Hierarchical Force-Based Block Spreading for Analytical FPGA Placement

Dries Vercruyce, Elias Vansteenkiste and Dirk Stroobandt
Department of Electronics and Information Systems
Computer Systems Lab, Ghent University
Ghent, Belgium

***Abstract*—Enabling efficient FPGA application development requires fast design compilation to high quality FPGA configurations. Placement and routing are the most challenging compilation steps. A promising trend to accelerate the placement problem is the use of iterative analytical placement techniques. Each iteration in these placers consists of an optimization and a legalization phase. The legalization phase is an important part of the technique, reducing block overlap in the optimized placements. In this work we propose a hierarchical pushing force-based block spreader that improves the legalization of CLBs. Overlap is reduced as the blocks push each other away under the influence of gravity, like a drop of water flowing over a surface. The algorithm is designed with parallelism in mind, making it suitable for GPU acceleration. The regularity of FPGAs is exploited to accelerate the calculations. Moreover, it further builds on a hierarchical FPGA CAD tool flow to enhance quality of results and runtime scalability. The spreading legalizer is embedded in the LIQUID placement technique. It replaces the default partitioning-based legalizer for sparse designs. The result is a reduction of 7.4% in post-routing total wire-length, without giving in on runtime. The source code is publicly available in the FPGA CAD framework on GitHub.**

I. INTRODUCTION

An FPGA configuration should be of high quality; mainly performance and compactness are important because it defines the potential of the application and the cost. Also, the compilation should run fast to enable efficient FPGA application development. In FPGA compilation, placement and routing are the most challenging problems [1]. For instance, placing one of the large TITAN23 designs [2] can take more than an hour with VPR [3]. A promising trend to accelerate the placement problem is the use of analytical techniques. An overview of existing analytical placers can be found in [5]. In this work we focus on iterative analytical FPGA placers, such as HEAP [4] and LIQUID [5]. Each iteration in these placers consists of an optimization and a legalization phase. The optimization phase reduces the placement cost by solving the problem as a linear system of equations. This phase, however, only minimizes the cost; it does not enforce a legal solution as many blocks in the optimized solution might overlap. This overlap issue is solved by spreading the optimized placement with a fast look-ahead legalizer. The block locations in the legalized placement then act as anchor points in the next iteration. A pseudo connection is added between each block and its current legal position. The weight of these pseudo connections grows through the iterations, thereby gradually spreading the placement.

Dries Vercruyce is supported by a PhD grant of the Research Foundation Flanders (FWO).

This work is supported by the European Commission in the context of the H2020-FETHPC EXTRA project (no. 671653).

Legalization [6]–[8] is an important part of the iterative analytical placement technique as it reduces block overlap in the optimized placements. Its quality should be high in order to optimally maintain the result of the optimized solution. FPGA legalization does, however, impose challenges because it is a heterogeneous assignment problem. FPGAs usually have a column-based architecture in which each column supports a certain type of blocks. A majority of the columns support Configurable Logic Blocks (CLB). The remaining columns embed optimized hard blocks for common functions, such as DSPs and BRAMs. Different types of blocks thus have to be placed on grid locations that support their functionality. This heterogeneity problem is solved by legalizing the block types one by one. HEAP uses a top-down recursive partitioning technique for all block types separately [4]. LIQUID improves the legalization of the optimized hard blocks with a high-quality annealing-based technique [5], whereas the default partitioning-based HEAP legalizer is used for the CLBs, as the annealing-based technique is not suitable for CLB legalization.

This work improves CLB legalization with a hierarchical pushing force-based block spreader that tries to maximally retain the quality of the optimized placements. A gradual spreading technique is used in which the blocks move according to the local block overlap in a global mass map, like the diffusion-based ASIC legalizer of Ren *et al.* [8]. Based on a drop of water flowing over a surface, the blocks are spread under the influence of gravitational forces. They push underlying blocks away as their mass is attracted by gravity. The regularity of FPGAs is exploited to accelerate the calculations. In addition, we further build on the hierarchical MULTIPART packer [9] to improve quality of results and runtime scalability. The algorithm is designed with parallelism in mind, making it suitable for GPU acceleration. The spreading-based legalizer is embedded in LIQUID and replaces the default partitioning-based legalizer for sparse designs. It reduces the post-routing total wire-length by 7.4%, without giving in on runtime.

The remaining part of the paper is organized as follows. The fundamentals of the pushing force-based spreading technique are explained in Section II. Section III contains a description of the hierarchical approach. Experimental results are presented in Section IV and Section V contains the conclusion.

II. Pushing Force-Based Spreading Technique

The pushing force-based legalizer uses a gradual spreading technique in which the blocks push each other away under the influence of gravity. In small steps, the blocks drift away from high-mass regions. Like a liquid, they flow over the area of the FPGA. Substantial block overlap can occur in the optimized placements, which makes the spreading problem challenging. Luckily, it is not required to exactly assign the blocks to a legal FPGA location in the optimize-legalize iterations as the increasing weight of the pseudo connections enforces that the placement gradually spreads. A final detailed legalization step is performed with the Abacus legalizer [6] on the placement that results from the best optimize-legalize iteration.

The gradual spreading technique is based on the gravitational forces in a global mass map. The concept of the global mass map is first explained below. We then describe how the calculation of the forces is accelerated by exploiting the regularity of FPGAs. Finally, we clarify how the spreading algorithm works.

A. Global Overlap Overview: Mass Map

The state of the system is stored in a global mass map, it contains the mass of the blocks on each position of the FPGA. The more mass a certain position has, the larger the corresponding spreading force will be. The spreading force that acts on a block is hence based on the pushing mass of the overlapping blocks. It is not feasible to exactly calculate the mass of each FPGA position. The CLB locations are therefore subdivided in equally sized bins that contain a certain mass. For all blocks we calculate the overlapping area with each of the rectangular bins. The size of these bins determines the accuracy of the spreading forces. Smaller bins better represent the exact mass on each location, at the cost of a larger runtime and memory consumption. The best runtime-quality tradeoff is achieved if each CLB location is subdivided into four rectangular bins (Fig. 1). The mass of a CLB is set equal to 1, homogeneously distributed over its area. Each bin can thus maximally contain a mass of 0.25 in an overlap-free placement, as each CLB overlaps with four bins.

B. Calculation of the Spreading Force

The spreading force of a block is calculated for the horizontal and vertical dimension separately. Similar to the mass map, we subdivide the block area into four equal sized sections (Fig. 1). The force in a section is based on the mass on top that is pushing it away, thereby spreading the mass in the mass map by moving the blocks away from congested regions. With the force in the sections we can easily calculate the horizontal and vertical force of a block. The force in the horizontal direction is the difference between the force to the west and the force to the east (1). The vertical force is the difference between the northern and southern force (2).

$$F_{hor} = F_{sw} + F_{nw} - F_{se} - F_{ne} \tag{1}$$
$$F_{ver} = F_{sw} - F_{nw} + F_{se} - F_{ne} \tag{2}$$

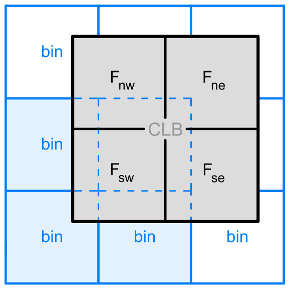


Fig. 1. The grid of bins (blue) and the four force sections of a block (gray).

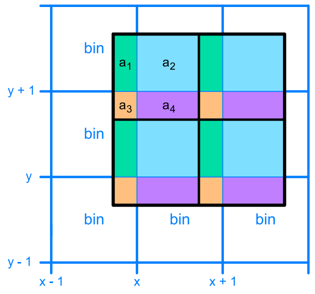


Fig. 2. Subdivision of the force sections into areas ($a_1 \rightarrow a_4$) snapped to the grid of bins.

The force in each of the sections is quickly calculated by further subdividing the sections into smaller areas ($a_1 \rightarrow a_4$), snapped to the grid of bins (Fig. 2). Each of the subsections overlaps with only one bin in the mass map. Due to the regular structure of an FPGA, the same areas are obtained in the four force sections of a block (Fig. 2). In each subsection we multiply the overlapping area with the mass of the corresponding bin in the mass map. The larger the overlap with a bin, the more influence the mass at that position has. The four products are added up and result in the force of a section. The equations for the four force sections are given in (3) - (6), relative to the coordinate system shown in Fig. 2, with $m_{x,y}$ the mass in the bin on position (x, y).

$$F_{nw} = a_1 m_{x-1,y+1} + a_2 m_{x,y+1} + a_3 m_{x-1,y} + a_4 m_{x,y} \tag{3}$$
$$F_{sw} = a_1 m_{x-1,y} + a_2 m_{x,y} + a_3 m_{x-1,y-1} + a_4 m_{x,y-1} \tag{4}$$
$$F_{ne} = a_1 m_{x,y+1} + a_2 m_{x+1,y+1} + a_3 m_{x,y} + a_4 m_{x+1,y} \tag{5}$$
$$F_{se} = a_1 m_{x,y} + a_2 m_{x+1,y} + a_3 m_{x,y-1} + a_4 m_{x+1,y-1} \tag{6}$$

C. Spreading Methodology

In an iterative way, the blocks are spread according to their horizontal and vertical force. In each iteration we move all blocks based on the current state of the mass map. The mass of a block is first removed from the mass map to avoid that the block has an influence on its own spreading force. The horizontal and vertical force is then calculated. The actual movement of the blocks is the product of the spreading force and the step size, which parameterizes the movement of the blocks in the direction of the force. It depends on the current optimize-legalize iteration number and the size of the FPGA. When a block is moved, the overlapping areas with the bins ($a_1 \rightarrow a_4$) are updated. Finally, the mass of the block at its new position is added to the mass map.

The spreading methodology is designed with parallelization in mind. The force of the blocks can be calculated in parallel, which makes it suitable for GPU acceleration. The force only depends on the overlap of the blocks, which is given in the global mass map. The synchronization between the parallel moves thus occurs through this mass map, which will not always be up-to-date as multiple blocks move at the same time. Therefore we should select blocks that are not in each other's neighborhood to move in parallel. This is usually feasible due to the large number of movable blocks.

III. Hierarchical Cluster Spreading

The spreading technique discussed above has poor scalability with an increasing design size because larger designs have more blocks to be spread over a larger area. As a solution we propose a hierarchical approach with a fast cluster-spreading phase and a fine-grained refinement phase. It further builds on the hierarchical FPGA CAD tool flow, introduced by the MultiPart packer [9]. MultiPart partitions the circuit into a set of weakly interconnected clusters that are packed independently. During placement we observe that the placement cost is minimized if the blocks in a cluster are placed close to each other, as shown in Fig. 3, where the blocks in each cluster have their own color. Using this information entails two advantages. Firstly, it improves quality because the clusters provide information about the natural design hierarchy. Secondly, a faster and more scalable runtime is achieved because the clusters are spread independently and relative to each other. The hierarchical approach consists of three spreading techniques:

1) Intra-cluster spreading: the overlap in each cluster is spread separately with an independent mass map (Fig. 4b). The blocks are only pushed by other blocks in the cluster. Intra-cluster spreading is fast due to a large problem size reduction.

2) Inter-cluster spreading: the clusters are moved relative to each other (Fig. 4c). The relative position of the blocks inside a cluster remains unchanged as they move as rigid objects. First we calculate the force of each block separately. The force that acts on the cluster is the combined force of all blocks.

3) Block spreading: after the hierarchical cluster spreading, we further refine with block spreading. Block spreading is the default spreading technique discussed in the previous section. It is slower than cluster spreading but results in a more fine-grained solution. The starting point of block spreading is shown in Fig. 4d and leads to the spread placement in Fig. 4e.

The hierarchical approach consists of a cyclic cluster-spreading phase and a fine-grained refinement phase. Starting from the optimized solution (Fig. 4a), the placement is first spread based on the clusters. In a cyclic way, they are spread independently (intra-cluster) and relative to each other (inter-cluster). Intra-cluster spreading increases the overlap between the clusters as their region expands (Fig. 4b). Cluster overlap is then reduced with inter-cluster spreading by moving the overlapping clusters (Fig. 4c). The alternating intra-cluster and inter-cluster spreading phase is repeated until enough overlap is reduced. Overlap is defined as the sum of the excess mass in each bin, relative to an overlap-free placement. The criterion to switch from hierarchical cluster spreading to fine-grained block spreading states that the average fraction of overlapping area of a block should be lower than α. The partially spread solution (Fig. 4d) is then further refined with block spreading (Fig. 4e). This is required to spread the remaining overlap and to close the gaps between the clusters (Fig. 4d). These gaps arise because the clusters are spread independently and are moved as rigid objects. The parameter α sets a trade-off between the fast cluster-based spreading and the slow fine-grained block spreading. Its optimal value is equal to 0.2.

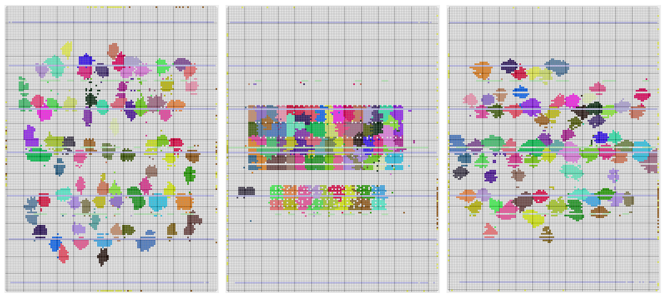
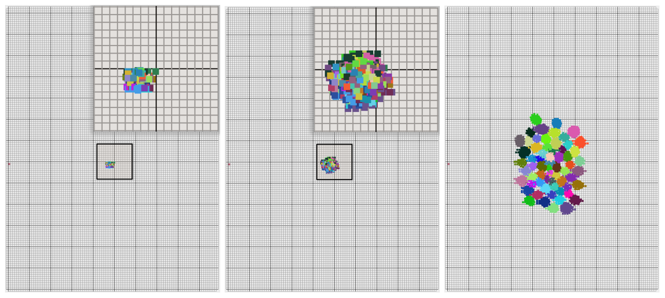

(a) simulated annealing (b) partitioning-based (c) spreading-based

Fig. 3. Final placement of *des90* for (a) annealing-based placement and analytical placement with (b) partitioning-based legalization and (c) hierarchical spreading force-based legalization.


(a) Optimized Placement (b) Intra-cluster (c) Inter-cluster

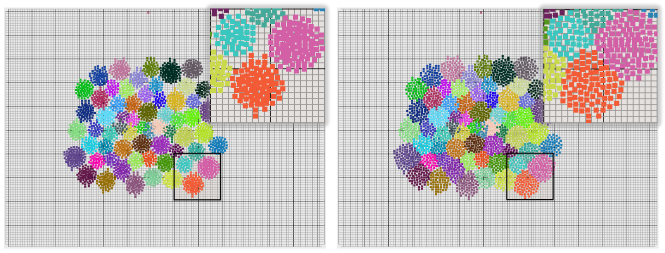

(d) Result of cluster spreading (e) Result of block spreading

Fig. 4. Hierarchical legalization: intra-cluster, inter-cluster, and independent block spreading. The blocks in each cluster have their own color.

IV. Experiments

The spreading legalizer is analyzed by embedding it in the Liquid placer (Liquid$_{\text{HiS}}$). It replaces the default partitioning-based legalizer (Liquid$_{\text{P}}$ [5]) for the CLBs. The high-quality annealing-based legalizer [5] is used for the hard blocks in both versions of Liquid. The Titan23 designs [2] are used for benchmarking. They are clustered with MultiPart. A variance in results is produced by MultiPart (hMetis partitioning) and VPR (placement seed). Therefore we compile the designs 10 times and report the average over these iterations. The target device is a model of Altera's Stratix IV FPGA [2]. The FPGA dimensions are sized for each design separately. A few Titan23 designs are omitted: routing congestion problems arise for *bitcoin_miner* with the default channel width of 300, an error occurs for *LU_network* and *gaussianblur* in VPR 7.0.7, and *LU230* fails during packing with MultiPart. Experiments are performed on a workstation with an Intel E5-2660v3@2.6GHz. All algorithms have a single-threaded implementation for a fair runtime comparison.

TABLE I
CLB USAGE (CLB$_{\%}$), POST-ROUTING CRITICAL PATH DELAY (CPD) AND TOTAL WIRE-LENGTH (TWL) OF LIQUID$_{P}$ COMPARED TO VPR.

Dense designs			
Circuit	CLB$_{\%}$	CPD	TWL
sparcT1_core	92	0.96	1.03
SLAM_spheric	89	1.01	0.93
segmentation	88	1.02	0.95
stap_qrd	90	1.08	0.94
sparcT2_core	95	0.84	0.99
denoise	88	1.01	0.90
mes_noc	92	1.02	1.05
directrf	93	1.19	0.83
geomean	**91**	**1.01**	**0.95**

Sparse designs			
Circuit	CLB$_{\%}$	CPD	TWL
neuron	28	0.90	1.06
stereo_vision	25	0.94	0.98
cholesky_mc	48	0.98	1.08
des90	21	1.05	1.10
bitonic_mesh	17	1.11	1.16
dart	51	0.92	0.97
openCV	29	1.09	0.95
minres	26	0.95	0.99
cholesky_bdti	47	1.08	1.04
gsm_switch	45	1.11	1.20
sparcT1_chip2	59	0.98	1.00
geomean	**33**	**1.01**	**1.05**

First we compare the quality of LIQUID$_{P}$ with the placer in VPR if MULTIPART is used for packing (Table I). Earlier work [5] showed that VPR and LIQUID$_{P}$ achieve similar quality of results if VPR is used for packing. However, the performance of LIQUID$_{P}$ depends on the CLB usage percentage (CLB$_{\%}$) if the packing approach is replaced by MULTIPART. Two types of designs are analyzed: sparse and dense designs. Sparse designs have a low CLB usage percentage due to the high number of required hard blocks. This results in a large FPGA because the hard block columns are sparse. Dense designs perform well with the default partitioning-based technique. A small loss of 1% is observed for the post-routing critical path delay (CPD) while the post-routing total wire-length (TWL) gains 5%. The increase in CPD is mainly caused by the large *directrf* design. A problem arises for the sparse designs. Both the CPD and TWL give in on quality with a loss of 1% and 5% respectively. The loss can be explained by examining Fig. 3. The partitioning-based legalizer enforces the CLBs into rectangular regions (Fig. 3b), whereas the annealing-based approach (Fig. 3a) better uses the available resources to optimize the placement cost. It results in a better placement of the independent CLB clusters and places them closely to their highly interconnected hard blocks in the discrete hard block columns that are scattered over the FPGA.

The spreading-based legalizer solves the wire-length gap as it better preserves the optimized placements. The blocks can flow freely over the chip, the only limitation being the edge of the FPGA. Table II contains the results of LIQUID$_{HiS}$ and the relative performance of VPR and LIQUID$_{P}$. On average, both legalizers result in similar runtime, with LIQUID$_{HiS}$ slightly outperforming LIQUID$_{P}$ by 1.08x. Although the spreading technique requires more time in one iteration, it better preserves the quality of the optimized placements, leading to a faster convergence. As a result, we could reduce the number of optimize-legalize iterations from 40 to 27. The CPD is slightly better with the spreading-based legalizer, which closes the gap between LIQUID and VPR. However, this gap is small and is susceptible to the variance of results produced by MULTIPART and VPR. A clear improvement is achieved for the TWL. The spreading technique reduces the TWL with 7.4%, leading to 3.1% lower wire-lengths when compared to VPR.

TABLE II
RUNTIME (RT), POST-ROUTING CRITICAL PATH DELAY (CPD) AND TOTAL WIRE-LENGTH (TWL) OF LIQUID$_{HiS}$ AND RELATIVE PERFORMANCE OF THE BASELINE PLACERS COMPARED TO LIQUID$_{HiS}$.

	VPR [3]			Liquid$_{P}$ [5]			Liquid$_{HiS}$		
Circuit	RT rel	CPD rel	TWL rel	RT rel	CPD rel	TWL rel	RT [s]	CPD [ns]	TWL [M]
neuron	9.3	1.09	1.06	1.00	0.98	1.12	15	8.8	0.77
stereo_vision	10.7	1.04	1.02	1.03	0.98	1.00	15	8.1	0.68
cholesky_mc	9.7	1.01	0.99	0.98	0.99	1.07	25	8.3	1.19
des90	10.6	1.04	1.06	1.11	1.10	1.17	36	11.0	2.36
bitonic_mesh	12.8	0.97	1.05	1.23	1.07	1.22	63	14.4	4.68
dart	16.0	1.08	1.07	1.04	0.99	1.04	36	14.8	2.09
openCV	15.0	0.83	1.19	1.21	0.90	1.14	42	13.3	3.09
minres	15.4	1.04	1.08	1.10	0.98	1.07	42	9.4	2.69
cholesky_bdti	17.9	0.99	1.03	1.12	1.07	1.07	46	9.8	2.49
gsm_switch	26.0	0.92	0.88	1.04	1.02	1.06	79	13.0	6.20
sparcT1_chip2	37.0	1.02	0.95	1.04	0.99	0.94	109	23.1	7.43
geomean ratio	**15.0**	**1.00**	**1.03**	**1.08**	**1.01**	**1.08**	**1.00**	**1.00**	**1.00**

V. CONCLUSION

In this work we enhance the CLB legalization in iterative analytical FPGA placement techniques with a hierarchical pushing force-based block spreader. A gravity-inspired mass-flow approach is used in which the blocks move based on the local congestion in a global mass map. Runtime scalability is improved with a hierarchical approach. The spreader better preserves the quality of the optimized placements, leading to a faster convergence and a higher quality of results. An improvement of 7.4% in post-routing total wire-length is achieved, without giving in on runtime. The source code is publicly available in the FPGA CAD Framework on GitHub [10].

REFERENCES

[1] R. Aggarwal, "FPGA place & route challenges," in *Proc. of the 2014 on int. symp. on physical design*. ACM, 2014, pp. 45–46.

[2] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven Titan: Enabling large benchmarks and exploring the gap between academic and commercial CAD," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, p. 10, 2015.

[3] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[4] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 143–150.

[5] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, "Liquid: High quality scalable placement for large heterogeneous FPGAs," in *Field Programmable Technology (ICFPT), 2017 International Conference on*. IEEE, 2017, pp. 17–24.

[6] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: fast legalization of standard cell circuits with minimal movement," in *Proc. of the 2008 int. symp. on Physical design*. ACM, 2008, pp. 47–53.

[7] U. Brenner and J. Vygen, "Legalizing a placement with minimum total movement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 12, pp. 1597–1613, 2004.

[8] H. Ren, D. Z. Pan, 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.

[9] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, "How preserving circuit design hierarchy during FPGA packing leads to better performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[10] "FPGA CAD Framework: MultiPart and Liquid," https://github.com/EliasVansteenkiste/FPGA-CAD-Framework, 2017.