Liquid: Fast Placement Prototyping Through Steepest Gradient Descent Movement

Elias Vansteenkiste, Seppe Lenders and Dirk Stroobandt
Hardware and Embedded Systems Group
Computer Systems Lab
Department of Electronics and Information Systems
Ghent University, Belgium
Email: {Elias.Vansteenkiste, Seppe.Lenders, Dirk.Stroobandt}@university.edu

***Abstract*—FPGA design compilation takes too much time to allow efficient design turnaround times. The largest runtime consuming steps of the compilation are placement and routing. To speed up the FPGA placement process, analytical placement techniques have become more popular in the last decade . Analytical techniques produce a placement in two steps, a placement prototyping step and a refinement step. In this work we focus on fast FPGA placement prototyping. Placement prototypes are also used to obtain fast accurate timing estimations and speed up the design cycle. In conventional analytical placement prototyping techniques the placement problem is formulated as a linear system which is solved several times to find a good legal placement. The most time consuming step of that process is solving the linear system. We show that it is not necessary to exactly solve this system, but that it is sufficient to optimize the placement following the steepest gradient descent in between legalization phases. This technique is implemented in our new placement tool called LIQUID. In LIQUID each block's position is updated several times following an accelerated gradient simulation. We compare this new technique with conventional analytical placement. The Titan23 designs and the Stratix IV FPGA are used for benchmarking. The net effect is that the runtime can be reduced by 2x on average compared to conventional analytical placement without losing any quality.**

I. INTRODUCTION

The most time-consuming steps of the FPGA compilation are the placement and routing steps. Murray et al. report a placement runtime of 7 hours for the one million block sized *bitcoin_miner* design, which is part of the Titan23 benchmark suite [1]. The increasing problem size of FPGA placement worsens. Shannon et al. observed an increase in both the size of applications and the size of the target devices following Moore's law [2]. The increase in size makes it hard to keep the placement scalable in terms of runtime and memory requirements. To overcome this problem, device manufacturers and academics developed multi-threaded versions of the previously serial implemented algorithms [3], [4]. Although seemingly successful the algorithms cope with problems such as serial equivalence, being deterministic and scalability in terms of number of threads.

Since the introduction of high-level synthesis and the emergence of new markets, more and more engineers with a software background attempt to accelerate applications with an FPGA. They are used to gcc-like compilation times and their design methodologies are adapted to these short compilation times. They cannot accept the long compilation times that are common in FPGA design. To help the design process a placement prototype suffices to estimate the post-routing performance of the design, because the post-placement quality of results estimation correlates much stronger with the post-routing performance [5].

In this work we present a new way to calculate placement prototypes and compare it to conventional analytical placement. The new placement prototyping algorithm is called LIQUID and is based on the same principle used in the SIMPL[6] and HEAP[7]. In each main iteration non-legal optimized placements and a legalised version of the optimized placement are generated consecutively. In conventional analytical placement tools the problem is described as a system of equations which is derived to find the optimal location for each block. However, the solution generated by solving the system is then partly destroyed while legalising the placement. In this paper we show that the linear system does not have to be solved exactly. In LIQUID all the blocks are moved in the direction which minimises the cost the most (the steepest gradient descent). For each block a move vector is calculated based on the previous position of the blocks that are attached to it. The cost function depends on the estimated routing and timing cost of the nets attached to the block. Calculating the move vectors is computationally less intensive than solving the linear system, which makes LIQUID much faster than classic analytical placement. All movement vectors can be calculated independently, which makes it uniquely suitable for parallelisation. In this work we present a single threaded implementation as a proof-of-concept. Test results are already showing great speedups for a single threaded version. Our placement tools are released as an open source project [8]. In this way we hope we can combine the academic research efforts done on analytical placement, because although some previous work about analytical placement on FPGAs is published, none of these publications have resulted in a public release of source code or binaries.

II. LIQUID

A. The Basic Algorithm

LIQUID is based on the same cyclic approach used in the analytical placers in [6] and [7]. Each iteration consists of an

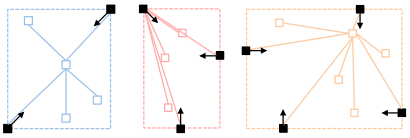

Fig. 1. The effect a net has on the blocks it connects / on its terminals.

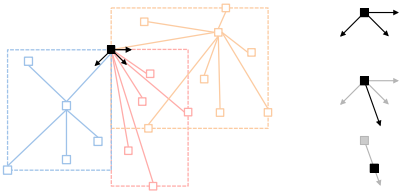

Fig. 2. The aggregated effect of all the nets attached to a block on the steepest gradient descent direction

optimization phase and a legalization phase.

- In the optimization phase the placement is optimised neglecting the legal positions on the device. In contrast to analytical placement there is no linear system that is solved in each iteration. In each iteration all the blocks are moved several times in the direction of the steepest gradient descent of the cost function. The net effect is that the placement cost is reduced but the position of the blocks in the placement is not legal anymore.
- In the legalized phase the solution is legalized following a similar method described in [7]. After legalising the placement, the timing graph is traversed to update the weights of the critical connections.

B. Inner Optimization loop

In the optimization phase all the blocks are moved several times to greedily reduce the cost of the placement. All blocks are considered to be moved in the direction of the steepest gradient descent. As an analogy we consider a particle with $\mathbf{v} = 0$ at some random location in a hilly landscape. Gravity is a conservative force so the particle has a potential energy, $U = mgh$ and the force can be expressed as $\mathbf{F} = -\nabla\mathbf{U}$. In LIQUID we try to simulate the particle rolling down to the bottom. In several steps we update the speed and the location of the blocks. If the blocks are given a certain mass, they can built up momentum and escape local minima. The new location and speed of the block, $\mathbf{x}_i$ and $\mathbf{v}_i$ are updated following Nesterov's momentum update. Nesterov's Accelerated Gradient Descent is an optimal method for smooth convex optimization [9]. The speed of each block at the beginning of the optimization phase is $\mathbf{v}_0 = 0$. The start location of the blocks, $\mathbf{x}_0$, is determined by the previous legalization procedure (or random in the first iteration). First the x-location is updated, $\mathbf{x}'$, with the momentum the block still has from the previous speed $\mathbf{v}_{i-1}$:

$$\mathbf{x}' = \mathbf{x}_{i-1} + \mu \cdot \mathbf{v}_{i-1} \tag{1}$$

Subsequently the velocity is integrated:

$$\mathbf{v}_i = \mu \cdot \mathbf{v}_{i-1} - \gamma \cdot \nabla\mathbf{C}(\mathbf{x}') \tag{2}$$

with $\nabla\mathbf{C}(\mathbf{x}')$ the gradient of the cost function for position $\mathbf{x}'$ and μ the coefficient of friction. Friction dampens the velocity and reduces the kinetic energy of the system, otherwise the block would never come at a stop. γ is the gradient sensitivity rate and can be compared to the product of the mass of the particle and the gravitational constant. μ and γ are parameters of the algorithm that mainly affect the quality of the placement. We obtained the best quality for $\mu = 0.2$ and $\gamma = 0.32$. With the new speed the new position is found by integrating:

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{v}_i \tag{3}$$

Let's assume that our placement tool is wirelength-driven then the cost function $\mathbf{C}(\mathbf{x})$ is the total sum of the HPWL of each net in the design.

$$C(\mathbf{x}) = \sum_{n \in nets} q(n) \cdot HPWL(\mathbf{x}, n) \tag{4}$$

A block only has effect on the nets attached to it, so the total sum is not calculated to find $\nabla\mathbf{C}(\mathbf{x})$ for one block. Only the nets attached to the block are under consideration. This is an important insight that allows us to consider each block separately. Additionally only the blocks on the edge of the bounding box have a direct impact on the HPWL cost.

Depending on where the block is located on the edge a different vector is calculated, as can be seen in Figure 1. Blocks on the corner are pulled inwards with a normalized vector that pulls equally in the x- and y-direction and blocks on an edge that are not on a corner are pulled inwards by a vector that is perpendicular to the edge. A block typically has mutliple nets attached to it and so the different vectors are added to find $\nabla\mathbf{C}(\mathbf{x})$. In Figure 2 an example is shown. A block is attached to three nets and the resulting forces are depicted. The gradient vector for one dimension is calculated in two steps. In the first step all the weight of the connections attached to the block are added or subtracted depending on the position of the attached block. The sign of the weight is negative if the connected block has a lower x-coordinate and vice versa. The sign of the sum indicates the direction the block will move. In the second step, the magnitude of the move vector is determined based on the position of the connected blocks calculated in the previous iteration, $\mathbf{x}_{i-1}$. A connection between blocks that are farther apart will increase the effective move vector. This is a second order term that showed improvement in the quality of results.

Up until now we neglected the anchor positions in our description. In order to converge towards a good legalized placement, Equation (2) is updated to include a weighted sum of the last legal position of the block $\mathbf{x}_l$ and $\nabla\mathbf{C}$ based on the anchor weight w_ρ;

$$\mathbf{v}_i = \mu \cdot \mathbf{v}_{i-1} - \gamma \cdot (w_\rho \cdot (\mathbf{x}_l - \mathbf{x}') + (1 - w_\rho) \cdot \nabla\mathbf{C}(\mathbf{x}')) \tag{5}$$

The new positions and speeds are calculated, the old positions and speeds are discarded. Next, a new iteration is started in which new positions and speeds are calculated.

LIQUID does not stop when the optimised placement cost reaches a fraction of the legalized placement cost as is the case for conventional analytical placement. Instead we empirically devised a fixed schedule in which all parameters are inferred from the number of outer loop iterations N_{outer}, which can be used as an effort level tuning parameter. An outer loop iteration contains an optimization phase and a spreading phase. Let i be the outer loop counter then the anchor weights are calculated as follows;

$$w_\rho(i) = i \cdot \frac{w_{\rho,target}}{N_{outer} - 1} \tag{6}$$

with $w_{\rho,target}$ the final anchor weight with a default value of 0.85. The number of inner loop iterations (number of momentum updates) N_{inner} starts at N_{outer} and is linearly decreased to end at 1 in the final outer iteration.

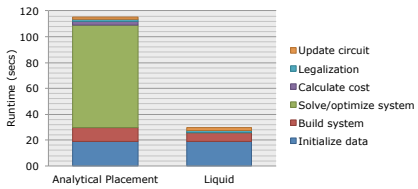

Fig. 3. The runtime breakdown for analytical placement ($\rho = 1.1, \beta = 0.8$) and LIQUID ($N_{outer} = 17$), both in wire-length driven mode to reach a placement cost of 221.6K for the *bitcoin_miner* design.

C. Timing-Driven Placement

To account for timing sensitive paths, we extra add source-sink connections to the system. Every connection corresponds with an edge in the timing graph and has a criticality. A connection from block i to block j is added to the linear system with the weight

$$w_{i,j} = \begin{cases} 0, & crit < \theta_c \\ \alpha \cdot \frac{Crit^{\epsilon_c}}{|x'_i - x'_j|}, & \text{otherwise} \end{cases} \tag{7}$$

There are three parameters that determine the weight of the source-sink connections. Source-sink connections with a criticality lower than the criticality threshold θ_c are not added to the system. The timing trade-off factor α determines the importance of the source-sink connections compared to the HPWL. And the criticality exponent enables us to put more emphasis on minimizing the lenght of the most critical connections.

D. Runtime breakdown comparison

The runtime breakdown of our analytical placer and LIQUID are charted in Fig. 3 for placing the bitcoin_miner benchmark design with the same placement quality. According to the breakdown we have succeeded in our intent to reduce the time to solve or optimize the system which is indicated in green in the bar chart. The gradient descent based approach in Liquid is with 0.1s much faster than solving the linear system with 79.1s. This leads to an overall 3.9x runtime speedup.

III. EXPERIMENTS

A. Methodology

The Titan23 designs are used for benchmarking. The results are listed in Table I. The target device for the Titan23 designs is Altera's Stratix-IV FPGA. Liquid and the conventional analytical placer are implemented in the FPGA placement framework, which is available online [8]. The framework is written in Java and the experiments are performed with OpenJDK 64-Bit Server VM v24.95 on a workstation with an Intel Core i7-3770@3.40GHz and with 16GB 1600Mhz DDR3 work memory.

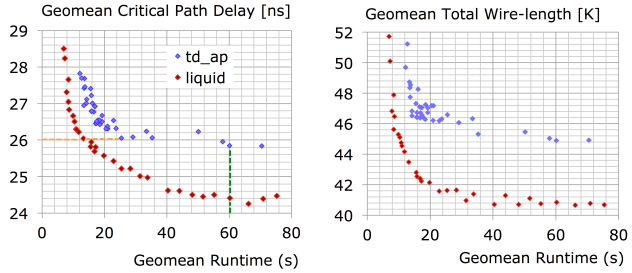

Fig. 4. Pareto front for LIQUID and analytical placement: critical path delay and wire-length versus runtime

B. Runtime versus Quality

To make a fair comparison we first give an overview of the capabilities of both LIQUID and analytical placement. For both algorithms we looked at the parameter that influenced the runtime-quality tradeoff the most. For Liquid this is N_{outer}, the number of outer iterations. N_{outer} is varied from 3 to 96. For analytical placement the most sensitive parameter is ρ, the anchor weight multiplier. ρ is varied from 1.005 to 2.4. The placement results in terms of the critical path delay are charted in Fig. 4. All metrics reported are the geometric mean of the values for for the individual benchmark designs. We can clearly observe that the placement results of LIQUID are superior to the placement results of analytical placement. All the data points on the Pareto front of Liquid have either a better runtime or a better critical path delay and wirelength cost with their respective counterparts on the Pareto front of the analytical placement results. While developing LIQUID we hypothesized that we can rely on the legalization phase to disturb the solution enough to keep the placement from crystallising and converging to a local minimum. It seems that this hypothesis is correct, because we are able to obtain better quality placement results.

C. Runtime speedup

To further investigate in detail we investigate how fast LIQUID is when trying to achieve the same quality as analytical placement. The datapoints chosen for the comparison are indicated by the orange striped line in Fig. 4. We choose the

TABLE I
RUNTIME (RT) [S], WIRE-LENGTH (WL) [K] AND CRITICAL PATH DELAY (CPD) [NS] FOR ANALYTICAL PLACEMENT (AP) AND LIQUID. THE RESULTS ARE REPORTED RELATIVE TO **LIQUID-FAST**

	Liquid - Fast			**AP - Fast**			**Liquid - HQ**			**AP - HQ**		
Setting	$N_{outer} = 20$			$\rho = 1.11$			$N_{outer} = 80$			$\rho = 1.018$		
Design	**RT**	**WL**	**CPD**	**RT**	**WL**	**CPD**	**RT**	**WL**	**CPD**	**RT**	**WL**	**CPD**
bitc∼	58	216	17	1.29	1.12	1.03	4.30	0.95	0.95	3.59	1.13	1.01
bito∼	20	67	17	1.53	1.08	1.07	3.69	1.01	1.05	3.35	1.11	1.06
c_bdti	11	33	11	1.61	1.32	0.95	5.26	0.91	1.03	2.48	1.33	0.99
c_mc	5.0	12	9.7	1.44	1.05	0.94	4.96	1.02	0.90	1.77	1.06	0.93
dart	9.0	31	18	2.84	1.09	1.14	5.56	0.98	0.94	6.26	1.07	1.11
denoise	25	56	1,5	8.22	0.77	0.74	5.66	0.90	0.84	41.5	0.78	0.75
des90	7.6	29	15	3.94	1.00	0.99	4.85	1.00	1.00	11.2	0.98	1.02
gsm∼	17	103	16	2.95	1.12	1.15	5.17	0.95	0.93	4.97	1.08	1.12
LU230	34	281	40	1.20	1.27	0.71	3.79	0.90	0.69	6.55	0.80	0.70
mes_noc	37	86	21	1.98	1.11	0.95	3.88	0.96	0.93	1.90	1.14	0.96
minres	14	53	11	1.10	0.91	0.98	3.79	0.92	0.99	4.31	0.85	0.93
neuron	3.0	13	11	0.39	1.24	1.14	5.40	0.95	0.97	0.38	1.24	1.14
openCV	16	48	13	1.57	1.36	1.17	3.66	0.93	0.93	5.94	1.08	1.11
seg∼	15	26	1.3	5.94	0.82	0.85	4.84	1.01	0.87	31.6	0.83	0.84
SLAM∼	9.2	24	110	6.80	0.83	0.93	5.02	0.87	0.89	37.7	0.83	0.93
sT1_chip2	42	124	32	1.84	1.19	1.17	3.49	0.90	1.02	2.00	1.25	1.18
sT1_core	3.6	12	10	3.08	1.17	1.20	6.15	0.97	0.98	10.2	1.12	1.18
sT2_core	17	50	13	2.93	1.13	1.12	3.95	0.96	1.01	3.76	1.13	1.10
stap_qrd	14	46	12	1.02	0.83	0.90	3.56	0.87	0.92	4.33	0.82	0.91
stereo∼	3.0	11	12	0.38	1.31	1.05	5.66	0.87	0.97	0.38	1.31	1.05
Geomean	**13**	**43**	**26**	**1.93**	**1.07**	**1.00**	**4.56**	**0.94**	**0.94**	**4.55**	**1.03**	**0.99**
Std. Dev.				2.1	0.18	0.14	0.85	0.05	0.08	12.38	0.18	0.13

datapoint ($\rho = 1.11$) in the knee of the analytical placement's curve, so AP can comfortably achieve this quality without slowing down too much. We call this modus *AP-Fast*. To reach the same quality LIQUID needs 20 outer iterations, this setting is called *Liquid-Fast*. The results are reported per benchmark design in the first two major columns of Table I. The results in the first major column *Liquid-Fast* are reported in real numbers. The other datapoints in Table I are reported relative to *Liquid-Fast*. LIQUID is able to produce the results 1.93x faster. Additionally the total wire-length cost of LIQUID's placement is 7% lower. However, we should note that there is a lot of variability if we look across the different designs for all metrics. The speedup is clearly visible with almost all runtimes of AP-fast being slower, but the wire-length and critical path delay results are more erratic, but that can be explained by the fact we are investigating the low effort modus of both placers.

D. The best achievable quality

To investigate the best achievable quality we gave both placement tools more time. We choose a geomean of 60s, because beyond this point the quality did not improve much for both placers. The datapoints under investigation are indicated by the green striped line in Fig. 4. For AP this comes down to $\rho = 1.018$, we call this *AP-HQ* and for LIQUID this equates to setting the number of outer iterations to 80. We call that modus *Liquid-HQ*. The individual results of the benchmark designs are reported in the third and fourth major column of Table I. All results are reported relative to *Liquid-Fast*. For *AP-HQ* we don't observe much quality improvement compared to *AP-Fast* 1% improvement for the critical path delay and 4% for the wirelength cost. For *Liquid-HQ* on the other hand a clear quality improvement compared to *Liquid-Fast* can be observed with a 6% improvement for both the critical path delay and the wirelength cost and clearly a lower variance in the results for the different benchmark designs. Overall this leads to a gap between *AP-HQ* and *Liquid-HQ* of 5% for the critical path delay and 9% for the wirelength cost.

IV. CONCLUSION

The main conclusion of this paper is that it is not necessary to solve the linear system built in conventional analytical placement techniques. It is sufficient to optimize the placement in between legalization phases by following the steepest gradient descent with a momentum simulation. We implemented this method in our new placer called LIQUID and achieved a speedup of 1.9 compared to our own conventional analytical placement implementation. Another important result is that given more time LIQUID is able to find higher quality placements in comparison to our analytical placement implementation with a 5% decrease in critical path delay and 9% decrease in wire-length cost. Unfortunately there is no source code available for analytical FPGA placement described in other publications, so it is hard to compare to them. We built an analytical placement tool ourselves and we want to help the community with releasing our source code for both LIQUID and our analytical placer on Github [8].

REFERENCES

[1] 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.

[2] L. Shannon, V. Cojocaru, C. N. Dao, and P. H. Leong, "Technology Scaling in FPGAs: Trends in Applications and Architectures," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 1–8.

[3] A. Ludwin and V. Betz, "Efficient and Deterministic Parallel Placement for FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 3, pp. 22:1–22:23, Jun. 2011.

[4] M. Gort and J. Anderson, "Accelerating FPGA Routing Through Parallelization and Engineering Enhancements Special Section on PAR-CAD 2010," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 1, pp. 61–74, Jan 2012.

[5] B. Severens, E. Vansteenkiste, K. Heyse, and D. Stroobandt, "Estimating circuit delays in FPGAs after technology mapping," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, Sept 2015.

[6] M.-C. Kim, D. Lee, and I. L. Markov, "SimPL: An Effective Placement Algorithm." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 50–60, 2012.

[7] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs." in *FPL*. IEEE, 2012, pp. 143–150.

[8] E. Vansteenkiste and S. Lenders, "Liquid: Fast FPGA Placement Via Steepest Gradient Descent Movement," https://github.ugent.be/eavsteen/The-Java-FPGA-Placement-Framework, 2016.

[9] Y. Nesterov, "A method of solving a convex programming problem with convergence rate O (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.