Improving the Performance of a SOM-Based FPGA-Placement-Algorithm Using SIMD-Hardware

Timm Bostelmann and Sergei Sawitzki

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

***Abstract*—Programmable circuits and nowadays especially field-programmable gate arrays (FPGAs) are widely applied in demanding signal processing applications. In a previous work, we have introduced a method to improve the results of the netlist-placement for FPGAs with a self-organizing map (SOM). However, the presented algorithm conveys a comparably high computational effort. Considering modern, agile hardware / software codesign approaches, a slow design automation process can act as a kind of show-stopper, because software compilation is already distinctly faster. Thus, in this conceptual work, we present and evaluate different approaches to exploit the inherent parallelism of the SOM to increase the computation-speed. These approaches are based on using the single instruction multiple data (SIMD) capabilities of the central processing unit (CPU) and the graphics processing unit (GPU) for vector operations. Furthermore, we present benchmark results of an optimized implementation, based on using the CPU's SIMD units and introduce a concept for a GPU-accelerated implementation as work in progress.**

***Keywords*–FPGA; netlist placement; GPU computing; parallelization; SIMD.**

I. INTRODUCTION AND BACKGROUND

The ever-growing complexity of 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-complete placement process. It consists of selecting a resource cell (position) on the FPGA for every cell of the applications netlist. Many current solutions optimize the placement iteratively. The academic EDA toolchain *Verilog-to-Routing (VTR) Project for FPGAs* [1] for example utilizes the simulated annealing [2] algorithm to solve the placement problem. Roughly described it starts with a random initial placement and applies random changes iteratively. The key of simulated annealing is the gradual reduction of the probability to keep disadvantageous changes over the time. Thereby the algorithm is able to leave local optima in early stages as well as to provide a fine optimization in later stages.

In [3], we have proposed a method to improve the placement results for FPGAs. Therefore, we have used a special SOM [4] to generate an initial placement optimized by a low temperature simulated annealing schedule. The placement generation consists of three stages (Figure 1). Initially for every cell of the netlist a training vector is generated. To guarantee that highly connected cells are represented by similar vectors, we use a hyperbolic distance function. The SOM is trained with these vectors in a random order. It consists of a two-dimensional competition layer and a one-dimensional input layer. During the training the SOM clusters these vectors by similarity, so that the vectors of highly connected cells will cluster together on the competition layer of the map, thus reducing the prospective connection-lengths between the respective cells. Finally after the training has completed, the placement learned by the SOM is exported. Figure 2 shows the details of the training process. In the test stage, the neuron with the smallest Euclidean distance to the selected vector is determined. It is called the winning neuron. In the learning-stage the weights of the winning neuron and its neighbors are pulled towards the current stimulation. This makes them more susceptible to similar stimulations in the future and thereby induces the clustering of similar vectors. To prevent illegal placement results – which contain multiple occupations of a single resource cell – we temporarily block former winning neurons in the test process, so that they can not win again. The learning process is not affected by this measure. Consequentially the clusters of highly connected cells grow and displace other cells until they occupy the necessary space.

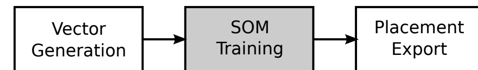

Figure 1. Flowchart of the SOM-based placement algorithm.

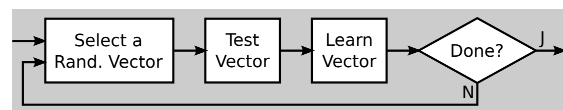

Figure 2. Flowchart of the internal SOM training.

In [3], we have compared this method (including the final low temperature simulated annealing) to the regular simulated annealing process of VPR [5]. We were able to reduce the clock-rate determining critical path by six percent in average, for a set of common benchmark netlists. However – as mentioned in our previous work – we did not consider the computation time yet, knowing that our prototypic "proof of concept" implementation was not competitive regarding speed. In fact the proposed method – and especially the training of the SOM – conveys a high amount of computational effort. Fortunately, it has also a high amount of inherent parallelism, which makes a parallel computation approach very promising.

The rest of this paper is organized as follows. In Section II, we analyze the necessary computation time of our original im-

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-496-1

TABLE I. Profiling results of the unoptimzed SOM implementation.

Netlist		FPGA	Relative Computation Time		
Name	Size	Size	Test	Learning	Others
net16	256	16 × 16	69.0 %	29.0 %	2.0 %
ex5p	1064	33 × 33	73.9 %	25.8 %	0.3 %
ex1010	4598	68 × 68	75.4 %	24.6 %	0.0 %
		Average	72.8 %	26.5 %	0.7 %

plementation, as basis for further optimization. In Section III, we present the results of an optimized implementation, which we have created based on our findings and give a prospect to further optimizations using GPU-computation with OpenCL [6]. Finally in Section IV, we summarize our findings and give a prospect to further work.

II. Parallelization

Table I summarizes profiling results of the unoptimized SOM implementation for different netlists from a Microelectronics Center of North Carolina (MCNC) benchmark-set [7]. It shows the relative computation times of the steps introduced above. Especially for larger netlists, the test and learning functions together consume almost 100 percent of the computation time. In this work, we focused on the test process because (with in average 73 percent) 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 (for example 368 seconds out of 369 seconds for the ex5p netlist). Based on these numbers, we have identified two levels of parallelism that could be exploited:

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

$$d = \sum_{i=0}^{N} (\vec{v}_i - \vec{w}_i)^2 \quad (1)$$

In a first attempt, we have optimized our SOM implementation by exploiting the parallelism of the vector operations. Therefore, we have created two alternative, parallel implementations of the distance function used heavily in the test loop. 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.

III. Results

Table II shows the results of the parallel implementations for different vector sizes. As this is only a preliminary test, we have used a desktop computer with an "Intel® Core™2 Duo E8400" processor. For a rough lower-bound approximation of the expectable GPU performance, we have used a "NVIDIA® GeForce® GTX 650" GPU. In comparison to the unoptimized implementation, the SSE implementation breaks even between vector sizes of 100 and 1000 cells, whereas

TABLE II. Time consumption of the parallel implementations of the distance function (1) 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
1000 cells	200 µs	74 µs	2.7	300 µs	0.7
10000 cells	2000 µs	112 µs	17.9	400 µs	5.0
100000 cells	23 ms	458 µs	50.2	454 µs	50.7
1000000 cells	238 ms	7000 µs	34.0	669 µs	355.8

TABLE III. Comparison of the computation times for one training cycle of our original SOM implementation 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	1064	33 × 33	350 s	31 s	11.3
seq	1750	42 × 42	1476 s	95 s	15.5
ex1010	4598	68 × 68	27211 s	1259 s	21.6

the GPU implementation brakes even between 1000 an 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 anyways. Further analysis has shown that the main problem of the tested OpenCL implementation lies in the small 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, we have improved our prototypic SOM implementation by using SSE for all vector operations. Table III shows the computation times of both SOM implementations 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. We have increased the overall speed of the training process 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 then our proposed SSE implementation. To bridge this gap, we propose to utilize the higher level of parallelism described above with an OpenCL implementation on a GPU. Thereby, we 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.

IV. Conclusion

We have presented an evaluation of different approaches to exploit the inherent parallelism of a SOM-based FPGA-placenment-algorithm to increase the computation-speed. These approaches are based on using the SIMD capabilities of the CPU and the GPU for vector operations. Furthermore, we have present benchmark results of an optimized SOM-implementation based on using the CPUs SIMD units and

outlined a concept for an even faster GPU-accelerated implementation. In the future, we are planing to evaluate the latest OpenCL to FPGA synthesis approaches [8]. Those are especially auspicious because of the flexible memory structure and extended inter-kernel communication.

Another promising lead has been published recently in [9]. The authors present a fast SOM-based FPGA-placement-algorithm, that is using the Shimbel Index [10] for the vector-generation and thereby reduces the vector-size distinctly. We will evaluate if the optimizations presented in this paper are fitting to speedup this new approach even further.

REFERENCES

[1] J. Rose et al., "The VTR project: Architecture and CAD for FPGAs from Verilog to routing," in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2012, pp. 77–86.

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

[3] 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.

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

[5] 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.

[6] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," IEEE Des. Test, vol. 12, no. 3, May 2010, pp. 66–73.

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

[8] T. S. Czajkowski et al., "From OpenCL to high-performance hardware on FPGAs," in International Conference on Field Programmable Logic and Applications (FPL), Aug 2012, pp. 531–534.

[9] 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.

[10] A. Shimbel, "Structural parameters of communication networks," The bulletin of mathematical biophysics, vol. 15, no. 4, 1953, pp. 501–507.

Copyright (c) IARIA, 2016. ISBN: 978-1-61208-496-1