~33% reduction in standard deviation at an expense of ~13% penalty on the critical path delay. The channel width has been reduced for 15 benchmark circuits, with the highest gains achieved for *dsip* ~33% and *bigkey* ~30% (see Figure 7), which also showed a reduction in standard deviation of interconnect demand by ~22% and ~40% (see Figure 8), respectively. On the other hand, the critical path delay for *bigkey* suffered heavily ~44% (see Figure 9) due to the extra ordinary gain achieved in the channel width. This huge plenty in the critical path delay of *bigkey* results from the pulling away of highly congested logic blocks from each other, which in the case of *bigkey* lie on the critical path. Hence, the logic blocks on the critical path which should be very close to each other, (to minimize the time required for the signals to traverse the critical path) encountering minimum wirelength and the smallest possible number of switch blocks, are now far away from each other, as a result of which the overall critical delay of the final solution has increased.

5.2.6. Comparison of the Three Approaches. Out of the three proposed approaches for inserting CLBs in the priority queue, the *1-sigma* approach produced the best results. The reason for the better performance of *1-sigma* approach is due to the number of CLBs inserted in the priority queue. As mentioned earlier in Section 3.3.2, logic blocks with high

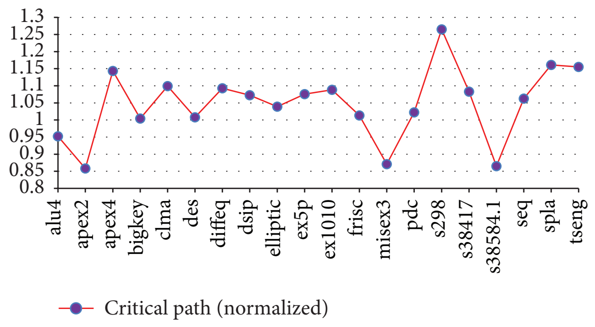

Figure 3: Comparative results of critical path for diffusion based placement algorithm *fixed* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

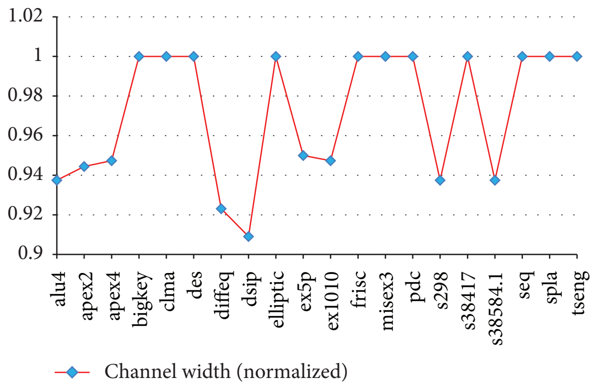

Figure 4: Comparative results of channel width for diffusion based placement algorithm using *average* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

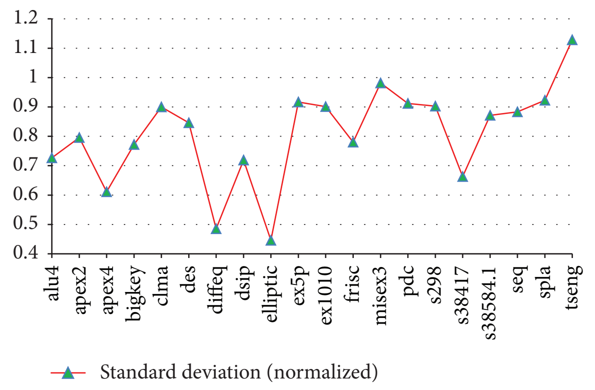

Figure 5: Comparative results of standard deviation in interconnect demand for diffusion based placement algorithm using *average* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

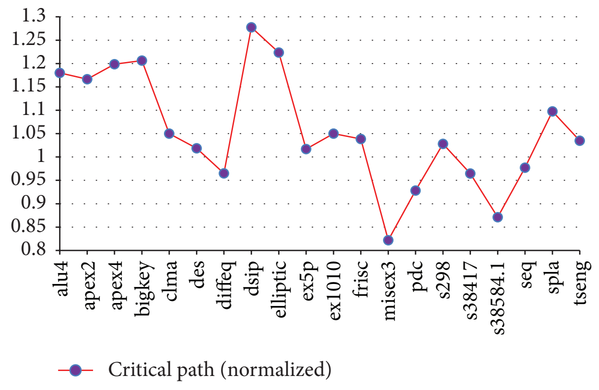

Figure 6: Comparative results of critical path for diffusion based placement algorithm using *average* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

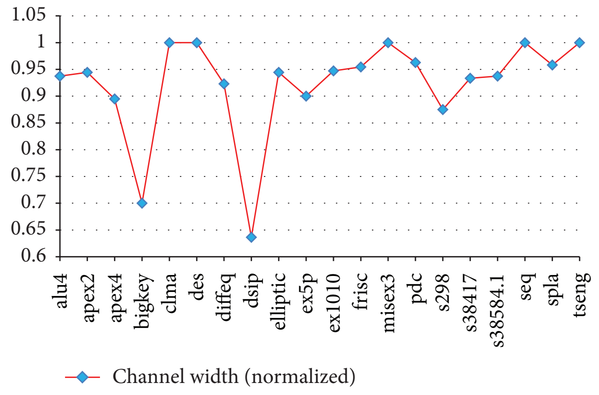

Figure 7: Comparative results of channel width for diffusion based placement algorithm using *1-sigma* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

occupancy are mainly responsible for congestion; hence, inserting them in the priority queue seems to be a feasible idea. But, what should be the criterion for a CLB to be considered for a priority queue? In some netlists, there are a large number of congested logic blocks; should we insert all of them in the priority queue or probably some percentage of them? The results obtained from the three mentioned approaches answer these questions.

The *fixed* approach which considers only a specific percentage of CLBs from a group of highly congested logic blocks manages to reduce channel width for only 4 circuits, with the highest gain in channel width achieved for *diffeq* (~ 8%). Although this approach manages to achieve the eventual goal of our proposed placement algorithm, which targets rectifying the interconnect imbalance by evenly spreading the highly congested logic elements across an FPGA chip (as a result of which there is a reduction in the routed channel width of an FPGA), the results produced by the *fixed* approach lack the glare to be considered as an efficient way of determining the percentage of CLBs to be inserted in the priority queue. Since different benchmark circuits have a different number of congested logic blocks, inserting the same fixed percentage of CLBs for all the circuit will not produce the desired results for every circuit.

The second approach for determining the percentage of CLBs was based on *average* approach in which CLBs with occupancies greater than μ are inserted in the priority

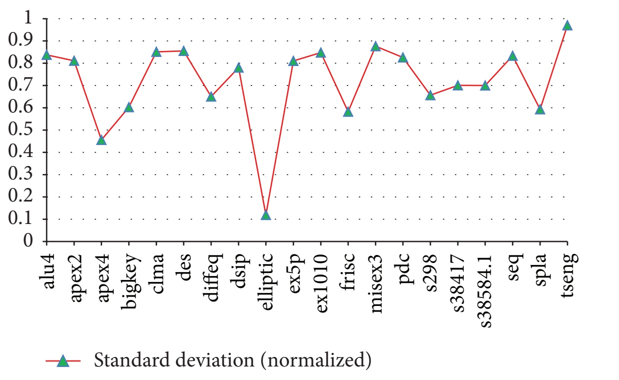

Figure 8: Comparative results of standard deviation in interconnect demand for diffusion based placement algorithm using *1-sigma* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

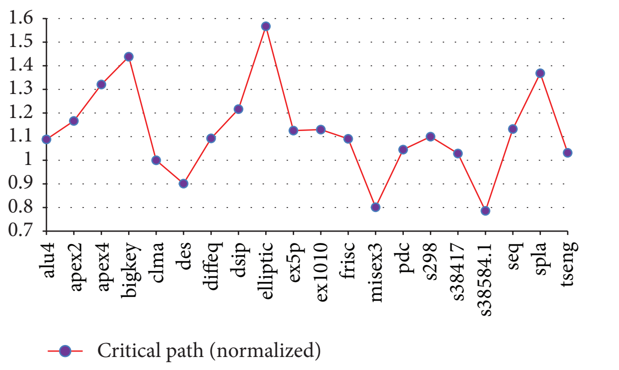

Figure 9: Comparative results of critical path for diffusion based placement algorithm using *1-sigma* approach normalized to the respective results from VPRs placement algorithm for cluster size $N = 4$.

queue. Table 4 shows the percentage of CLBs inserted in the queue for the *average* approach. The percentages for all the benchmark circuits are on the higher side (greater than 50% for all the netlists); inserting a high percentage of CLBs for netlists with a relatively small number of congested blocks can have an adverse effect on the performance of the algorithm, because a large number of CLBs will start competing for diffusing into low-interconnect demand regions which may result in CLBs with high occupancy (which are mainly responsible for congestion) failing to diffuse into low-interconnect demand regions. This is the reason for the modest quality of the results obtained for the *average* approach, which managed to reduce channel width for 9 benchmark circuits, with the highest gain in channel width achieved for *dsip* (~9%).

From the three proposed approaches, the 1-sigma (1-σ) approach produced the best results, reducing the channel width for 15 benchmark circuits, with the highest gain in channel width achieved for *dsip* (~33%). The reason for this performance is the better selection process for CLBs which are inserted in the priority queue. For the 1-σ approach, CLBs with occupancies greater than $\mu + \sigma$ are inserted in the queue; as a result, only CLBs with really high occupancies (which are primarily responsible for routing congestion) find themselves in the priority queue. Also, the 1-σ approach is highly adaptive as it can vary the percentage of logic blocks considered for the queue, in accordance with the number of congested blocks in a netlist. The percentages shown in Table 4 depict this adaptive behaviour of the 1-σ approach, where the percentage of inserted CLBs varies between ~19% for *s298* and ~10% for *bigkey*.

In terms of run-time, the *average* approach is the slowest with a ~3x penalty in placement time compared to the VPR's placement algorithm, while the *fixed* and *1-sigma* approaches are, respectively, ~1.05x and ~1.4x slower than VPR. As mentioned earlier, the run-time of our algorithm is directly related to the number of CLBs inserted in the priority queue; hence, the heavy run-time penalty for the *average* approach is due to the high percentages of CLBs inserted in the queue. On the other hand, the *fixed* and *1-sigma* approaches experience modest penalties in the run-time due to a nominal number of CLBs inserted in the queue.

6. Conclusion and Future Work

Despite the availability of modern resource-rich architecture, there will always be some circuits whose interconnect demand can exceed the available routing resources in particular architecture. In this work, we have proposed a new diffusion based placement algorithm which attempts to minimize the variations in interconnect demand by uniformly spreading the routing congestion across an FPGA chip to decrease the peak channel occupancy which as a result reduces the overall routing channel width. Our proposed algorithm targets the circuits which have a routing demand slightly above the resources available in the architecture, for which migration to resource-rich architecture is not a viable option, and unlike reclustering [10] which can achieve substantial reduction in channel width by increasing the FPGA size, this placement approach does not require an increase in the FPGA size and, hence, is the ideal option to reduce the channel width in situations when the logic utilization in an FPGA is close to 100% or the size of FPGA cannot be increased; however, the gain in channel width achieved for this approach is fairly modest as compared to reclustering. Furthermore, the routing channel width required by our placement approach never exceeds VPR which proves its stability.

The benchmarks that we used for experimentation lacked the inherent local congestion which resulted in smaller or no gains in standard deviation of interconnect usage and channel width for some netlists. Hence, testing the functionality of our placer on much larger set of synthetic benchmarks as used in [10] will make the picture much clearer. We would also like to test the performance of our algorithm on some commercially available FPGA devices from vendors like Altera. To do so, we intend to make use of QUIP [19] toolkit which allows students and researchers to access the Quartus ll CAD suite at different stages of the CAD flow. We intend to pass the technology mapped netlist file from Quartus ll to our placer which will perform the placement of the design; the placement information will then be passed back to the Quartus ll CAD flow for routing. The routing results generated using this modified flow will then be compared with the results produced by the complete Altera flow. Another approach for calculating $\Delta Congestion\_Cost$ is under consideration. Instead of placing all the clusters in a single placement iteration, each cluster can be individually placed on different iterations. Hence, the placer will iteratively run several times where the number of placer iterations will be equal to the number of clusters. We expect much better results for both run-time and standard deviation after the successful completion of this future work.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was sponsored in part by the National ICT R&D Fund, Pakistan. We are indebted to their generous support.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, pp. 114–117, 1965.

[2] A. DeHon, "Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization)," in *Proceedings of the 7th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '99)*, pp. 69–78, ACM, Monterey, Calif, USA, February 1999.

[3] Altera, *Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs*, Altera Corporation, 2010.

[4] D. Dye, *Partial Reconfiguration of Xillinx FPGAs Using ISE Design Suite*, Xilinx Corporation, 2012.

[5] V. Betz and J. Rose, "Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 551–554, IEEE, Santa Clara, Calif, USA, May 1997.

[6] V. Betz and J. Rose, "VPR: a new packing placement and routing tool for FPGA research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pp. 213–222, London, UK, September 1997.

[7] C.-L. E. Cheng, "RISA: accurate and efficient placement routability modeling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 690, pp. 690–695, 1994.

[8] Y. Zhuo, H. Li, and S. P. Mohanty, "A congestion driven placement algorithm for FPGA synthesis," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 1–4, IEEE, Madrid, Spain, August 2006.

[9] G. Parthasarathy, M. Marek-Sadowska, A. Mukherjee, and A. Singh, "Interconnect complexity-aware FPGA placement using Rent's rule," in *Proceedings of the International Workshop on System-Level Interconnect Prediction (SLIP '01)*, pp. 115–121, ACM, Rohnert Park, Calif, USA, April 2001.

[10] M. Tom, D. Leong, and G. Lemieux, "Un/Do pack: re-clustering of large system-on-chip designs with interconnect variation for low-cost FPGAs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '06)*, pp. 680–687, IEEE, San Jose, Calif, USA, November 2006.

[11] D. Leong and G. G. F. Lemieux, "Replace: an incremental placement algorithm for field programmable gate arrays," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 154–161, IEEE, Prague, Czech Republic, August-September 2009.

[12] D. Z. Pan, H. Ren, C. J. Alpert, P. G. Villarrubia, and G.-J. Nam, "Diffusion-based placement migration with application on legalization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 12, pp. 2158–2172, 2007.

[13] A. Asghar and H. Parvez, "Diffusion-based placement algorithm for reducing high interconnect demand in congested regions of FPGAs," in *Reconfigurable Computing: Architectures, Tools, and Applications*, vol. 8405, pp. 291–297, Springer, 2014.

[14] J. Jaffari and M. Anis, "Thermal driven placement for island-style MTCMOS FPGAs," *Journal of Computers*, vol. 3, no. 4, pp. 24–30, 2008.

[15] L. McMurchie and C. Ebeling, "A negotiation-based performance-driven router for FPGAs," in *Proceedings of the 1995 ACM 3rd International Symposium on Field-Programmable Gate Arrays*, pp. 111–117, Monterey, Calif, USA, February 1995.

[16] MCNC, "LGSynth93 benchmark suite," Tech. Rep., Microelectronics Centre of North Carolina, Durham, NC, USA, 1993.

[17] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 41–48, Brisbane, Australia, December 2004.

[18] S. J. Wilton, *Architecture and algorithms for field programmable gate arrays with embedded memory [M.S. thesis]*, University of Toronto, 1997.

[19] Altera, *Quartus-II University Interface Program*, Altera Corporation, 2009.