Accelerating FPGA-Placement With a Gradient Descent Based Algorithm

Timm Bostelmann, Tobias Thiemann and Sergei Sawitzki

FH Wedel (University of Applied Sciences)
Wedel, Germany
Email: {bos,inf103917,saw}@fh-wedel.de

***Abstract*—Programmable circuits and, nowadays, especially Field-Programmable Gate Arrays (FPGAs) are widely applied in computationally demanding signal processing applications. Considering modern, agile hardware / software codesign approaches, an Electronic Design Automation (EDA) process not only needs to deliver high quality results, but also has to be swift because software compilation is already distinctly faster. Slow EDA tools can in fact act as a kind of show-stopper for an agile development process. One of the major problems in EDA is the placement of the technology-mapped netlist to the target architecture. In this work, a method to reduce the runtime of the netlist placement for FPGAs is evaluated. The approach is a variation of analytical placement, with the distinction that a gradient descent is used for the optimization of the placement. This work is based on previous publications of the authors, in which a placement algorithm using self-organizing maps is introduced and optimized. In comparison, the gradient placement approach is shown to be up to 3.8 times faster than the simulated annealing based reference with about the same quality regarding the bounding-box and routing-resource costs.**

Keywords–EDA; FPGA; placement; gradient descent.

I. Introduction

The ever-growing complexity of Field-Programmable Gate Arrays (FPGAs) has a high impact on the performance of Electronic Design Automation (EDA) tools. A complete compilation from a hardware description language to a bitstream can take several hours. One step highly affected by the vast size of netlists is the NP-equivalent placement process. It consists of selecting a resource cell (position) on the FPGA for every cell of the applications netlist. In previous publications of the authors, a placement algorithm for FPGAs based on a self-organizing map [1] was presented [2] and optimized [3]. With that approach, placements of high quality were produced. However, it was relatively slow for large netlists, even when accelerated using a Graphics Processing Unit (GPU) [4]. Therefore, in this work, a faster approach for netlist placement based on a gradient descent is presented as an updated version of the authors' previous work [4].

Due to the complexity of the netlist placement problem, many current algorithms work in an iterative manner. A well known example is simulated annealing [5], which starts with a random initial placement and swaps blocks stepwise. The result of every step is evaluated by a cost function. A step is always accepted, if it reduces the cost. If it increases the cost, it is accepted with a probability that declines with time (cooling down). An annealing schedule determines the gradual decrease of the temperature, where a low temperature means a low acceptance rate and a high temperature means a high acceptance rate. Generally, the temperature is described by an exponentially falling function like

$$T_n = \alpha^n \cdot T_0, \tag{1}$$

where typically $0.7 \leq \alpha \leq 0.95$. However, there has been a lot of research on the optimization of the annealing schedule like in [6][7]. As a result, there are many variations available for any related problem.

Analytical placement is a different approach, where the problem is described as a system of equations. By solving this system of equations, the optimal position for every element can be derived. However, solving such large equation systems takes much time. Therefore, Vansteenkiste et al. [8] have introduced a method to approximate the solution of the equation system by the steepest gradient descent. This approach is shown to be two times faster than a conventional analytical placement on average, without any penalties in quality.

In this work, a simplified implementation of the steepest gradient descent placement is described and benchmarked extensively. It is not compared to other analytical placement methods. Instead, the established implementation of the simulated annealing approach of the Versatile Place and Route (VPR) tool [9] for FPGAs is used as reference.

In Section II, the problem of netlist placement for FPGAs is introduced and the principle of netlist placement with a gradient descent is described. In Section III, the proposed algorithm is described including some details of its implementation. In Section IV, the results of the proposed algorithm are presented. As representation for real world applications, a set of twenty Microelectronics Center of North Carolina (MCNC) benchmarks [10] is used. Finally, in Section V, the results of this work are summarized and a prospect to further work is given.

II. Background

This section is separated into two parts. First, the problem of netlist placement for FPGAs is introduced. Second, the general idea of using a gradient descent for the placement of netlists for FPGAs is described.

A. Netlist Placement for FPGAs

The problem of netlist placement for FPGAs can be roughly described as selecting a resource cell (a position) on the target FPGA for every cell of the given netlist. In Figure 1, an exemplary graph of a netlist is defined. An exemplary

 ISBN: 978-1-61208-748-1

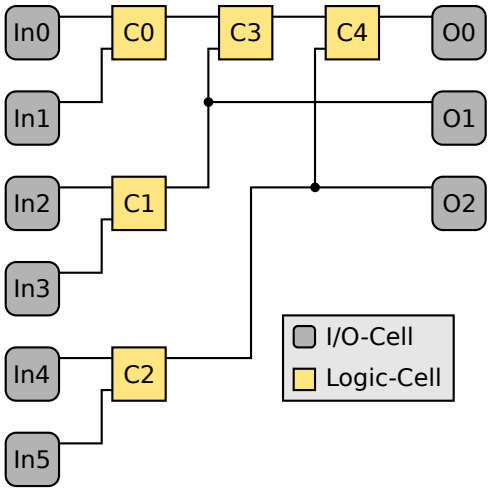

Figure 1. An exemplary graph of a netlist consisting of input-, output-, and logic-cells.

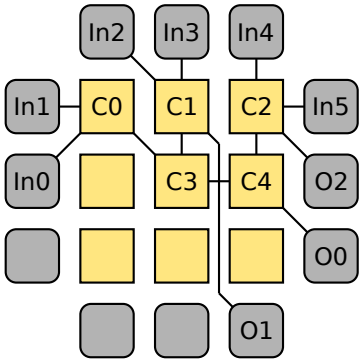

Figure 2. A valid placement for the graph in Figure 1 on a simple island-style FPGA architecture.

placement for this netlist is presented in Figure 2. The positions must be chosen in a way that:

1) Every cell of the netlist is assigned to a resource cell of the fitting type (e.g., Input / Output or Logic).
2) No resource cell is occupied by more than one cell of the netlist.
3) The cells are arranged in a way that allows the best possible routing.

The first two rules are necessary constraints. A placement that is failing at least one of these two constraints is illegal and, therefore, unusable. The third rule is a quality constraint, which is typically described by a cost function. The goal of a placement algorithm is to optimize the placement regarding this function without violating one of the necessary constraints. Usually, the length of the critical path and the routability are covered by the cost function.

B. Netlist Placement With a Gradient Descent

The netlist placement with a gradient descent is done by iteratively optimizing the positions of all elements of the netlist in the direction of the steepest gradient descent. During this process, the nodes are not bound to the grid of the FPGA architecture. Instead, they are positioned in a continuous space. To generate a valid placement – without overlapping and under consideration of the FPGA's architecture – in this approach, a cycle of optimization and legalization is used. This procedure is customary for analytical placement methods for FPGAs, like Gort and Anderson have introduced in [11]. A different approach would be to generate only valid placements by exclusively moving the nodes on the architectural grid of the FPGA.

III. Implementation

A. Gradient Calculation

At the beginning of every optimization step, the bounding-box size of every net in the netlist is determined. This is a necessary preparation for the cost-function, which is described later in this section. To determine the size of a net, all nodes with a connection to the net are determined. For all these nodes, the minimum and maximum of the horizontal positions (X_i) and the vertical positions (Y_i) are determined and stored for the calculation of the gradient. Additionally, the sum of all sizes in X and Y direction is calculated, as a metric for the global quality of the current placement.

The goal of every optimization step is to move the nodes in a direction that leads to a reduction of the bounding-box size of the containing net. A cost-function is necessary to determine the influence of every node on the size of the corresponding net. The gradient of this cost-function can then be used to determine the direction of the movement of each node. All nodes of the netlist are moved towards the steepest gradient descent to reduce the global cost.

An intuitive approach would be to use the sum of the bounding-box sizes of all nets as cost-function. However, with this metric, only the outermost nodes would be moved and even nodes that are very near to the bounding-box would be ignored. Furthermore, the min and max functions contained in the metric can not be derived to calculate the gradient.

To solve these issues, an exponential function over the distance between the position of the node and the bounding-box of the net is chosen as basis of the cost-function. The cost-function for a node with the index k is

$$C_k = \alpha_2 \cdot \sum_{n \in N_k} \Big(e^{\alpha_1 \cdot (x_k - \max_x(n))} + e^{\alpha_1 \cdot (\min_x(n) - x_k)} + e^{\alpha_1 \cdot (y_k - \max_y(n))} + e^{\alpha_1 \cdot (\min_y(n) - y_k)} \Big), \tag{2}$$

where x_k and y_k describe the X and Y coordinates of the current node, N_k describes the set of all nets that contain the node and $\min_x$, $\max_x$, $\min_y$ and $\max_y$ are the minimal and maximal coordinates of the current net (i.e., the bounding-box). α_1 and α_2 are parameters for the cost-function, which allow to influence the behavior of the function. With α_1, it can be determined how large the distance between the node and the bounding-box must be to reduce its influence in the cost-function. The influence of α_1 on the gradient is shown in Figure 3 for the X coordinate of a node, assuming a net with the boundaries $\min_x = 1$ and $\max_x = 7$. With α_2, the cost can be increased or reduced to influence the steepness of the gradient.

Based on (2), the gradients for the X and Y coordinates

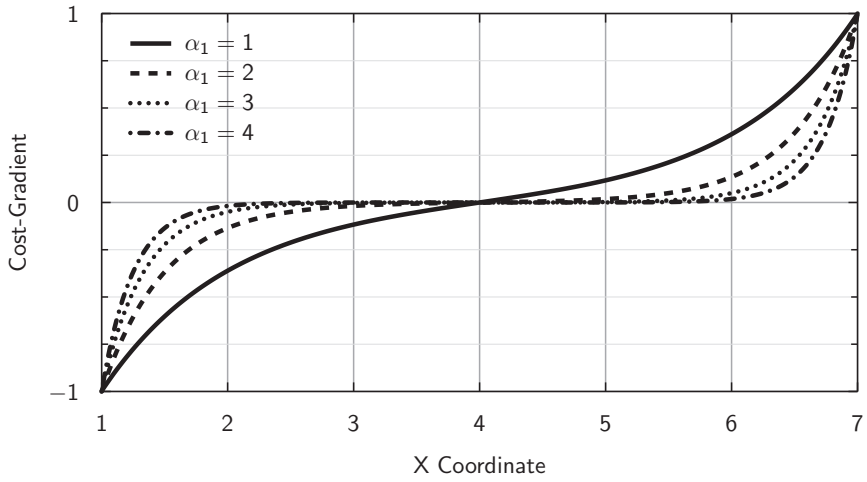

Figure 3. Exemplary plot of possible gradients for the X coordinate of a node, assuming a net with the boundaries $\min_x = 1$ and $\max_x = 7$.

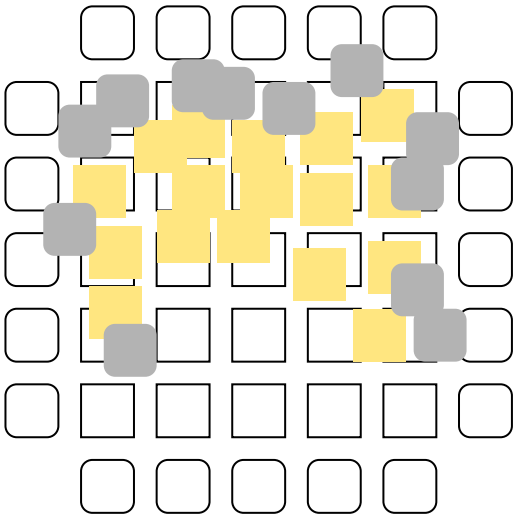

Figure 4. Exemplary placement before the legalization step.

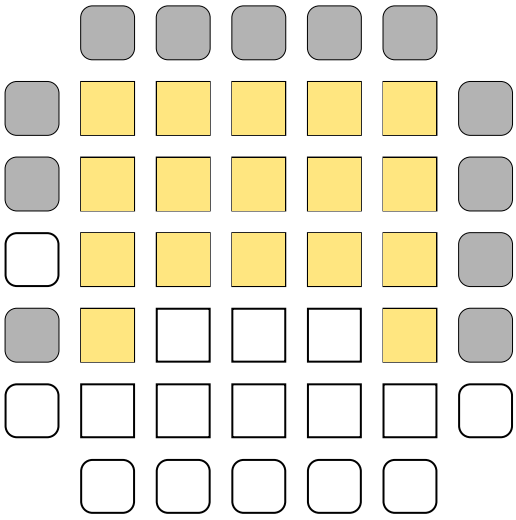

Figure 5. Exemplary placement after the legalization step.

can be calculated as

$$\frac{\partial C_k}{\partial x_k} = \alpha_2 \cdot \sum_{n \in N_k} \left(e^{\alpha_1 \cdot (x_k - \max_x(n))} - e^{\alpha_1 \cdot (\min_x(n) - x_k)} \right), \quad (3)$$

$$\frac{\partial C_k}{\partial y_k} = \alpha_2 \cdot \sum_{n \in N_k} \left(e^{\alpha_1 \cdot (y_k - \max_y(n))} - e^{\alpha_1 \cdot (\min_y(n) - y_k)} \right). \quad (4)$$

As a result, the coordinates of nodes that are near the bounding-box of their containing net have a gradient of $\pm\alpha_2$, where the coordinates of nodes with a larger distance to the bounding-box have a much lower gradient, as shown in Figure 3. Consequentially, nodes with a larger gradient value must be moved further to improve the placement optimally.

B. Legalization

During the optimization step, the nodes can take any position. Thereby, illegal placements are produced, due to overlapping of nodes, as well as violation of the architectural grid of the FPGA. Therefore, the optimized placement must be legalized in a separate step. This is done by finding the nearest valid position for every node, as depicted in Figure 4 (before the legalization) and Figure 5 (after the legalization).

The algorithm for the legalization is inspired by the work of Gort and Anderson [11]. The basic idea of that approach is to find regions that contain more nodes than the corresponding region of the FPGA provides. Then, those regions are gradually expanded. When two regions overlap, they are merged. This is done until the regions are large enough to place all contained nodes to a proper resource cell of the FPGA. In the next step, the regions are split recursively and the nodes are assigned to the new sections by their position. This is repeated until a region contains no more nodes, or only one node. In the latter case, the position of the single remaining node is set to the position of its containing region.

In this work, the search for regions that contain more nodes than the corresponding region of the FPGA provides and the following expansion and merge phases are skipped. Instead, all nodes are assigned to one large region from the start and the phase of recursive splitting starts directly. By this measure, the computational effort for the legalization is

 ISBN: 978-1-61208-748-1

reduced significantly without a dramatic impact on the global quality. This is because – especially when a large amount of the available resources is used – the result of the expansion phase is containing usually very few large regions or often only one large region anyway.

C. Optimization

For the optimization, the algorithm Adam – which was introduced by Kingma and Ba in [12] – is used. The used update rules are:

$$
\begin{aligned}
g_t &= \Delta\phi_t && \text{Gradient of the variable}\\
m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1)\cdot g_t && \text{Running average force one}\\
v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^2 && \text{Running average force two}\\
\hat{m}_t &= m_t / \left(1-\beta_1^t\right) && \text{Bias corrected force one}\\
\hat{v}_t &= v_t / \left(1-\beta_2^t\right) && \text{Bias corrected force two}\\
\phi_t &= \phi_{t-1} - S_a \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t} + \epsilon\right) && \text{Update of the variable}
\end{aligned}
$$

The constants β_1 and β_2 define how fast the averages of the first and second forces change. In this work, the constants were defined as $\beta_1 = 0.96$ and $\beta_2 = 0.998$. The variable S_a defines the learning rate or, more specifically, the step-width. It starts at $S_a = 1.5$, but changes over time (i.e., in the different phases of the placement).

D. Placement Phases

The previously described steps are executed for every iteration. The placement process is separated into five phases, with different parameters. Each phase consists of a given number of iterations. The number of iterations per phase was determined empirically and is fixed (i.e., independent of the size of the design). The phases are:

1) Presorting (5000 iterations)
   In this phase, all nodes are moved with a high step width in the general direction of their final position.
2) Grid placement (1000 iterations)
   In this phase, the force of the legalization is increased. Thereby, the nodes are pulled harder towards legal positions (i.e., to fitting cells of the architecture). This is necessary – for example – to prevent input and output cells from getting stuck in the logic block section of the architecture.
3) Initial detailed placement (1000 iterations)
   In this phase, the global step-width is reduced to one tenth of the initial value. This influences the legalization and the optimization equally, so that the balance between those two steps is not changed. However, the changes are much smaller, resulting in a more precise outcome.
4) Detailed placement (5000 iterations)
   In this phase, the step-width of the optimization is reduced linearly to 20 percent of its original value. Thereby, the nodes are pulled relatively harder towards their final positions in the grid.
5) Final placement (100 iterations)
   In this phase the influence of the optimization is reduced to zero, so that effectively only the legalization is active. Hence, the nodes are moved to their final position in the grid.

TABLE I. A LIST OF THE USED BENCHMARKS AND THEIR CHARACTERISTICS, THE NUMBER OF CLBS, INPUT BLOCKS, OUTPUT BLOCKS AND THE GLOBAL BLOCK COUNT

Name	Inputs	Outputs	CLBs	Blocks
ex5p	8	63	1064	1135
tseng	52	122	1047	1221
apex4	9	19	1262	1290
misex3	14	14	1397	1425
alu4	14	8	1522	1544
diffeq	64	39	1497	1600
dsip	229	197	1370	1796
seq	41	35	1750	1826
apex2	38	3	1878	1919
s298	4	6	1931	1941
des	256	245	1591	2092
bigkey	229	197	1707	2133
frisc	20	116	3556	3692
spla	16	46	3690	3752
elliptic	131	114	3604	3849
ex1010	10	10	4598	4618
pdc	16	40	4575	4631
s38417	29	106	6406	6541
s38584.1	38	304	6447	6789
clma	62	82	8383	8527

IV. RESULTS

In this section, the benchmark results of the previously described placement algorithm are presented. VPR is used as reference for the comparison of the placement results, as well as for the routing and timing analysis.

All used MCNC benchmarks [10] and their characteristics, namely, the number of Configurable Logic Blocks (CLBs), input blocks, output blocks and the sum of all blocks are listed in Table I, sorted by ascending complexity (i.e., the global block count). The netlists are placed on a homogeneous island-style architecture with four input lookup tables.

A. Bounding-Box Costs

The standard metric used for the approximation of the quality of a placement in VPR is the bounding-box cost. It is basically the sum of the half perimeter of the bounding-boxes (i.e., length plus width) of all nets. As introduced by Betz and Rose in [9], the bounding-box metric can be described as

$$
Cost = \sum_{n=1}^{N_{nets}} q(n) \left[\frac{bb_x(n)}{C_{av,x}(n)} + \frac{bb_y(n)}{C_{av,y}(n)} \right], \tag{5}
$$

where $bb_x(n)$ and $bb_y(n)$ describe the horizontal and vertical size of the net n. $C_{av,x}(n)$ and $C_{av,y}(n)$ describe the average capacity of horizontal and vertical channels in the region of the net (in the considered case, the capacity is homogeneous over the whole architecture, so these values are constant. $q(n)$ corrects the effort for nets with more than three nodes, because it would otherwise be approximated to low.

In Table II, the bounding-box costs for the previously introduced benchmark netlists are presented. The results of VPR and the gradient placer are shown as absolute values and in relation to each other:

$$
Cost_{Relative} = \frac{Cost_{VPR}}{Cost_{Gradient}} \cdot 100\,\% \tag{6}
$$

TABLE II. COMPARISON OF THE BOUNDING-BOX COSTS BETWEEN THE GRADIENT PLACEMENT AND THE SIMULATED ANNEALING OF VPR

Netlist	VPR	Gradient	Relative / %
ex5p	180.599	173.701	96.18
tseng	102.398	101.112	98.74
apex4	195.338	190.657	97.60
misex3	200.456	199.160	99.35
alu4	204.692	200.965	98.18
diffeq	155.531	156.375	100.54
dsip	199.845	179.254	89.70
seq	260.789	267.686	102.64
apex2	280.120	293.168	104.66
s298	225.344	217.479	96.51
des	257.643	268.889	104.36
bigkey	209.470	201.344	96.12
frisc	587.227	593.630	101.09
spla	628.155	672.990	107.14
elliptic	497.645	503.854	101.25
ex1010	684.798	720.589	105.23
pdc	939.813	976.890	103.95
s38417	687.198	784.862	114.21
s38584.1	684.220	774.451	113.19
clma	1502.330	1598.670	106.41
Average			101.85

It can be seen that especially the smaller netlists profit from the gradient placement. Remarkably, for all netlists with less than 1600 nodes, the bounding-box costs are less with the gradient placer than with VPR. If the larger netlists are included, the costs for the gradient placer are only 1.85 percent higher on average, which is almost equal.

B. Channel Width

After their generation, the placements were routed with the VPR router and the Channel Width (CW), as well as the amount of necessary wire elements as a measure for the total Wire Length (WL) were determined. The results are shown in Table III. The differences in the channel width are given as a simple delta between the results:

$$\Delta CW = CW_{VPR} - CW_{Gradient} \tag{7}$$

The differences in the wire length are given as ratio between the results in percent:

$$WL_{Relative} = \frac{WL_{VPR}}{WL_{Gradient}} \cdot 100\,\% \tag{8}$$

The needed channel width of the gradient method is on average 0.5 channels smaller than the reference, whereas its total wire length is 0.09 percent longer. Both values are considered to be almost equal to the reference.

C. Runtime

In the previous sections, it was shown that the gradient placer produces a similar placement quality as VPR in regard of the bounding-box cost and the required routing resources. In this section, the runtime of both algorithms is measured and evaluated. The configuration of the system that has been used for the benchmarking is provided in Table IV.

The results are shown in Table V. The presented numbers are each an average of ten measurements. All single measurements varied less than two percent of the average of the measurement series.

TABLE III. COMPARISON OF THE MINIMAL CHANNEL WIDTH (CW) AND THE TOTAL WIRE LENGTH (WL) BETWEEN THE GRADIENT BASED PLACEMENT ALGORITHM AND THE SIMULATED ANNEALING OF VPR

Netlist	VPR		Gradient		Relative	
	CW	WL	CW	WL	ΔCW	WL / %
ex5p	15	20034	14	19541	-1	97.54
tseng	8	10200	7	9463	-1	92.77
apex4	15	22215	13	22116	-2	99.55
misex3	13	21884	12	21820	-1	99.71
alu4	12	22319	11	21261	-1	95.26
diffeq	9	15369	8	15292	-1	99.50
dsip	7	18065	7	15260	0	84.47
seq	12	28469	13	28977	1	101.78
apex2	12	30826	12	31905	0	103.50
s298	8	22335	9	21801	1	97.61
des	9	28084	9	28764	0	102.42
bigkey	8	21424	7	20315	-1	94.82
frisc	17	63146	14	64220	-3	101.70
spla	16	68364	16	72288	0	105.74
elliptic	11	44742	12	51127	1	114.27
ex1010	13	71891	12	73653	-1	102.45
pdc	19	104065	19	106057	0	101.91
s38417	8	64626	9	68999	1	106.77
s38584.1	10	64626	9	64180	-1	99.31
clma	14	141660	14	142695	0	100.73
Average					-0.5	100.09

TABLE IV. CONFIGURATION OF THE SYSTEM THAT HAS BEEN USED FOR THE BENCHMARKING OF THE GRADIENT ALGORITHM AND VPR

Property	Value
Processor	Intel® Core™ i7-4510U
Cores	2
Threads	4
Base Frequency	2.00 GHz
Turbo Frequency	3.10 GHz
Cache	4 MB
RAM	16 GB

On average, the gradient based placement algorithm needs less than half of the time of the simulated annealing placer of VPR. Furthermore, the ratio is even better for large netlists, as can be seen clearly in Figure 6. For example, the largest netlist in this benchmark series – the clma netlist – is placed 3.8 times faster with the gradient based approach.

V. CONCLUSION AND FUTURE WORK

In this work, a fast approach for netlist placement based on a gradient descent was presented. The gradient placer was compared to the simulated annealing based placer of VPR. It has been shown that the quality of the placement in regard of the bounding-box cost and the occupation of routing resources (i.e., channel width and total wire length) is equal to the reference within a reasonable margin of error, as proven by placing twenty prominent benchmarking netlists of different complexity. Notably, the presented approach is shown to be up to 3.8 times faster than the reference. On average, it needs less than half of the time to compute the result. However, preliminary results show that the resulting length of the critical path is worse with the gradient placer (about twenty percent for the largest netlist in this work). This would need to be addressed in future work.

 ISBN: 978-1-61208-748-1

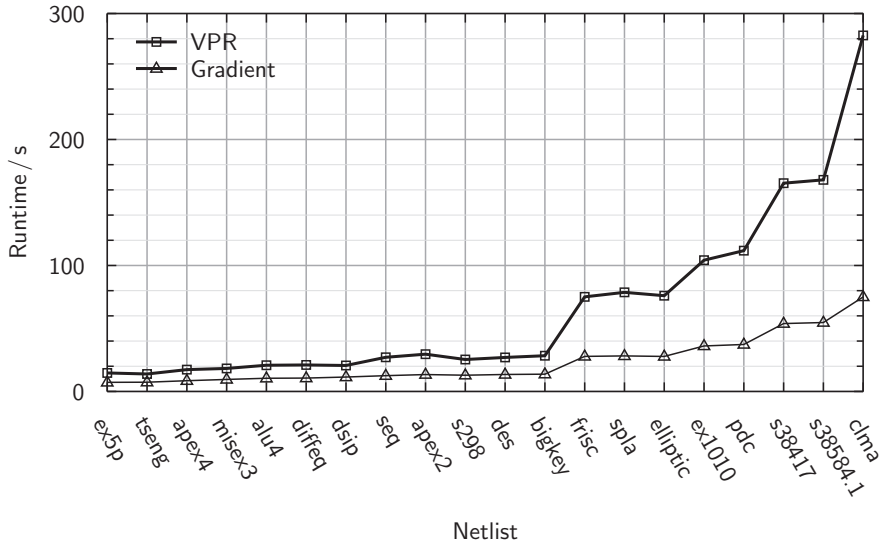

Figure 6. Diagram of the runtime as average of ten measurements between the gradient based placement algorithm and the simulated annealing of VPR.

TABLE V. COMPARISON OF THE RUNTIME AS AVERAGE OF TEN MEASUREMENTS BETWEEN THE GRADIENT BASED PLACEMENT ALGORITHM AND THE SIMULATED ANNEALING OF VPR

Netlist	VPR / s	Gradient / s	Relative / %
ex5p	14.69	7.23	49.23
tseng	13.86	7.34	53.00
apex4	17.34	8.53	49.16
misex3	18.27	9.54	52.20
alu4	20.81	10.48	50.36
diffeq	21.05	10.63	50.47
dsip	20.62	11.45	55.54
seq	27.12	12.53	46.21
apex2	29.60	13.41	45.31
s298	25.35	12.90	50.90
des	27.01	13.48	49.92
bigkey	28.36	13.72	48.36
frisc	75.10	27.83	37.05
spla	78.67	28.21	35.85
elliptic	76.02	27.79	36.56
ex1010	104.21	36.11	34.65
pdc	111.76	37.30	33.37
s38417	165.32	53.89	32.60
s38584.1	167.96	54.72	32.58
clma	282.60	75.04	26.55
Average			43.49

As the current implementation of the gradient placer is executed only single-threaded, the next logic step would be to parallelize its execution to make it even faster. The calculation of the gradients could be executed in parallel on node level, and even large parts of the legalization (e.g., the assignment of nodes to the regions) could be parallelized. Hence, a multi-threaded implementation would be beneficial and even a GPU-computing approach seems to be promising.

Even though the gradient placement approach was shown to be comparably fast for large netlists, a more recent set of benchmarks like the one included in [13] – containing much larger netlists – could be used to underline the scalability of the approach.

REFERENCES

[1] T. Kohonen, Self-Organizing Maps. Springer, 1995.

[2] T. Bostelmann and S. Sawitzki, “Improving FPGA placement with a self-organizing map,” in International Conference on Reconfigurable Computing and FPGAs (ReConFig), December 2013, pp. 1–6.

[3] T. Bostelmann and S. Sawitzki, “Improving the performance of a SOM-based FPGA-placement-algorithm using SIMD-hardware,” in The Ninth International Conference on Advances in Circuits, Electronics and Micro-electronics (CENICS), July 2016, pp. 13–15.

[4] T. Bostelmann, P. Kewisch, L. Bublies, and S. Sawitzki, “Improving FPGA-placement with a self-organizing map accelerated by GPU-computing,” International Journal On Advances in Systems and Measurements, vol. 10, no. 1 & 2, 2017, pp. 45–55.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, May 1983, pp. 671–680.

[6] L. Ingber, “Adaptive simulated annealing (ASA): Lessons learned,” Control and Cybernetics, vol. 25, 1996, pp. 33–54.

[7] M. M. Atiqullah, “An efficient simple cooling schedule for simulated annealing,” in International Conference on Computational Science and Its Applications (ICCSA). Springer, 2004, pp. 396–404.

[8] E. Vansteenkiste, S. Lenders, and D. Stroobandt, “Liquid: Fast placement prototyping through steepest gradient descent movement,” in 2016 26th International Conference on Field Programmable Logic and Applications (FPL), August 2016, pp. 1–4.

[9] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in International Conference on Field Programmable Logic and Applications (FPL). Springer, 1997, pp. 213–222.

[10] S. Yang, “Logic synthesis and optimization benchmarks user guide version 3.0,” Microelectronics Center of North Carolina, Tech. Rep., 1991.

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

[12] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015, pp. 1–15.

[13] J. Luu et al., “VTR 7.0: Next generation architecture and CAD system for FPGAs,” ACM Trans. Reconfigurable Technol. Syst., vol. 7, no. 2, June 2014, pp. 6:1–6:30.

 ISBN: 978-1-61208-748-1