Improving FPGA-Placement with a Self-Organizing Map Accelerated by GPU-Computing

Timm Bostelmann, Philipp Kewisch, Lennart Bublies and Sergei Sawitzki

FH Wedel (University of Applied Sciences)
Wedel, Germany
Email: bos@fh-wedel.de, publications@kewis.ch, edu@bublies-it.de, 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. It 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 mayor problems in EDA is the placement of the technology-mapped netlist to the target architecture. In this work a method to improve the results of the netlist placement for FPGAs with a self-organizing map is presented. The admittedly high computational effort of this approach is covered by the exploitation of its inherent parallelism. Different approaches of parallelization are introduced and evaluated. A concept to accelerate the self-organizing map by using the single instruction multiple data (SIMD) capabilities of the central processing unit (CPU) and the graphics processing unit (GPU) for low-level vector operations is presented. This work is based on our previous publications, which are joined, updated and extended. Specifically, a new metric to generate training vectors for the self-organizing map – that has been introduced by Amagasaki et al. – was integrated into our work. It is shown that – in case of our application – the original vectorization metric creates higher quality results, even though the new metric is unmistakably faster. Addressing this issue, in addition to the previous low-level parallelization, a new high-level parallelization approach is introduced and detailed benchmark results are presented.**

Keywords–FPGA; netlist placement; OpenCL; GPU-computing; parallelization; SIMD.

I. Introduction

The ever-growing complexity of FPGAs has a high impact on the performance of 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-complete placement process. It consists of selecting a resource cell (position) on the FPGA for every cell of the applications netlist. In this work, our previous publications regarding the optimization of the placement process [1], [2] are joined, updated and extended. Explicitly, a new GPU-accelerated implementation is presented and benchmarked. Furthermore, an additional method for the generation of training vectors is evaluated.

Due to the complexity of the netlist placement problem, many current algorithms work in an iterative manner. A well known example is simulated annealing [3], 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 by 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 [4], [5], [6]. As a result, there are many variations available for any related problem.

It has been shown by Banerjee et al. [7] that the speed and result of an iterative placement algorithm can be improved by the use of an initial placement created in a constructive manner out of the structural information of the netlist. For this purpose, the netlist was recursively bisectioned, resulting in an one-dimensional mapping. This mapping was spread to a two-dimensional plane with space-filling curves to create an initial placement for the simulated annealing algorithm. In comparison to the classical random initialization the computation time was reduced by about 44.5 percent without having a significant impact on the quality.

Self-organizing maps [8] – also known as Kohonen maps after their inventor Teuvo Kohonen – are used to classify multidimensional datasets. They belong to the group of unsupervised learning algorithms. Therefore, neither the input data nor the resulting classes have to be known beforehand. The input data is grouped by similarity and mapped to an usually two-dimensional plane.

In this work, it is shown how a self-organizing map can be adapted to map a netlist to a two-dimensional plane and how a valid placement for the netlist can be derived. Additionally, different approaches to utilize the inherent parallelism of this modified self-organizing map are introduced and evaluated. These approaches are based on using the SIMD capabilities of the CPU and the GPU.

In Section II, the problem of netlist placement for FPGAs is introduced and the functional principle of a self-organizing map is described. Furthermore, the basics and challenges of GPU-computing are introduced. In Section III, the proposed algorithm is described including details on how the training algorithm of the self-organizing map has been modified to assure that only valid placements are produced. Furthermore, different metrics for the mapping of the structural netlist-information to so called training vectors are introduced and evaluated. In Section IV, the results of a prototypic software implementation of the proposed algorithm are presented. A reasonable usage of the structural information is proven by placing synthetic, homogeneous netlists. As representation for real world applications a selection of Microelectronics Center of North Carolina (MCNC) benchmarks [9] is introduced. A modified version of the Versatile Place and Route (VPR) [10] tool for FPGAs is used to show the gain of using an initial placement for simulated annealing, which has been created using a self-organizing map. In Section V, the levels of parallelism inherent to the self-organizing map (i.e., vector-level and map-level) are analyzed and different approaches to exploit them are introduced. Specifically, a low-level and a high-level parallelization approach on CPU and GPU are described and benchmarked in detail. Finally, in Section VI, this work is summarized and a prospect to further work is given.

II. Background

In the following subsections the problem of netlist placement for FPGAs is introduced and the functional principle of a self-organizing map is described. Furthermore, the basics and challenges of GPU-computing are introduced.

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 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., IO, CLB or DSP).
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 them 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. Principle of self-organizing maps

A self-organizing map is a special kind of artificial neuronal network. Figure 3 shows the general structure of a two dimensional self-organizing map. It consists of two layers, the competition layer $K_{i,j}$ with $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$ and the input layer E_k with $k \in \{1, 2, \ldots, l\}$. The neurons of the competition layer are placed in a two dimensional grid. They are horizontally and vertically adjacent. Furthermore, every neuron of the competition layer is connected to every neuron of the input layer by a weight $W_{i,j,k}$. The input layer corresponds to a vector with l elements and is able to classify l-dimensional input data.

In Figure 4, the training-cycle of a self-organizing map is shown as a flowchart. The self-organizing map is trained

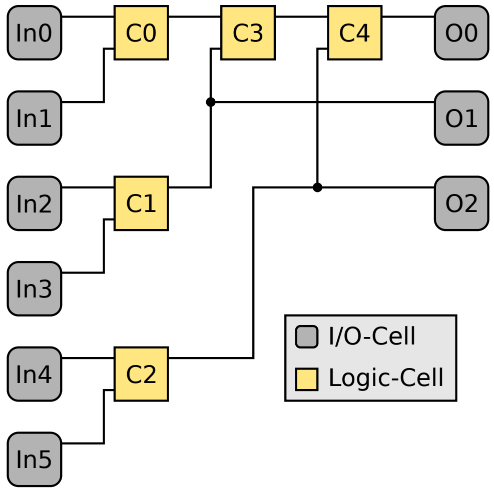

Figure 1. An exemplary graph of a netlist consisting of input-, output-, and logic-cells

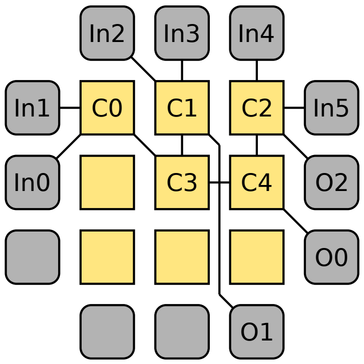

Figure 2. A valid placement for the graph in Figure 1 on a simple island-style FPGA architecture

by repeated stimulation of the input layer with the input data (training vectors) in a random order. In every step – thus for every stimulation – a winning neuron is determined by the distance of its weight vector to the current stimulation of the input layer, so that the neuron with the smallest Euclidean distance to the training vector wins. After this step the weights of the winning neuron and its neighbors are pulled towards the current stimulation by the function

$$W'_{ijk} = W_{ijk} + (E_k - W_{ijk}) \cdot \beta_{ij} \tag{2}$$

where $0 \leq \beta_{ij} \leq 1$ is the influence. The influence is depending on the distance to the winning neuron on the competition layer by the function

$$\beta_{ij} = e^{-\left(\frac{|I-i| + |J-j|}{r}\right)} \tag{3}$$

where (I, J) is the position of the winning neuron, so that $|I-i|+|J-j|$ is the rectilinear distance between the influenced and the winning neuron, and r is the radius of the function. Hence, the influence on the winning neuron itself is the highest and decreases by distance. Consequentially, similar training vectors stimulate mainly adjacent neurons, so that a clustering by similarity is developed.

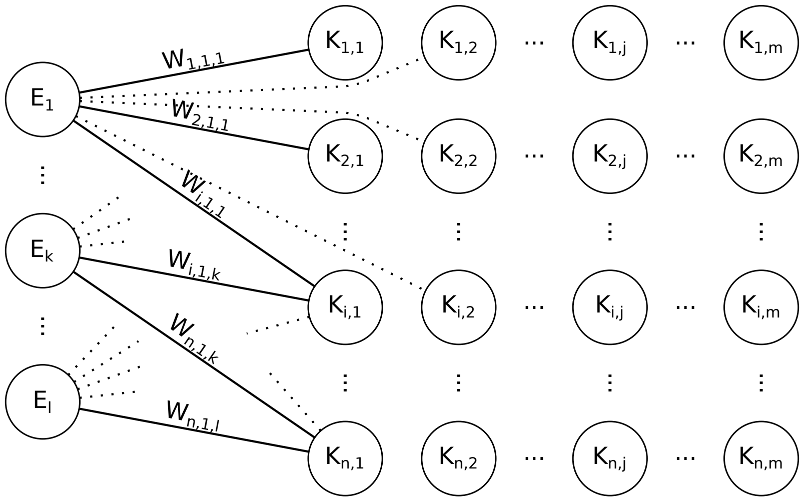

Figure 3. General structure of a self-organizing map

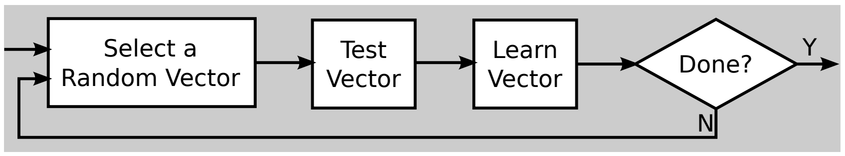

Figure 4. Flowchart of the training-cycle of a self-organizing map

C. GPU-computing with OpenCL

OpenCL is an universal interface for parallel SIMD-computing. It supports various kinds of target hardware. These are multicore CPUs and their streaming extensions as well as GPUs and even special hardware like FPGAs. Especially GPUs are – due to their structure – able to execute large amounts of uniform tasks in parallel. For example, the *AMD® RADEON™ RX 480* GPU is specified with a peak performance of up to 5.8 teraflops, utilizing 2304 stream processors and a memory bandwidth of 224 gigabytes per second. This computation power is usually used for the calculation of pixel-colors in a three-dimensional scene, namely computer games. Even so, thanks to interfaces like OpenCL it is also available for general purpose computing. However, due to the special hardware architecture of GPUs, several specifics must be taken into account when using OpenCL. Besides the obvious need for parallelization, the differences regarding the memory model convey the highest impact to the programmer. Instead of a global memory model with transparent caches, which is used in CPUs, an explicit multi level model is used. In Figure 5, the memory model of OpenCL is shown. It can be mapped to any recent GPUs memory structure. The work-items of the GPU are grouped to workgroups. Each workgroup shares a fast local memory. Work-items can be efficiently synchronized within a workgroup. An exchange of data over the boundaries of workgroups is only possible by using the global memory.

Assuming the GPU implements dedicated memory – as every GPU with considerable computing power does – the transfer between host memory and global memory is comparably slow and has to be reduced to the minimum. Even though the global memory of the GPU is noticeably faster than the host memory of the host-device, it has to feed all the work-items. Thus, the global memory should be used as sparsely as possible. Instead, the workgroup's local memory should be used, if applicable. Copying data from the global memory to the local memory should be done sequentially to exploit the burst capabilities of the dynamic random access memory (RAM).

Finally, it has to be noted that all parameters like the size of the workgroups and the speed and the size of the memories are varying significantly between different devices, let alone different device-classes. Therefore, an implementation performing well on one GPU might lack performance on another model.

III. Proposed Method

In the following subsections the proposed algorithm is described including details on how the training algorithm of the self-organizing map has been modified to assure that only valid placements are produced. Furthermore, different metrics for the mapping of the structural netlist-information to so called training vectors are introduced and evaluated.

A. Principle of netlist placement with a self-organizing map

To generate a netlist placement with a self-organizing map, in addition to the training process described above two general steps are necessary (Figure 6). Those are the generation of training vectors (preparation) and the extraction of placement information from the self-organizing map after the training (interpretation).

For every cell of the netlist, which has to be placed, a training vector is generated in a way that highly connected cells are represented by similar vectors. Since the self-organizing map will cluster these vectors by similarity, the vectors of highly connected cells will cluster together on the competition

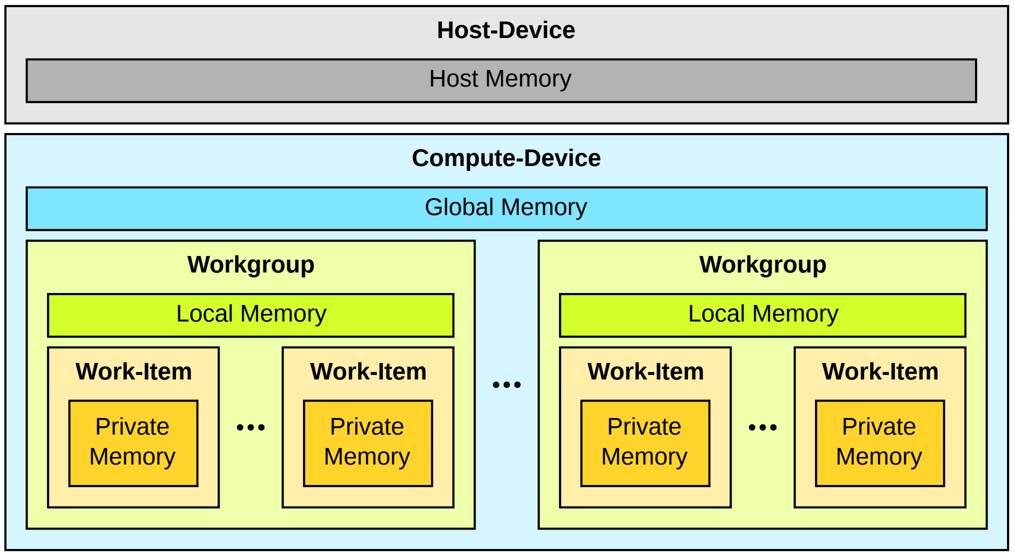

Figure 5. The OpenCL memory- and computation-model

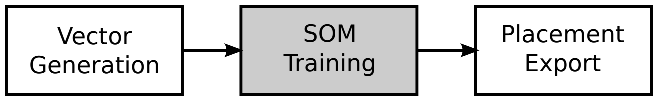

Figure 6. Flowchart of the placement algorithm based on a self-organizing map (SOM)

layer. A favorable placement of the cells can therefore be determined by the positions of the corresponding training vectors on the competition layer. To allow a distinct interpretation the neurons of the competition layer are arranged in a one-to-one mapping with the FPGA resources. This means every neuron is corresponding to a resource cell and the neighborhood relationships between the neurons are corresponding to the interconnections of the FPGA architecture.

To support different cell types (e.g., logic bocks and input / output blocks) every neuron is tagged with the type of the corresponding FPGA resource cell and every training vector is tagged with the type of the corresponding cell of the netlist. During the determination of the winning neuron only neurons of the same type as the training vector are analyzed, so that only a fitting neuron (position) can win. The training – namely the manipulation of the weights around the winning neuron – happens independently of the type to assure a global clustering. On the level of the neuronal model this means that there is one input layer for every cell type. The neurons of the competition layer are connected only to those input neurons that are of the same type as their corresponding resource cell. Consequentially, the training vectors are stimulating only the input neurons that are of the same type as their corresponding cell of the netlist.

B. *Mapping of the structural information into training vectors*

As shown before the training data has to be available in form of homogeneous sized vectors – one for every cell of the netlist – to be processed by the self-organizing map. Five metrics for the generation of these vectors, depending on the structural information of the netlist, have been evaluated.

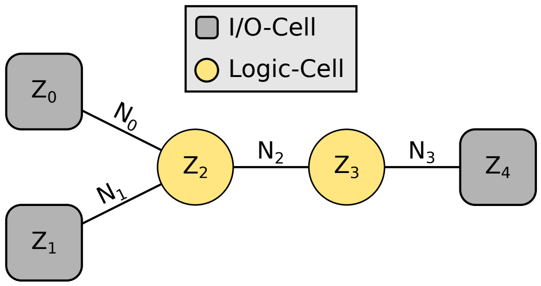

Figure 7. Graph of an exemplary netlist

TABLE I. Training vectors for the graph shown in Figure 7 generated by the vectorization method “net membership”

Cell	Vector
Z_0	$(1, 0, 0, 0)$
Z_1	$(0, 1, 0, 0)$
Z_2	$(1, 1, 1, 0)$
Z_3	$(0, 0, 1, 1)$
Z_4	$(0, 0, 0, 1)$

Metric 1 Net membership: In the first metric the vectors are generated depending on the membership of cells in nets. The dimension of the vectors is equal to the number of nets in the netlist. Thereby, every element of a vector is mapped to a net of the netlist. An element is 1, if the cell corresponding to the vector is connected to the respective net, otherwise it is 0. The vector generation using this approach is very fast because the vectors are generated directly from the netlist. Figure 7 shows a graph of a simple netlist, where Z_i for $i \in \{0, 1, \ldots, 4\}$ are cells and N_j for $j \in \{0, 1, \ldots, 3\}$ are nets of the netlist. The vectors generated for this graph are shown in Table I.

Metric 2 Hyperbolic distance: In the second metric the vectors are generated depending on the pairwise distance between cells. The dimension of the vectors is equal to the number of cells in the netlist. Thereby, every element of a vector is mapped to a cell of the netlist. Let V_i be the vector

TABLE II. Training vectors for the graph shown in Figure 7 generated by the vectorization method "hyperbolic distance"

Cell	Vector
Z_0	$\left(\frac{1}{1}, \frac{1}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right)$
Z_1	$\left(\frac{1}{3}, \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right)$
Z_2	$\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{1}, \frac{1}{2}, \frac{1}{3}\right)$
Z_3	$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{2}, \frac{1}{1}, \frac{1}{2}\right)$
Z_4	$\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{1}{1}\right)$

TABLE III. Training vectors for the graph shown in Figure 7 generated by the vectorization method "linear distance"

Cell	Vector
Z_0	$\left(\frac{4}{4}, \frac{2}{4}, \frac{3}{4}, \frac{2}{4}, \frac{1}{4}\right)$
Z_1	$\left(\frac{2}{4}, \frac{4}{4}, \frac{3}{4}, \frac{2}{4}, \frac{1}{4}\right)$
Z_2	$\left(\frac{3}{4}, \frac{3}{4}, \frac{4}{4}, \frac{3}{4}, \frac{2}{4}\right)$
Z_3	$\left(\frac{2}{4}, \frac{2}{4}, \frac{3}{4}, \frac{4}{4}, \frac{3}{4}\right)$
Z_4	$\left(\frac{1}{4}, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, \frac{4}{4}\right)$

corresponding to the cell Z_i and let d_{ij} be the minimal distance between the cells Z_i and Z_j. The hyperbolic equation

$$v_{ij} = \frac{1}{1 + d_{ij}} \quad (4)$$

describes the generation of the training vectors. Table II shows the vectors generated for the graph (Figure 7) used in the previous example.

Metric 3 Linear distance: The third metric – like the second one – depends on the pairwise distance between cells. Therefore, the structure of the vectors is the same. In addition to the former definition let d_{max} be the greatest distance occurring in the netlist. The linear equation

$$v_{ij} = 1 - \frac{d_{ij}}{d_{max} + 1} \quad (5)$$

describes the generation of the training vectors. Table III shows the vectors generated for the graph (Figure 7) used in the previous examples.

Metric 4 Distance to I/O-cells: This metric has to the best of our knowledge been introduced by Amagasaki et al. in [11]. Instead of the pairwise distances between all cells, only the distances to the input- and output-cells (I/O-cells) are used. The dimension of the vectors is equal to the number of the I/O-cells in the netlist. Table IV shows the vectors generated for the graph (Figure 7) used in the previous examples.

Metric 5 Hyperbolic distance to I/O-cells: The fifth metric is equal to the fourth metric, with the difference that the distances are normalized by the hyperbolic equation (4) like the third metric. Table V shows the vectors generated for the graph (Figure 7) used in the previous examples.

TABLE IV. Training vectors for the graph shown in Figure 7 generated by the vectorization method "I/O-distance"

Cell	Vector
Z_0	$(0, 2, 3)$
Z_1	$(2, 0, 3)$
Z_2	$(1, 1, 2)$
Z_3	$(2, 2, 1)$
Z_4	$(3, 3, 0)$

TABLE V. Training vectors for the graph shown in Figure 7 generated by the vectorization method "hyperbolic I/O-distance"

Cell	Vector
Z_0	$\left(\frac{1}{1}, \frac{1}{3}, \frac{1}{4}\right)$
Z_1	$\left(\frac{1}{3}, \frac{1}{1}, \frac{1}{4}\right)$
Z_2	$\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{3}\right)$
Z_3	$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{2}\right)$
Z_4	$\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{1}\right)$

A disadvantage of the the second and third metric is that the pairwise distance between all cells has to be determined before the training vectors can be generated. Thanks to heuristics like the one introduced by Edmond Chow [12] this is not as time consuming as it might seem. An advantage of the fourth and fifth metric is the comparatively small vector size.

C. Assuring a valid placement

A problem of all proposed metrics is that very similar vectors will not activate two adjacent neurons as desired, but exactly the same neuron. This leads to the generation of an invalid placement because the cells must be placed distinctly and are not allowed to overlap. The placement could be legalized in an additional step, but this would clearly increase the computational effort. An approach of making the vectors distinguishable by the addition of orthogonal data, which causes repulsion between them, also drastically increases the computational effort for the self-organizing map. Therefore, both approaches were rejected.

Instead, the self-organizing map was modified in a way that the activation of the same neuron by different vectors is already prevented during the training and thereby only valid placements are generated. Therefore, first of all the training of the self-organizing map was divided into training-cycles, where each training-cycle means the stimulation with all training vectors in a random succession. Furthermore, neurons that have already won during a training-cycle are blocked until the end of this cycle, so that they cannot win again. On the level of the neuronal model this means the connection between the winning neuron on the competition layer and the input layer is temporarily muted until the end of the training-cycle. Thereby, it cannot be activated again by a similar vector corresponding

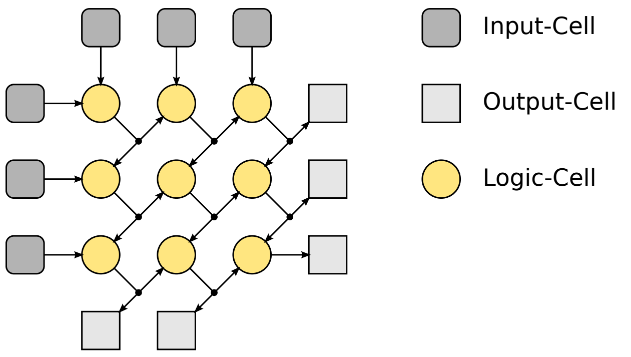

Figure 8. Synthetic, homogeneous graph of the size three

TABLE VI. Results for a synthetic, homogeneous 8 × 8 graph similar to Figure 8

Nr.	Metric	Channels	Path length
1	VPR	8	10.9 ns
2	membership	10	12.8 ns
3	linear distance	8	10.9 ns
4	hyperbolic distance	6	10.4 ns
5	I/0-distance	8	10.6 ns
6	hyperbolic I/0-distance	8	10.7 ns
7	hyperbolic distance	8	9.0 ns

to another cell. Instead, because the neighborhood influence between the neurons on the competition layer remains active, a similar vector will probably activate a neuron adjacent to the blocked one. All the blocked neurons are released at the beginning of every cycle. The mandatory competition between the vectors about the neurons on the competition layer is not suppressed by the blocking because of two reasons. First, because in every cycle the vectors are used in a random succession, a different vector has the chance to be the first one in each cycle. Second, because neurons that have been blocked are still influenced by their neighbors for the rest of a cycle, neurons may "attract" totally or marginally different vectors in the next cycle, depending on how much they have been influenced.

With this approach not only the generation of a valid placement is assured, but also the computational effort of the determination of the winning neuron is reduced. This is because there is no need to evaluate the distances between the input vector and the weight vectors of the blocked neurons, which cannot win anyway.

IV. Results

For a first analysis the proposed method was implemented prototypically in Python. The focus of this implementation lies in adaptivity and interchangeability of the different modules instead of a high computational performance. The software has been used to evaluate the five metrics for vector generation proposed in Section III. Therefore, synthetic, homogeneous netlists were placed by the self-organizing map and routed by VPR. Figure 8 shows a graph of the used netlist of the size three meaning 3×3 logic blocks plus input and output blocks.

Table VI shows the results for a similar graph of the size eight. The first line contains the results of VPR and should be considered as the reference. The lines two to six show the results of the different vectorization metrics for the self-organizing map. The result achieved with vectorization by net membership is worse than the reference solution of VPR both in terms of the channel width and the length of the critical path. The result achieved by linear distance is similar to the reference. The vectorization by hyperbolic distance produces a smaller minimal channel width than the reference (see line four). This placement is one of the ideal solutions for the given problem. To allow a fair comparison of the critical path's length the placement was routed again with the channel width achieved by VPR. The result is shown in line seven. The results for the I/O-distance based vector generation – with and without normalization – are shown in line six and seven. Both generate slightly better results than VPR, but the difference is in the margin of error. As can be seen the critical path of the reference placement generated by simulated annealing is about 21 percent longer than the one generated by the self-organizing map with vectorization by hyperbolic distance.

TABLE VII. Results for a synthetic, homogeneous 16 × 16 graph similar to Figure 8

Nr.	Metric	Channels	Path length
1	VPR	8	21.1 ns
2	membership	12	33.9 ns
3	linear distance	8	25.5 ns
4	hyperbolic distance	8	18.6 ns
5	I/O-distance	10	26.1 ns
6	hyperbolic I/O-distance	8	25.9 ns

TABLE VIII. Characteristics of the selected MCNC benchmarks

Nr.	Name	CLBs	Nets	Inputs	Outputs
1	e64	273	338	65	64
2	ex5p	1 064	1 072	8	63
3	apex4	1 261	1 270	9	18
4	misex3	1 397	1 411	14	14
5	alu4	1 522	1 536	14	8
6	seq	1 750	1 791	41	35
7	apex2	1 878	1 916	38	3
8	ex1010	4 598	4 608	10	10

Table VII shows the results for a graph of the size 16. These results and further tests with synthetic benchmarks have shown that the vectorization by hyperbolic distance creates the best results, often even ideal ones. It has to be mentioned that the large vector-size of this method has a high impact on the the computation time of the self-organizing map. If this poses a problem, then a vectorization method based on the I/O-distance should be considered. However, this work concentrates on the quality of the placement by selecting the hyperbolic distance method. The computational effort is handled by parallelization and utilizing the GPU.

To test the suitability of the self-organizing map for real world netlists a selection of MCNC benchmarks was used. Netlists with a global routing flag – often used for the clock signal – were not supported by the first prototypic implementation and therefore were not examined. Table VIII shows the selected benchmarks and their characteristics, namely the number of logic blocks (CLBs), nets, input and output pins.

In a first approach the MCNC benchmark netlists were placed by the self-organizing map and routed by VPR like

TABLE IX. Placement results for MCNC benchmarks generated by the self-organizing map (SOM) in comparison to the classical annealing with random initialization (VPR)

		Critical Path		Min. Channels	
Nr.	Name	VPR	SOM	VPR	SOM
1	e64	7.4 ns	10.5 ns	14	10
2	ex5p	10.4 ns	16.4 ns	20	36
3	apex4	10.1 ns	14.7 ns	22	42
4	misex3	8.8 ns	11.6 ns	18	48
5	alu4	11.0 ns	16.0 ns	16	48
6	seq	8.7 ns	15.5 ns	18	46
7	apex2	10.4 ns	22.6 ns	20	46
8	ex1010	17.1 ns	31.8 ns	18	72

TABLE X. Detailed placement results for the MCNC benchmark *e64* generated by the self-organizing map in comparison to the classical annealing with random initialization (VPR)

Nr.	Metric	Channels	Path length
1	VPR	14	7.4 ns
2	Membership	10	11.9 ns
3	Membership	14	11.7 ns
4	linear distance	10	9.5 ns
5	linear distance	14	9.5 ns
6	hyperbolic distance	10	10.5 ns
7	hyperbolic distance	14	10.2 ns

the synthetic benchmarks before. The results are shown in Table IX. It is obvious that the self-organizing map is not able to compete with the reference by any measure. The MCNC benchmarks (like real world applications) are not as structured as the previous synthetic examples. Because the self-organizing map only uses the structural information of the netlist and neither the critical path's length, nor the channel width is optimized directly, the sobering results concerning these two indicators are not surprising.

The only exception is how well the self-organizing map handles the *e64* netlist regarding the minimal channel width (see line one of Table IX). Because of this property, the detailed results of this particular netlist were analyzed and are shown in Table X. Even though the vectorization by liner distance surpasses the vectorization by hyperbolic distance in this special case, the latter method is kept up. The different results of the *e64* netlist are ascribed to its special structure and characteristics. The *e64* netlist has an unusually high I/O to CLB ratio, which leads to a full occupation of the surrounding I/O-cells, whereas the CLB-cells are used sparsely. In this special case the self-organizing map tends to scatter the logic over the whole plane (Figure 9), thereby optimizing the routability and channel width. The simulated annealing in contrast groups the logic in one part of the plane (Figure 10) because it primarily optimizes the length of the critical path.

Because of the formerly stated drawbacks in using the self-organizing map directly for the generation of the final placement, its suitability as an initial placement for the iterative algorithm simulated annealing was examined. The initial temperature of the simulated annealing process was reduced, so that only approximately the final 20 percent of the usual swapping steps are executed. By this it is assured that the generated initial placement is not "melted down" completely,

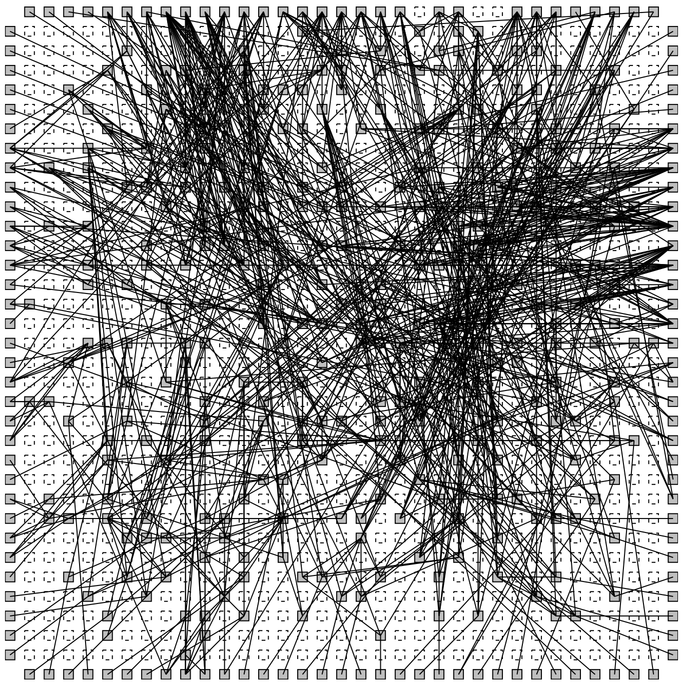
Figure 9. A placement generated with a self-organizing map for the *e64* netlist on a 33 × 33 CLB island-style architecture. Visualization by VPR

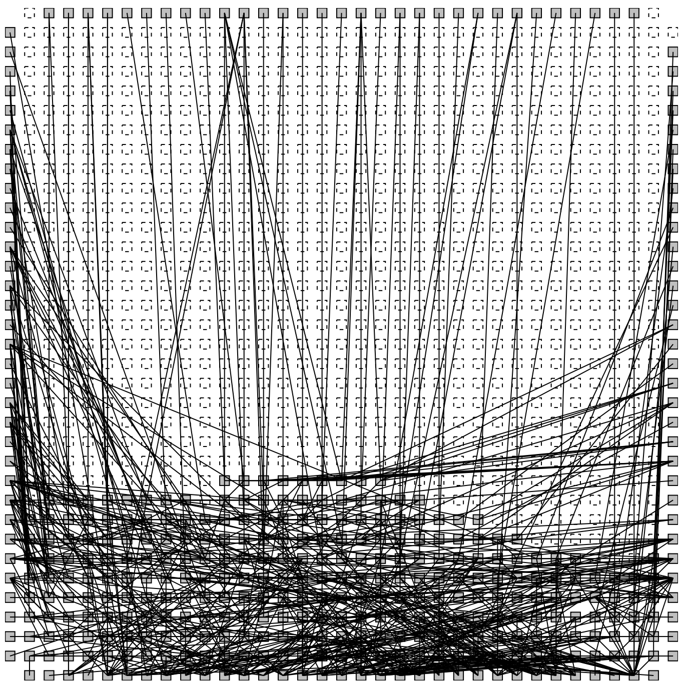
Figure 10. A placement generated with simulated annealing for the *e64* netlist on a 33 × 33 CLB island-style architecture. Visualization and placement by VPR

which would result in the loss of the structural information gained through the self-organizing map. Instead, it is optimized locally to improve the length of the critical path and the minimal channel width. For this series of measurements VPR was modified to allow the loading of an initial placement and used with a custom annealing schedule.

TABLE XI. Placement results for MCNC benchmarks generated by the self-organizing map (SOM) with additional low temperature annealing in comparison to the classical annealing with random initialization

				Critical Path			Min. Channels		
Nr.	Name	Size	Channels	Random	SOM	Relative	Random	SOM	Relative
1	e64	33 × 33	14	7.4 ns	5.9 ns	0.80	14	12	0.86
2	ex5p	33 × 33	22	10.4 ns	8.9 ns	0.86	20	22	1.10
3	apex4	36 × 36	22	10.1 ns	8.8 ns	0.87	22	20	0.91
4	misex3	38 × 38	18	8.8 ns	9.6 ns	1.09	18	16	0.89
5	alu4	40 × 40	16	11.0 ns	10.7 ns	0.97	16	16	1.00
6	seq	42 × 42	20	8.7 ns	8.2 ns	0.94	18	20	1.11
7	apex2	44 × 44	20	10.4 ns	10.4 ns	1.00	20	20	1.00
8	ex1010	68 × 68	18	17.1 ns	16.9 ns	0.99	18	16	0.89
Average						0,94			0,97

Table XI shows the results of placements for the formerly introduced MCNC benchmarks. Column two shows the name of the tested netlist, column three the size of the target architecture. The length of the critical path obtained by simulated annealing with random initialization – the reference – is shown in column five, the length of the critical path for simulated annealing with initialization by the self-organizing map in column six. Because the two approaches sometimes reach different minimal channel widths – as shown in column eight and nine – the larger channel width is used to determine the critical path's length, which is shown in column four. Column seven shows the length of the critical path produced by the initialization with the self-organizing map in relation to the reference (random initialization).

The benchmarks are arranged by ascending size. Statistically the results for the relative critical path's length for smaller netlists are better than those for bigger ones. This can be partially ascribed to the fact that in this series of measurements the same amount of training cycles was used for all netlists. The results for the *misex3* netlist are of special interest. Even though the proposed method needs a smaller channel width than the reference for the routing of this netlist, it is the only netlist for which the length of the critical path gets worse. On average, the critical path's length is reduced by six percent through the use of the self-organizing map.

The minimal channel width is also affected by the use of the self-organizing map. In four cases it is reduced by two and in three other cases it is increased by two. Note that the architecture demands an even channel width, so a change by two is in fact only one step. The relative results are shown only for the sake of completeness. On average, the minimal channel width is reduced by three percent through the use of the self-organizing map. Due to the small sample size and the formerly described distribution of the results the significance of this value is precarious.

V. Parallelization

The previous tests have shown that the sequential implementation of the self-organizing map is very slow, compared to simulated annealing. For this reason, the algorithm has been profiled to analyze the options to speedup its execution. In Table XII the profiling results of the unoptimized implementation of the self-organizing map for different sized netlists from the MCNC benchmark-set introduced above are summarized. It shows the relative computation times of the steps introduced above. Especially for larger netlists, the test and learning functions together consume almost all of the computation time. In this part of the work, the focus is on the test process because (with 73 percent on average) it consumes the highest amount of time. In the test process, the Euclidean distance between the stimulating vector and every neuron is determined. The neuron with the lowest distance is selected as winning neuron. The subfunction for the calculation of the distance consumes more than 99 percent of the test process (e.g., 368 seconds out of 369 seconds for the ex5p netlist). Based on these numbers, two levels of parallelism that could be exploited have been identified:

TABLE XII. Profiling results of the unoptimzed self-organizing map implementation

Netlist		FPGA	Relative Computation Time		
Name	Size	Size	Test	Learning	Others
net16	256	16 × 16	69.0 %	29.0 %	2.0 %
ex5p	1 064	33 × 33	73.9 %	25.8 %	0.3 %
ex1010	4 598	68 × 68	75.4 %	24.6 %	0.0 %
Average			72.8 %	26.5 %	0.7 %

1) **Vector-level:** The vector operation to determine the distance d between the stimulating vector $\vec{v}$ and a neuron's weight $\vec{w}$ as described in (6), assuming $\vec{v}$ and $\vec{w}$ have N elements.
2) **Map-level:** The calculation of all the distances and the selection of the lowest distance.

$$d = \sum_{i=0}^{N} (\vec{v}_i - \vec{w}_i)^2 \tag{6}$$

Implementations to exploit both these levels of parallelism have been developed and benchmarked. The corresponding results are presented in the following subsections.

A. Vector-level parallelization

In a first attempt, the parallelism of the vector operations was exploited to speedup the implementation of the self-organizing map. Therefore, two alternative, parallel implementations of the distance function used heavily in the test loop were created. One implementation is using the processor's *Streaming SIMD Extensions (SSE)* for vector operations, the other is delegating the vector operations to the GPU using OpenCL.

TABLE XIII. Time consumption of the parallel implementations of the distance function (6) for different vector sizes

	CPU	CPU SSE		GPU OpenCL	
Vector Size	Time	Time	Speedup	Time	Speedup
100 cells	27 µs	64 µs	0.4	170 µs	0.2
1 000 cells	200 µs	74 µs	2.7	300 µs	0.7
10 000 cells	2 000 µs	112 µs	17.9	400 µs	5.0
100 000 cells	23 ms	458 µs	50.2	454 µs	50.7
1 000 000 cells	238 ms	7 000 µs	34.0	669 µs	355.8

TABLE XIV. Configuration of the "desktop class" test-system

CPU	GPU
Intel® Core™2 Duo E8400	NVIDIA® GeForce® GTX 950
2 Cores	768 CUDA Cores
3 GHz Core Clock	1024 MHz Core Clock
6 MB Level 2 Cache	105.6 GB/s Memory Bandwidth
4 GB DDR2 RAM	2048 MB GDDR5 RAM

Table XIII shows the results of the parallel implementations for different vector sizes. The speedups are given as the ratios between the reference and the corresponding new approach as follows:

$$Speedup = \frac{Reference\ Time}{Benchmarked\ Time} \tag{7}$$

In this case these are the ratios between the calculation time of a sequential implementation on a CPU (reference time) and the parallel implementations on a CPU and GPU mentioned above (benchmarked times). For this benchmark a desktop computer with an "Intel® Core™2 Duo E8400" processor and a "NVIDIA® GeForce® GTX 950" GPU was used. The detailed configuration of the test-system is shown in Table XIV. In comparison to the unoptimized implementation, the SSE implementation breaks even between vector sizes of 100 and 1000 cells, whereas the GPU implementation breaks even between 1000 and 10000 cells. The SSE implementation and the GPU implementation break even at a vector size of 100000 cells. Even tough there are commercial FPGAs available with more than a million CLBs today, the netlists are typically partitioned to a smaller size before the placement and need much faster placement algorithms anyway. Further analysis has shown that the main problem of the tested OpenCL implementation lies in the low complexity of a single distance calculation. This causes a relatively large overhead for the memory transfer between host and GPU memory.

Based on these findings, an improved version of the prototypic, sequential implementation of the self-organizing map was created. It uses the CPUs SSE extensions for all vector operations. Table XV shows the computation times of both implementations of the self-organizing map for a subset of the netlists used in our previous work. The time is given for one training cycle, meaning the training of every vector. The overall speed of the training process was increased by a factor of up to 20. Especially the larger netlists benefit from the parallelization because the wider vectors give a better utilization of the SIMD-hardware. However, the simulated annealing algorithm of VPR is still about one hundred times faster than the proposed SSE implementation.

TABLE XV. Comparison of the computation times for one training cycle of the original implementation of the self-organizing map (SOM) and an improved version using SSE-accelerated vector operations

Netlist		FPGA	Computation Time		
Name	Size	Size	SOM CPU	SOM SSE	Speedup
net16	256	16 × 16	5 s	2 s	2.5
e64	273	33 × 33	23 s	7 s	3.3
ex5p	1 064	33 × 33	350 s	31 s	11.3
seq	1 750	42 × 42	1 476 s	95 s	15.5
ex1010	4 598	68 × 68	27 211 s	1 259 s	21.6

B. Map-level parallelization

To bridge this gap, the exploitation of map-level parallelism with OpenCL on a GPU is evaluated. The goal is to create bigger chunks of computational work and minimize the overhead for memory transfer between host and GPU. Ideally, the complete training loop takes place on the GPU, so that a memory transfer is only necessary after the vector generation and for the placement export. In Figure 11, a flowchart of the proposed implementation is presented. The management of the training data and the random selection of training vectors is still executed on the host CPU, but the rest of the training-cycle is executed on the GPU. Especially the comparatively large datastructure of the self-organizing map is kept in the GPU's memory over the complete training. The set of training vectors is kept in the GPU's memory as well, eliminating the need to copy the data from host- to device-memory for every training loop. This is achieved by transferring only the individual starting address of the vector to the kernel. Another approach could be to address the vectors by transferring an index. The computation on the GPU is done by three OpenCL kernels, which are described in the following:

1) Calculate Distances: The *Calculate Distances* kernel receives the map of weight vectors, the training vector and a map marking the already occupied positions by reference. The kernel is calculating the distance between every weight and the given training vector, storing the results in a two-dimensional map. It is rolled out in three dimensions, calculating each distance (6) in a workgroup (if large enough). Thereby, a fast workgrop-local buffer can be used to build the sum. If the vector size is larger than the workgroup-size, the partial sums of each workgroup are stored temporarily and summed up after synchronization. If the corresponding position of the kernel is marked as already occupied the distance is set to "MAXFLOAT", so that it will be ignored in the following reduction.

2) Find Lowest Distance: The *Find Lowest Distance* kernel receives the two-dimensional map of distances created by the *Calculate Distances* kernel by reference. It searches the map for the lowest distance and returns the corresponding position, as well as the distance. Again the reduction is done in workgroups, utilizing the fast local memory.

3) Learn Vector: The *Learn Vector* kernel – like the *Calculate Distances* kernel – receives the map of weight vectors, the training vector and a map marking the already occupied positions by reference. Additionally, it receives the position of the previously determined minimal distance weight (the winning position). The kernel modifies the weights in the map according to their distance to the winning position and marks

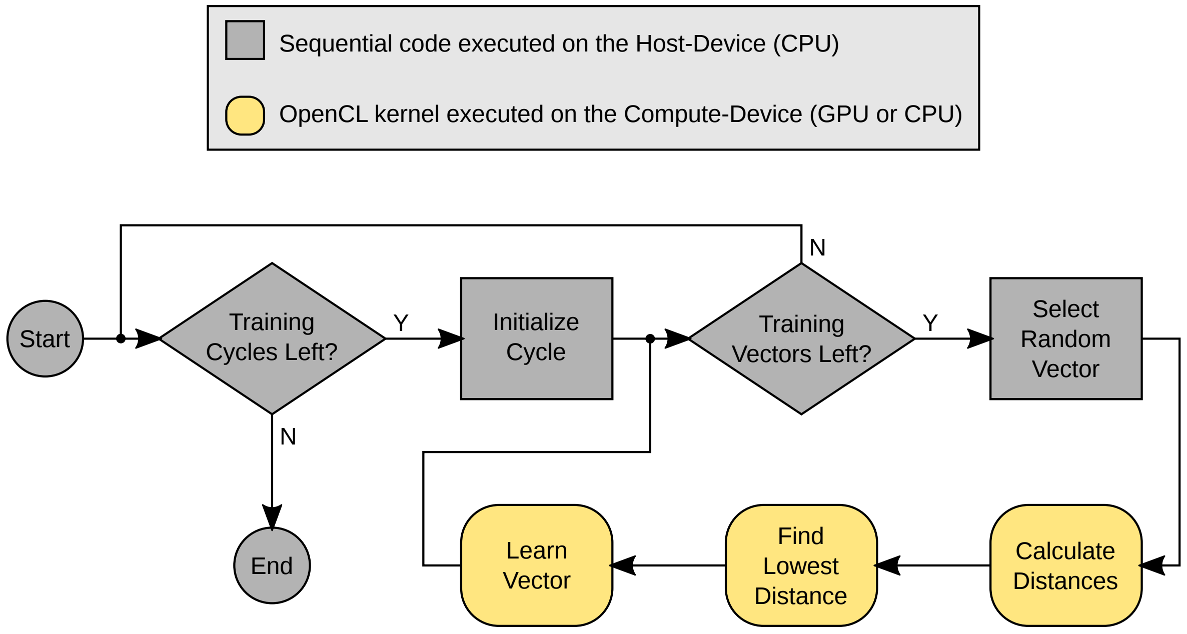

Figure 11. Flowchart of a training-cycle of the self-organizing map using OpenCL

the winning position itself as occupied in the corresponding map. There is no explicit grouping into workgroups and no local memory is used because all operations happen independently of each other and the results have to be stored in the global memory.

To benchmark this implementation two test-systems have been used. The first system is the "desktop class" computer that has already been used to benchmark the vector-level parallelization approach (see Table XIV). The second system is a "workstation class" system. Its detailed configuration is shown in Table XVI.

The benchmark results of the "desktop class" computer are presented in Table XVII. They compare the computation times of the high-level parallelization approach (GPU OpenCL) with the low-level approach (CPU SSE) introduced above. The durations are given for a complete placement-generation from reading the netlist over ten full training-cycles to writing the generated placement into a file. The speedup is given as the ratio between the two approaches (7). It is shown that even the placement of small netlists can be accelerated even further compared to the already improved SSE implementation. Netlists are placed up to 34 times faster on the GPU than on the CPU's SIMD-hardware. Generally, the speedup is higher for larger netlists. The only exception is the largest netlist in the benchmark (ex1010), which is experiencing the worst gain of all. This is because its vector size supersedes the maximal workgroup-size of the GPU and therefore a partially sequential reduction scheme is used. Compared to the original, sequential implementation the speedup is about 200 on average. For the seq netlist the speedup is even 310.

The benchmark results of the "workstation class" computer are presented in Table XVIII, it contains two additional (even larger) netlists. Furthermore, the ability of OpenCL to target not only GPUs, but also multicore CPUs has been evaluated. Obviously the modern workstation CPU is considerably faster than the comparatively older desktop CPU. However, the workstation GPUs performance is comparably weak on average, especially considering that its rated peak performance is more than three times higher than the peek performance of the desktop GPU (5.8 TFLOPS versus 1.7 TFLOPS). Additional tests with other algorithms have underpinned the assumption that the GPU is not unfolding its full potential in the workstation. It has to be evaluated if this is due to driver- or compatibility-problems. Also, the performance of the OpenCL code executed on the CPU is underwhelming. Even though it uses all eight cores of the CPU to full capacity, it is more than thirty times slower than the single threaded SSE implementation on the same hardware.

TABLE XVI. Configuration of the "workstation class" test-system

CPU	GPU
Intel® Xeon® E3-1245 v5	AMD® RADEON™ RX 480
4 Cores	36 Compute Units
8 Threads	2304 Stream Processors
3.5 GHz Core Clock	1120 MHz Core Clock
8 MB SmartCache	224 GB/s Memory Bandwidth
64 GB DDR4 RAM	8 GB GDDR5 RAM

TABLE XVII. Benchmark results of the high-level parallelization using OpenCL on a "desktop class" system described in Table XIV

Netlist	FPGA	CPU SSE	GPU OpenCL	
Name	Size	Time	Time	Speedup
e64	33 × 33	84 s	5 s	16
ex5p	34 × 34	341 s	23 s	15
apex4	36 × 36	449 s	28 s	16
misex3	38 × 38	570 s	33 s	17
alu4	40 × 40	820 s	38 s	22
seq	43 × 43	958 s	47 s	20
apex2	44 × 44	1 777 s	52 s	34
ex1010	68 × 68	9 100 s	993 s	9

TABLE XVIII. Benchmark results of the high-level parallelization using OpenCL on a "workstation class" system described in Table XVI

Netlist	FPGA	CPU SSE	GPU OpenCL		CPU OpenCL	
Name	Size	Time	Time	Speedup	Time	Speedup
e64	33 × 33	25 s	6 s	4.0	387 s	0.063
ex5p	34 × 34	121 s	27 s	4.5	4 208 s	0.029
apex4	36 × 36	150 s	32 s	4.7	5 160 s	0.029
misex3	38 × 38	193 s	54 s	3.6	6 895 s	0.028
alu4	40 × 40	297 s	58 s	5.1	7 662 s	0.039
seq	43 × 43	322 s	48 s	6,7	10 159 s	0.032
apex2	44 × 44	364 s	50 s	7.3	11 206 s	0.032
ex1010	68 × 68	3 269 s	313 s	10.4	251 654 s	0.013
s38417	81 × 81	8 086 s	554 s	14.6	513 873 s	0.016
clma	93 × 93	17 004 s	1 486 s	11.4	2 068 893 s	0.008

VI. Conclusion and Future Work

In this work, an approach to improve the results of netlist placement for FPGAs with a self-organizing map has been presented. Different methods to generate the training vectors have been compared based on synthetic benchmarks. For a set of 8 MCNC benchmarks it has been shown that the length of the critical path can be reduced by 6 percent on average. The cost is a high computational effort for the training of the self-organizing map.

To accelerate the self-organizing map, two parallelization approaches have been introduced and benchmarked. A low-level approach – exploiting the SSE units of the CPU – was shown to accelerate the self-organizing map up to twentyfold. It has been shown that the low-level approach is not suited to be executed on a GPU because the chunks of work are too small, resulting in a high overhead for memory transfer. For this reason a high-level parallelization approach – conveying the complete training loop to the GPU – has been introduced. It has been shown that it again accelerates the execution up to more than thirtyfold. On average, the OpenCL implementation on the GPU is about 200 times faster than the original sequential implementation. The results of OpenCL on a CPU are not satisfying.

In future work the speed of the accelerated self-organizing map should be compared directly to established placement tools. At this point it is expected that the self-organizing map is still perceptibly slower than for example VPR. If this poses a problem, the vector size can be reduced by using the I/O-distance metric for the vector generation. As has been covered by Amagasaki et al. [11], this should improve the speed while slightly reducing the quality.

References

[1] 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.

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

[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, no. 4598, May 1983, pp. 671–680.

[4] L. Ingber, "Adaptive simulated annealing (ASA): Lessons learned," Control and Cybernetics, vol. 25, no. 1, 1996, pp. 33–54.

[5] 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.

[6] J. Lam and J.-M. Delosme, "Performance of a new annealing schedule," in Design Automation Conference (DAC), 1988, pp. 306–311.

[7] P. Banerjee, S. Bhattacharjee, S. Sur-Kolay, S. Das, and S. C. Nandy, "Fast FPGA placement using space-filling curve," in International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2005, pp. 415–420.

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

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

[10] 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.

[11] M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "FPGA placement based on self-organizing maps," International Journal of Innovative Computing, Information and Control, vol. 11, no. 6, 2015, pp. 2001–2012.

[12] E. Chow, "A graph search heuristic for shortest distance paths," Lawrence Livermore National Laboratory, Tech. Rep., 2005.