Simultaneous Timing Driven Clustering and Placement for FPGAs

Gang Chen and Jason Cong

Computer Science Deparment, University of California,
Los Angeles, CA 90095, USA
{chg, cong}@cs.ucla.edu

Abstract. Traditional placement algorithms for FPGAs are normally carried out on a fixed clustering solution of a circuit. The impact of clustering on wirelength and delay of the placement solutions is not well quantified. In this paper, we present an algorithm named *SCPlace* that performs simultaneous clustering and placement to minimize both the total wirelength and longest path delay. We also incorporate a recently proposed path counting-based net weighting scheme [16]. Our algorithm *SCPlace* consistently outperforms the state-of-the-art FPGA placement flow (*T-VPack* + *VPR*) with an average reduction of up to 36% in total wirelength and 31% in longest path delay.

1 Introduction

A typical LUT-based FPGA architecture [1] contains a two-level physical hierarchy: Basic Logic Elements (BLE) and Cluster-based Logic Blocks (CLB). As described in **Fig. 1**, each BLE contains one *K*-input LUT and one flip-flop (FF), and the LUT and FF share the same output. As described in **Fig. 2**, each CLB contains *N* BLEs, *I* inputs and *N* outputs. Each of the *I* inputs can drive all the BLEs, and each BLE drives an output. Here *K*, *N*, and *I* are parameters described by the architecture file. The interconnect delay between BLEs within the same CLB is usually much smaller than the delay between BLEs in different CLBs.

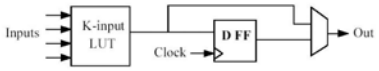

Fig. 1. *VPR*'s Basic Logic Element (BLE)

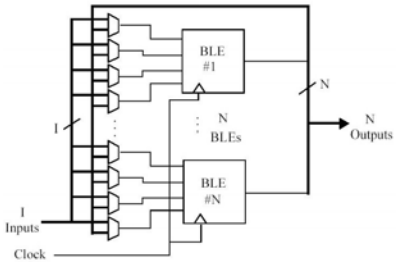

Fig. 2. *VPR*'s Cluster-Based Logic Block (CLB)

In a typical FPGA design flow, a circuit is first synthesized and mapped into a netlist of LUTs and FFs. Then it goes through the following three steps: clustering, placement and routing. The clustering step arranges LUTs and FFs into CLBs according to the timing and the connectivity of the mapped netlist; the placement places the clustered netlist onto the array of on-chip CLBs; the routing routes all the wires in the netlist with the available routing resources on the device.

The drawback of this design flow is that the clustering and placement stages are artificially separated. During the clustering stage, we have great freedom to change a circuit's structure, but a fast and accurate estimation of the final placement wirelength, timing and routability information is not available. During the placement stage, we can optimize wirelength, timing and routability simultaneously, but the solution space is greatly confined because we are committed to a fixed circuit structure. Since the mistakes made during the clustering phase cannot be corrected during the placement process, it will ultimately generate a sub-optimal place and route result.

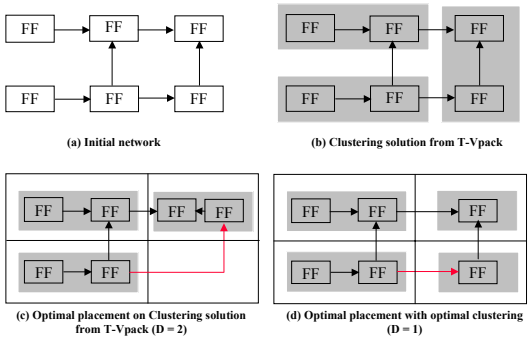

Fig. 3. Impact of Clustering on Placement

Fig. 3 illustrates the impact of clustering on placement. The initial network (a) consists of six FFs. We assume each CLB contains two BLEs, and the device is a 2 x 2 grid. The delay model used here is the Manhattan distance. The "optimal" clustering solution in (b), which consists of three CLBs and one logic level, can be obtained from *T-VPack* [17] (which minimizes the number of clusters and the number of levels). However, the optimal placement solution (c) on this optimal clustering has a longest path delay of two. Instead, when we perform clustering together with placement, we can obtain a placement solution (d) with a longest path delay of one.

In this paper, we propose a novel algorithm to perform clustering optimization during the placement for wirelength and timing minimization. We also incorporate a recently proposed path counting-based net weighting scheme in our approach. This new algorithm outperforms the current state-of-the-art FPGA placement flow *T-VPack + VPR* with an average reduction of up to 36% in total wirelength and 31% in longest path delay. Another significant contribution is that our combined approach has a runtime complexity similar to the existing *VPR* placement algorithm.

2 Review of Existing FPGA Clustering and Placement Algorithms

Packing LUTs and FFs into CLBs is a critical step in the cluster-based FPGA design flow, since it has a great impact on both timing and routability. *VPack* [17] packs each logic block to its capacity to minimize the number of clusters and encourages input sharing to minimize the number of connections between clusters. The timing-driven version, *T-Vpack* [17], minimizes the number of connections on the critical path since on average the internal connections are much faster than the external connections. *Rpack* [4] introduces an effective routability metric and presents a routability driven clustering algorithm for cluster-based FPGAs. *PRIME* [10] integrates retiming with performance-driven clustering/partitioning. For a given area bound for each cluster, if duplication is allowed, *PRIME* can generate a quasi-optimal solution with a delay of no more than a small constant over the minimal delay.

Placement is a classic problem and becomes increasingly difficult and important as the design size rapidly increases. There are three classes of widely used placement methods: min-cut based placer [11][5][23], analytical placer [12][15][20] and simulated annealing-based placer [14][21][1]. Min-cut based placers recursively partition the circuit until the number of cells in each partition is small enough and then assign cells to appropriate rows. The min-cut based methods are usually very fast, but since the cutsize is not an exact function of either wirelength, timing or routability, the quality is not as good as other placers. The analytical method includes the force directed and quadratic programming method. The force directed method introduces attracting, repelling and other additional forces and then solves a linear equation system using the forces. The quadratic programming (QP) method solves the placement problem by solving a sequence of quadratic programming problems derived from the circuit connectivity information. The force directed and quadratic-programming methods have a short runtime and produce good results, but they are not flexible enough to handle complex constraints. The simulated annealing algorithm simulates the annealing process that is used to produce high-quality metal structures by gradually cooling down the temperature. The initial placement is gradually optimized by performing a number of moves at each temperature. Each move is accepted with a certain probability $p = e^{-delta\_cost/T}$, where *delta_cost* is the change in cost function and T is the current temperature. Simulated annealing-based placers are very flexible for handling different kinds of constraints, and they usually generate a good solution in a reasonable amount of time. In recent years there have been several novel placement algorithms that incorporate multiple placement techniques. For example, *Mongrel* [13] adopts a *middle-down* methodology in which a *global placement* solution is obtained by placing logic cells into coarse bins. During the global placement phase, a *Relaxation Based Local Search* methodology is applied to generate global complex modifications to the current placement. A novel *ripple move* [13] based legalization procedure is also presented. After the global placement is completed, a detailed placement is obtained by applying the *optimal interleaving* [13] technique. *Dragon2000* [22] uses a top down hierarchical approach, and integrates the partitioning-based cutsize minimization techniques and the simulated annealing-based wirelength minimization techniques. mPL [6] and mPG [7] are based on the multilevel framework to improve both runtime and quality of the placement

3 Simultaneous Timing Driven Clustering and Placement Algorithm

3.1 Overview

Our algorithm uses a simulated annealing-based optimization engine [21][1][18]. We first perform an initial clustering on the mapped netlist, and then generate a random placement of the clustered netlist. During the annealing process, we optimize the clustering structure and circuit placement at the same time. To improve the sub-optimal clustering structure during placement, we introduce a *fragment level move*. After each move, we update the cost function and decide whether to keep the move or not. We iteratively perform a certain number of moves at each temperature and then reduce the temperature until the acceptance rate is too low. In order to optimize both wirelength and circuit delay, we minimize a weighted function of bounding-box wirelength cost and timing cost (weighted edge delays). For the net weighting, we implement a recently proposed path counting-based net weighting scheme.

3.2 Clustering Optimization During Placement

Our main contribution is to perform clustering optimization during placement. There are two types of moves in our approach. The first type of move is the *block level move*, in which an entire CLB is moved to a new location and swapped with another CLB if necessary. The second type of move is the *fragment level move*, in which only a BLE is moved to a new CLB and swapped with another BLE if necessary. Due to the powerful *fragment level move*, we are able to significantly improve the sub-optimal clustering structure to achieve a high quality placement. This is especially important when the chip utilization is high, and the clustering stage has to perform unrelated packing to squeeze the design into the device. Due to the lack of physical information, it is almost impossible for a clustering algorithm to make the right packing decisions among unrelated logics. With the simultaneous clustering optimization and placement optimization, we can correct mistakes made during the previous stage and significantly improve both routability and timing.

When we perform a *fragment level move*, we need to check whether the new CLB is in a valid configuration. When we check the feasibility of each CLB, we need to check the number of BLEs and the number of inputs. For real industry architectures, we also need to check the number of clocks, the number of feedbacks, the number of control signals, etc. Hence, we dynamically update a hash-map for each involved CLB whenever a *fragment level move* is performed. The complexity of the update is $O(K)$, where K is the input size of the LUT.

3.3 Path Counting-Based Net Weighting

The net-based timing-driven placers (e.g. [18]) convert timing information into net weight and optimize a weighted function of all nets. The basic idea of net weighting

is to assign higher weights to timing critical nets and lower weights to non-critical nets. The net weighting scheme is both efficient and flexible enough to handle complex constraints, but most existing methods do not take into account the path information.

Here we incorporate a novel net weighting scheme [16] proposed by Dr. T. Kong, which accurately counts all paths (critical and non-critical) for certain types of discount functions such as $D(x, y) = a^{-x/y}$. This scheme considers path sharing, and thus assigns a higher weight to the edges shared by two or more critical paths. For more details about path counting, please refer to [16].

4 Runtime/Quality Trade-Off

For a given architecture, each CLB contains N BLEs, I inputs and N outputs. In the input clustered netlist, the number of CLBs is n, and the number of BLEs is m. $n \leq m \leq N*n$, and $O(m) = O(N*n) = N*O(n)$. If every swap performed at each temperature is at the BLE level, the number of swaps needed will be $O((N*n)^{4/3})$, which is quite costly.

However, we perform both block level move and fragment level move in our approach. At each temperature, the number of block level moves performed is $n^{4/3}$, and the number of fragment level moves performed is $(\alpha*m)^{1.33} \approx (\alpha*N*n)^{1.33}$. We can change the value of α between 0 and 1, and achieve the runtime/quality trade-off.

5 Complexity Analysis

We first analyze the computation complexity of *VPR*'s placement engine T-VPlace [18]. The timing analysis is performed once per temperature change, which is an $O(n)$ operation. At each temperature the inner loop of the placer is executed $O(n^{4/3})$ times (i.e., $O(n^{4/3})$ swaps are performed). In the inner loop is the incremental-bounding-box-update operation that is worst case $O(k_{max})$, where k_{max} is the fanout of the largest net in the circuit. The average case complexity for this bounding box update is $O(1)$ [2][3]. Also in the inner loop is the computation of the *Timing_Cost* for each connection affected by a swap. This is also $O(k_{max})$. In the average case this is $O(k_{avg})$ where k_{avg} is the average fanout of all nets in the circuit. Since k_{avg} is typically quite small, the average complexity of this *Timing_Cost* computation is $O(1)$ as well. The overall result is that the *VPR* algorithm is worst case $O[k_{max} \cdot (n)^{4/3}]$, but on average it is $O(n^{4/3})$. The average case complexity is really the only relevant value here. The complexity of the algorithm is the average over millions of swaps, so a user will always see the average case complexity.

In our algorithm *SCPlace*, at each temperature the complexity of the block level moves is $O(n^{4/3})$, and the complexity of the fragment level move is $O((\alpha*N*n)^{4/3})$. In reality, the value of N is not very big, and we can always choose α to make

$O((\alpha*N*n)^{4/3}) = O(n^{4/3})$. Hence, the overall complexity is $O(n^{4/3}+n^{4/3}) = O(n^{4/3})$. As a result, our algorithm's complexity can be similar to *VPR*, and hence very scalable.

6 Experimental Results

We implemented our algorithm *SCPlace* under the *VPR* framework. For the purpose of comparison, we downloaded the *VPR* 4.3 source code, architecture file and the complete set of 20 MCNC benchmark circuits used by *VPR* from [24]. We modified the architecture file to specify the number of BLEs contained in a single CLB. For all of the 20 MCNC circuits, we compare with the commonly used academic FPGA design flow [17]. We first run the *script.algebraic* in SIS [19], followed by *Flowmap* [9]. Then we run *T-VPack* [17] to generate an initial clustering solution. This initial clustering is then given to both *VPR* and *SCPlace* to perform placement. The default architecture we use assumes that each CLB contains 4 LUTs, and each LUT has 4 inputs. In section **6.1** and **6.2**, we perform 100% fragment moves and no block moves. In section **6.3** we perform both block and fragment moves and explore the trade-off between quality and runtime. Only the runtime of the second half of benchmark set is reported since the circuits in the first half are too small.

6.1 Wire-Length Comparison

Table 1. Wirelength Comparison with *T-VPack + VPR*

Circuit	*VPR*	*SCPlace*	Improvement
ex5p	112.47	98.91	13.71%
apex4	113.639	101.45	12.02%
misex3	123.616	105.64	17.02%
Tseng	94.9456	70.57	34.55%
alu4	123.03	104.68	17.53%
dsip	195.544	138.69	41.00%
seq	173.641	152.99	13.50%
diffeq	132.271	107.20	23.39%
apex2	190.324	165.73	14.84%
s298	166.899	164.96	1.17%
des	278.122	257.286	8.10%
bigkey	171.986	196.81	-12.61%
spla	426.227	352.635	20.87%
elliptic	359.011	284.821	26.05%
ex1010	463.618	364.774	27.10%
pdc	704.286	580.969	21.23%
frisc	584.732	482.289	21.24%
s38584.1	576.457	354.476	62.62%
s38417	696.701	494.657	40.85%
clma	1701.02	1271.88	33.74%
Average			**21.89%**

Table 2. Impact of Architecture on Wirelength

Circuit	*CLB=2*	*CLB=4*	*CLB=6*	*CLB=8*	*CLB=10*
ex5p	8.14%	13.75%	15.33%	19.66%	23.02%
apex4	4.33%	13.41%	22.02%	25.10%	28.01%
misex3	6.75%	14.36%	19.17%	20.28%	17.11%
Tseng	14.70%	34.42%	30.72%	33.41%	36.11%
alu4	10.36%	19.24%	18.59%	22.57%	17.20%
dsip	-12.77%	39.57%	56.25%	75.67%	70.18%
seq	5.34%	16.51%	19.78%	21.18%	24.71%
diffeq	6.47%	23.01%	34.68%	35.88%	40.03%
apex2	3.08%	15.08%	15.99%	25.35%	21.41%
s298	-2.06%	0.73%	-0.22%	4.05%	4.53%
des	1.62%	7.22%	19.56%	12.41%	23.19%
bigkey	-20.79%	-12.61%	-7.76%	14.66%	36.21%
spla	13.37%	21.30%	26.25%	26.44%	27.21%
elliptic	9.23%	25.23%	42.19%	35.79%	45.67%
ex1010	13.87%	27.99%	43.18%	43.43%	52.82%
pdc	12.76%	19.97%	25.45%	28.09%	32.44%
frisc	3.15%	16.38%	28.35%	27.06%	36.00%
s38584.1	25.04%	60.29%	71.14%	68.35%	75.66%
s38417	23.84%	43.79%	51.20%	47.81%	59.23%
clma	14.92%	35.27%	41.90%	53.35%	54.96%
Average	**7.07%**	**21.75%**	**28.69%**	**32.03%**	**36.28%**

In **Table 1**, we compare our algorithm *SCPlace* with *VPR* using the total weighted bounding box wire lengths as the only optimization objective. The weights for nets of different sizes can be found in [8]. When we combine clustering with placement, we can outperform *VPR* by 22% on average.

In **Table 2**, we illustrate the impact of architecture on the wirelength improvement obtained from *SCPlace*. When we change the size of the CLB (*N)* from 2 to 10, the wirelength gap between *SCPlace* and *T-Vpack+VPR* increases monotonically from

7% to 36%. The result shows that as the size of CLB increases, it is more and more difficult to generate a good clustering solution with small wirelength without physical information. Since *SCPlace* explores different clustering solutions during the placement stage, it generates clustering and placement solutions with much shorter wirelength.

6.2 Timing Comparison

Table 3. Timing Comparison with *T-VPack + VPR*

Circuit	*VPR*	*Path count*	%	*Fragment*	%	*Frag +path count*	%
ex5p	50.45	44.95	12.24%	44.65	12.99%	40.75	23.80%
apex4	47.44	44.71	6.12%	44.02	7.77%	41.44	14.49%
misex3	51.04	44.15	15.61%	43.91	16.24%	38.53	32.47%
tseng	38.85	36.43	6.65%	35.89	8.24%	35.11	10.65%
alu4	53.16	45.46	16.95%	47.51	11.89%	42.50	25.07%
dsip	38.32	40.96	-6.45%	38.32	0.00%	40.12	-4.49%
seq	51.26	46.56	10.11%	43.97	16.59%	42.90	19.51%
diffeq	47.73	38.76	23.15%	39.58	20.60%	41.15	16.01%
apex2	56.36	50.97	10.58%	52.37	7.62%	47.23	19.34%
s298	87.36	90.76	-3.74%	89.31	-2.18%	80.98	7.88%
des	83.88	67.15	24.91%	74.39	12.75%	65.44	28.18%
bigkey	41.37	40.95	1.03%	43.35	-4.57%	41.35	0.03%
spla	72.47	63.12	14.81%	64.09	13.07%	58.27	24.35%
elliptic	71.07	55.72	27.54%	59.17	20.11%	48.49	46.58%
ex1010	97.88	79.10	23.75%	86.85	12.70%	74.82	30.81%
pdc	113.15	76.95	47.04%	78.44	44.24%	67.60	67.38%
frisc	81.39	92.53	-12.0%	79.82	1.97%	75.73	7.47%
s38584.1	64.37	46.66	37.96%	56.93	13.07%	47.78	34.71%
s38417	76.63	70.15	9.24%	56.89	34.71%	49.89	53.61%
clma	137.20	116.8	17.52%	113.8	20.61%	102.0	34.52%
Average			**14.15%**		**13.42%**		**24.62%**

Table 4. Impact of Architecture on Timing

Circuit	*CLB=2*	*CLB=4*	*CLB=6*	*CLB=8*	*CLB=10*
ex5p	8.69%	23.80%	23.04%	17.09%	13.71%
apex4	5.82%	14.49%	16.79%	24.06%	23.25%
misex3	11.19%	32.47%	34.07%	30.73%	14.08%
Tseng	1.71%	10.65%	0.16%	22.32%	20.69%
alu4	22.95%	25.07%	28.02%	15.56%	20.23%
dsip	5.36%	-4.49%	13.35%	-6.06%	7.40%
seq	18.38%	19.51%	35.20%	27.04%	27.26%
diffeq	4.93%	16.01%	12.58%	30.15%	6.21%
apex2	11.51%	19.34%	25.09%	16.86%	18.39%
s298	-4.20%	7.88%	1.10%	-6.83%	4.40%
des	5.82%	28.18%	7.48%	38.16%	17.10%
bigkey	11.87%	0.03%	-7.81%	27.89%	0.61%
spla	40.52%	24.35%	39.41%	28.42%	40.85%
elliptic	31.94%	46.58%	19.21%	23.02%	33.41%
ex1010	23.26%	30.81%	28.96%	41.19%	27.86%
pdc	31.40%	67.38%	44.15%	59.63%	45.95%
frisc	6.95%	7.47%	5.21%	-4.52%	3.81%
s38584.1	23.67%	34.71%	-1.80%	-0.68%	20.86%
s38417	53.19%	53.61%	45.03%	70.61%	41.38%
clma	26.05%	34.52%	60.14%	58.87%	79.80%
Average	**17.05%**	**24.62%**	**21.47%**	**25.68%**	**23.36%**

In **Table 3**, we compare *SCPlace* with both *VPR* and *TTT* [16] in timing optimization. If we use path counting-based net weighting scheme only in *SCPlace*, we can outperform *VPR* by 14% (column 4); if we perform clustering optimization only in *SCPlace*, we can outperform *VPR* by 13% (column 6); if we integrate the path counting-based net weighting scheme with the clustering optimization, *SCPlace* significantly outperforms the original *VPR* result by 25%.

In **Table 4**, we illustrate the impact of architecture on the delay improvement obtained from *SCPlace*. For architecture with the CLB size of 2, the timing gap between *SCPlace* and *T-Vpack+VPR* is 17%. When the size of the CLB (*N)* increases from 4 to 10, the timing gap between *SCPlace* and *T-Vpack+VPR* remains in a narrow range between 22 to 25%. The result shows that even when the CLB size is relatively small (2 or 4), it is difficult to generate a good clustering solution with small delay without physical information. Since *SCPlace* explores different clustering solutions during the placement stage, it generates clustering and placement solutions with much better delay.

6.3 Runtime Speedup

Table 5. Effect of α on timing (CLB = 4)

	α=0.25		α=0.50		α=1.0	
Circuit	Timing	runtime	Timing	runtime	Timing	runtime
des	24.47%	26.15%	28.29%	38.13%	32.64%	70.28%
bigkey	-10.20%	30.42%	16.00%	42.59%	7.18%	72.58%
spla	27.09%	41.17%	34.99%	49.39%	28.57%	76.59%
elliptic	51.06%	42.89%	48.61%	51.12%	49.63%	73.86%
ex1010	29.69%	36.08%	31.24%	41.53%	31.73%	64.08%
pdc	58.75%	32.75%	69.24%	39.72%	86.41%	58.41%
frisc	0.66%	33.61%	-0.72%	40.23%	7.33%	60.61%
s38584.1	43.58%	26.62%	47.86%	32.53%	34.81%	47.41%
s38417	27.05%	32.01%	53.17%	37.47%	60.67%	59.77%
clma	27.80%	25.82%	38.38%	31.75%	41.08%	48.29%
Average	**21.59%**	**32.75%**	**31.18%**	**40.45%**	**30.83%**	**63.19%**

Table 6. Effect of α on timing (CLB = 10)

	α=0.25		α=0.50		α=1.0	
Circuit	Timing	runtime	Timing	runtime	Timing	runtime
des	16.59%	24.53%	19.22%	36.35%	20.88%	71.59%
bigkey	-4.34%	36.24%	-9.78%	54.49%	2.95%	98.44%
spla	33.13%	61.99%	43.21%	74.56%	40.32%	121.37%
elliptic	50.30%	45.57%	54.03%	47.84%	52.97%	72.16%
ex1010	38.80%	54.47%	26.00%	61.75%	32.12%	98.68%
pdc	49.99%	50.79%	52.58%	60.38%	54.95%	95.26%
frisc	-2.29%	46.78%	10.42%	57.29%	14.86%	88.07%
s38584.1	30.30%	31.73%	39.66%	33.08%	39.01%	52.39%
s38417	22.18%	43.91%	41.60%	50.76%	40.63%	81.65%
clma	54.89%	28.85%	71.51%	36.69%	83.96%	58.08%
Average	**20.18%**	**42.49%**	**27.05%**	**51.32%**	**31.25%**	**83.77%**

Table 7. Routed Delay and Track Count Comparison

	VPR		*SCPlace*		%	
Circuit	Routed delay	#tracks	Routed delay	#tracks	Routed delay	#tracks
ex5p	52.38	646	45.47	627	15.20%	3.03%
apex4	55.93	627	46.32	665	20.75%	-5.71%
misex3	56.56	588	40.92	588	38.22%	0.00%
tseng	41.08	483	36.35	437	13.01%	10.53%
alu4	55.16	594	47.47	506	16.20%	17.39%
dsip	38.80	935	35.73	660	8.59%	41.67%
seq	58.13	744	49.12	768	18.34%	-3.13%
diffeq	50.41	506	39.57	506	27.39%	0.00%
apex2	58.00	775	48.03	725	20.76%	6.90%
s298	103.69	648	89.43	621	15.95%	4.35%
des	88.32	960	69.26	832	27.52%	15.38%
bigkey	42.60	495	48.40	550	-11.98%	-10.00%
spla	78.65	1452	67.68	1287	16.21%	12.82%
elliptic	75.16	1156	62.14	1054	20.95%	9.68%
ex1010	102.88	1188	81.58	1008	26.11%	17.86%
pdc	125.46	2028	93.18	1755	34.64%	15.56%
frisc	87.64	1560	127.02	1600	-31.00%	-2.50%
s38584.1	66.41	1276	47.02	924	41.24%	38.10%
s38417	81.54	1410	54.15	1128	50.58%	25.00%
clma	144.02	2760	124.14	2040	16.01%	35.29%
Average					**19.23%**	**11.61%**

For a given architecture, each CLB contains *N* BLEs, *I* inputs and *N* outputs. In the input clustered netlist, the number of CLBs is *n*, and the number of BLEs is *m*, and $m \approx N*n$. From **Table 1** to **Table 4**, we perform $m^{1.33} \approx (N*n)^{1.33}$ fragment moves and 0 block moves. In this section, we fix the number of block moves to be $n^{1.33}$, and set the number of fragment moves to be $(\alpha*m)^{1.33} \approx (\alpha*N*n)^{1.33}$, where α is between 0 and 1.

In **Table 5**, we show the impact of α on the amount of timing improvement achievable. It is no surprise that when α increases, i.e., the number of fragment moves increase, the timing improvement increases from 22% to 31%. And this is better than the 25% we achieve in **Table 4** when we perform fragment moves only. The results illustrate that performing both block and fragment moves is better than only performing one type of moves. Our runtime is generally shorter than *VPR* due to the fact that the number of block moves we perform is only 10% of *VPR*'s. If we reduce the number of block moves *VPR* performs to be the same as *SCPlace*, it yields about 5% worse result (both timing and wirelength) and consumes 15% of standard *VPR* 's runtime. When α = 0.25*, SCPlace* uses 33% of standard *VPR* 's runtime. *SCPlace*'s runtime increases up to 63% as α increases to 1. **Table 6** shows the same trend when the

size of the CLB is 10. The bottom line is that you could easily tradeoff runtime with quality by changing the value of α.

6.4 Routed Results

In **Table 7,** we show the comparison of routed delay and track count between *SCPlace* and *T-Vpack+VPR*. The given architecture has a CLB size of 4, and the *SCPlace* run is from **Table 5** when $\alpha = 0.50$. The routed delay improvement is 19% on average and the reduction in routed tracks is 12% on average. This is consistent with the estimated delay/wirelength reduction after placement.

7 Conclusions

We introduce a novel simultaneous clustering and placement algorithm and incorporate a novel path counting-based net weighting scheme. The new algorithm produces impressive results for both bounding box wire length optimization and timing optimization. When compared with the state-of-the-art separate FPGA design flow *T-VPack + VPR*, our algorithm improves up to 36% in wirelength and 31% in longest path delay. Since our algorithm has a similar computational complexity, our approach is also very scalable.

Acknowledgement

This research is partially supported by NSF Grant CCF-0096383.

References

[1] V. Betz and J. Rose, “VPR: A New Packing, Placement and Routing Tool for FPGA Research,” *International Workshop on Field Programmable Logic and Application*, pp. 213-222, 1997

[2] V. Betz, “Architecture and CAD for Speed and Area Optimization of FPGAs,” *Ph. D. Dissertation*, *University of Toronto*, 1998

[3] V. Betz, J. Rose and A. Marquardt, “Architecture and CAD for Deep-Submicron FPGAs,” *Kluwer Academic Publishers*, February 1999

[4] E. Bozorgzadeh, S. Ogrenci and M. Sarrafzadeh, "Routability-Driven Packing for Cluster-Based FPGAs," *ASPDAC*, Yokohama, Japan, 2001

[5] Andrew E. Caldwell, Andrew B. Kahng and Igor L. Markov, “Can Recursive Bisection Alone Produce Routable Placements?” *ACM/IEEE Design Automation Conference*, pp. 477–482, 2000

[6] T. Chan, J. Cong, T. Kong and J. Shinnerl, "Multilevel Optimization for Large-scale Circuit Placement," *Proc. IEEE International Conference on Computer Aided Design*, San Jose, California, pp. 171-176, November 2000

[7] C.-C. Chang, J. Cong, D. Pan, and X. Yuan, "Multilevel Global Placement with Congestion Control," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 22, no. 4, pp. 395-409, July 2002

[8] C. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 690-695, 1994

[9] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", *IEEE Trans. on Computer-Aided Design*, vol. 13, no. 1, pp. 1-12, January 1994

[10] J. Cong, H. Li and C. Wu "Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization," *Proc. 36th ACM/IEEE Design Automation Conf.*, New Orleans, Louisiana, pp. 460-465, June 1999

[11] A. Dunlop and B. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4:92-98, January 1985

[12] H. Eisenmann and F.M. Johannes, "Generic Global Placement and Floorplanning," *ACM/IEEE Design Automation Conference*, pp. 269-274, 1998

[13] S-W Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement," *IEEE/ACM International Conference on Computer-Aided Design*, pp 165-170, 2000

[14] S.S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing," *Science*, pp. 671-680, May 13, 1983

[15] J.M. Kleinhans, G. Sigl, F.M. Johannes and K.J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10:356-365, 1991

[16] T. Kong, "A Novel Net Weighting Algorithm for Timing-driven Placement," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 172-176, 2002

[17] A. Marquardt, V. Betz and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, pp. 37-46, 1999

[18] A. Marquardt, V. Betz and J. Rose, ``Timing-Driven Placement for FPGAs," *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, pp. 203 – 213, February 2000

[19] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," *Electronics Research Laboratory*, Memorandum No. UCB/ERL M92/41, 1992

[20] G. Sigl, K. Doll and F.M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?" *ACM/IEEE Design Automation Conference*, pp. 427-432, 1991

[21] W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits," *ACM/IEEE Design Automation Conference*, pp. 211-215, 1995

[22] M. Wang, X. Yang and M. Sarrafzadeh, "Dragon2000: Standard-Cell Placement Tool For Large Industry Circuits," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 260-263, 2000

[23] K. Zhong and S. Dutt, "Effective Partitioning-Driven Placement with Simultaneous Level Processing and Global Net Views," *Proc. IEEE International Conference on Computer Aided Design*, San Jose, California, pp. 254-259, November 2000

[24] http://www.eecg.toronton.edu/~vaughn/challenge/challenge.html, "The FPGA Place-and-Route Challenge"