# A New Partitioning Method for Parallel Simulation of VLSI Circuits on Transistor Level

Norbert Fröhlich and Volker Glöckel and Josef Fleischmann
Institute of Electronic Design Automation
Technical University of Munich
80333 Munich, Germany
{Norbert.Froehlich,Volker.Gloeckel,Josef.Fleischmann}@ei.tum.de

## Abstract

*Simulation is still one of the most important subtasks when designing a VLSI circuit. However, more and more elements on a chip increase simulation runtimes. Especially on transistor level with highly accurate element modelling, long simulation runtimes of typically several hours delay the design process. One possibility to reduce these runtimes is to divide the circuit into several partitions and to simulate the partitions in parallel. But the success of such a parallel simulation is heavily depending on the quality of the partitioning. This paper presents a new approach for partitioning VLSI circuits on transistor level and gives runtimes of parallel simulations of large industrial circuits. The resulting runtimes show considerable improvement compared to a known partitioning method, the Node Tearing method [10].*

## 1. Introduction

Integrated circuits belong to the most important products of today's industry, addressing a huge market. Therefore, the competitors make great efforts to produce the best chip with the lowest cost in the shortest possible time. All three objectives are addressed by high quality design tools. For achieving high quality, the tools perform more and more work on each design task.

One of the most time consuming design tasks is simulation on transistor level, but often these simulations are inevitable for design validation. Simulation runtimes increase both because of the permanent increase in number of transistors per chip and the shrinking dimensions inside a chip, which require more accurate and more complex modelling. Simulation of comparably small designs of about 20,000 transistors for a simulated time of about 200 nanoseconds lasts about 10 hours on a high performance single processor workstation.

One possibility to reduce simulation runtime is to split the circuit into several pieces and to simulate the pieces in parallel one piece per processor. Our cooperation partner Infineon Technologies, the former semiconductor division of Siemens, uses this approach and developed the inhouse tool TITAN [4] for parallel simulation on transistor level. This simulator is based on a parallel multilevel Newton method and shows excellent speedups for circuits which can be divided in fairly independent pieces. But in typical VLSI circuits the elements are highly connected and a partitioning usually shows a lot of connections between the partitions causing significant dependencies. For achieving relevant speedups, it is inevitable to find a partitioning with as few connections as possible between the partitions. Secondly, for equal usage of each processor the partitions should have about the same size.

At Infineon Technologies also the partitioning tool TIPART (TItan PARTitioning) [12] has been developed. This tool uses the *Node Tearing algorithm* [10], which was one of the first published algorithms for partitioning on transistor level.

Starting from an input voltage source, adjacent elements are gathered until a specified partition size is reached. Selection of the next adjacent element to gather is performed by a *contour table*. If there are several possibilities for the selection the node is taken, which causes less connections to other clusters.

Further approaches for partitioning on transistor level are some clustering algorithms as building *DC/Channel/Strongly Connected Components* [6, 11] or *Diagonal Dominance Norton Partitioning* [2]. Some *hierarchical methods* use clustering or exploit the subcircuit information contained in the circuit description as first step and run classical *Fiduccia-Mattheyses* [3] like methods for postprocessing [1, 7, 9]. Most of these approaches are only applicable to MOS-technology circuits, others are highly dependent on the simulation tool they have been developed for.

Another approach simply splits the ASCII file containing the circuit description and improves this initial partitioning by shifting components [8]. The parallel simulation speedup for this simple partitioning is only 9 for 49 processors on a distributed memory message passing parallel computer.

The work presented in the following is focused on a new partitioning method for the parallel simulation tool TITAN. After a special circuit preparation for respecting the constraints of the simulation tool, elements are clustered by evaluating the level of coupling between adjacent elements. The algorithm is probabilistic, constructive, and very fast. As it outperforms the other partitioning method TIPART developed for TITAN, it is now used in combination with the parallel TITAN, which is applied intensely in industry, because of its excellent performance. It proved to speedup the simulation by a factor close to the ideal speedup of 8 for 8 processors applied on fairly independent circuits [4] and also proved to be able to simulate very large circuits of up to 3 million transistors on transistor level.

The rest of the paper gives a detailed description of the new partitioning method COPART (COuple PARTitiong) [5] and shows simulation results for large state-of-the-art industrial circuits.

## 2. Partitioning for Parallel Simulation

### 2.1. Partitioning Objectives

The main objective of partitioning for parallel simulation is to reduce the parallel simulation runtime. For low simulation runtimes it is crucial to achieve a **low number of signals connecting the partitions**: Firstly, each connection signal causes time consuming communication. Secondly, the parallel simulation by TITAN [4] is synchronized by a master process, which calculates the connection network while the parallel simulation on the slave processors is stopped. The connection network calculation is very time critical and increases cubically with each additional connecting signal.

Besides a low number of connecting signals the second main issue for partitioning is an equal workload for each slave processor, i.e. simulation effort should be distributed optimally. To be able to estimate workload, each element is assigned a weight according to the computational effort needed for its simulation. Thus, similar workload for each slave processor is given, if the the **element weight sums are similar for all partitions**.

### 2.2. Preparing the Circuit for Partitioning

Input for TITAN simulations is a circuit description in a SPICE-like format. Before partitioning this description has to be parsed and the structure of the circuit to be extracted. The parallel simulation requires some constraints: E.g., no paths are allowed from a pin of a partition to ground, which contain only voltage sources and inductors, as this would lead to unsolvable equations. Also controlling elements of controlled sources have to be packed into the same partition, because the controlled sources need information about the status of the controlling elements. Some flexibility for selection of the partitioning algorithm is achieved by mapping the structure of the circuit to an undirected graph, where the nodes represent the circuit elements and the edges are the signals connecting them. The parallel simulation constraints are met by a special packing of critical elements into a single graph node [4]. Thus, the partitioning operates on an undirected graph.

As described in [4] we used a graph partitioning tool originally developed for layout synthesis. The achieved partitioning quality outperformed TIPART in terms of fewer signals between the partitions and in lower simulation runtimes, but the partitioning itself took about 4 hours for a 40,000 transistor circuit design, whereas TIPART partitioning lasted just a few minutes. To reduce partitioning runtimes, COPART has been developed which also has runtimes of just a few minutes. Partitioning results showed even fewer connecting signals as the layout synthesis partitioning tool [5]. As COPART is faster by magnitudes and achieves better results, we dismissed the layout synthesis partitioning tool and concentrated on COPART.

Another big advantage of COPART is its much smaller program code, which makes it easier to fine-tune the partitioning algorithm to the application. Experience shows us, that a good partitioning quality can only be achieved by an application specific tool.

## 3. The Partitioning Method COPART

### 3.1. Outline of COPART

COPART creates clusters by merging adjacent nodes. For node merging an edge of the graph is selected from all edges following a suitable objective (Section 3.5). The two nodes connected to the selected edge are merged into one node and the edge is absorbed (Section 3.4). Continuously selecting edges, merging nodes and absorbing edges decreases the number of nodes until it equals the number of desired partitions. Each grown node represents a part of the circuit and at the end a partition.

Most clustering approaches operate locally, whereas in COPART the selection of the next edge is always performed regarding all edges of the graph. This ensures the global view of the algorithm.

### 3.2. Modelling the Circuit

In related publications circuits are often modelled by a hypergraph $H = (V_e, E')$ with nodes $V_e$ representing the

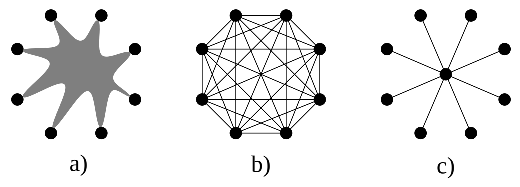


**Figure 1. Signal Representations**

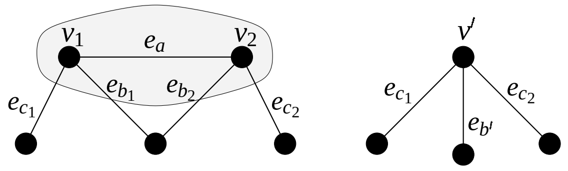


**Figure 2. Node merging**

elements and hyperedges $E'$ (Figure 1a) representing signals. Hyperedges connecting more than two nodes can be mapped into a set of binary edges, i.e. a clique model (Figure 1b), or can be modelled by additional nodes, the signal nodes $V_s$, with binary edges to each connected element (Figure 1c). COPART's good performance is heavily based on following the new approach of using both methods simultaneously for transforming the hypergraph to the undirected graph $G = (V, E)$ with $V = V_e \cup V_s$. Large signals connected to a lot of elements are widely distributed over the circuit, i.e. these are very likely to be cut. Therefore, a partitioning tool should not take much effort on large signals, but focus on small signals to avoid cutting them. In COPART, signals with few connected elements are represented by a clique model and thus become highly connected parts of the graph, which reduces the possibility for these to be cut. Large signals are modelled by additional signal nodes resulting in a weak connection structure, which favors cutting them. The tradeoff value between clique or signal node modelling is an optimization parameter for COPART. Our experiments showed good results for a tradeoff value around nine elements connected to a signal.

If two nodes are connected by several edges, these edges are merged into one edge with a higher weight.

## 3.3. Node and Edge Weights

The graph has assigned weights $w_V$ for the nodes and weight $w_E$ for the edges. $w_V$ represents the computational effort needed to simulate the element represented by the node. Signal nodes represent no elements, therefore $w_V = 0$ for $v \in V_s$. The size of a partition $P_i$ in a partitioning into k partitions $P^k = \{P_1, P_2, \ldots, P_k\}$ is determined by the sum of its node weights $size(P_i) = \sum_{v_i \in P_i} w_{V_i}$.

Edges of signals connecting $r$ elements are weighted with $w_E = \frac{1}{r}$. Both the modelling and the weight assignment cause lower significance for larger signals.

Signals connected to grounded voltage sources have a given potential all over the simulation run. Thus, the parallel simulator puts a grounded voltage source in each partition connected to this source, i.e. it causes no extra simulation cost if signals connected to grounded voltage sources are cut. The zero cut cost of these edges is considered by assigning the weight $w_E = 0$.

The edge weights are stored in an $(n_V \times n_V)$ adjacency matrix $\mathbf{A} = [w_{E_{ij}}]$, where $n_V = |V|$ and $w_{E_{ij}}$ is the weight between node $i$ and $j$. $\mathbf{A}$ is huge but symmetric and extremely sparse, as one node is typically connected only to about five other nodes for graphs of VLSI circuits described on transistor level. Therefore, an efficient matrix representation is possible.

## 3.4. Node Merging

As each edge is only connected to two nodes, merging is always performed on two nodes. Merging the nodes $v_1$ and $v_2$ in Figure 2 yields the new node $v'$ with the weight $w_{V'} = w_{V_1} + w_{V_2}$. For considering the edges, three cases have to be distinguished:

- Nodes to be merged are selected by their connecting edge $e_a$. This edge becomes an inner edge of $v'$ and is removed from the graph.
- The edges $e_{b_1}$ and $e_{b_2}$ are connected to a common neighbor of $v_1$ and $v_2$. These are combined to one single edge $e_{b'}$ with weight $w_{E_{b'}} = w_{E_{b_1}} + w_{E_{b_2}}$. The weights are updated by row and column additions in the adjacency matrix.
- Edges $e_{c_1}$ and $e_{c_2}$ become edges of $v'$ and keep their weights.

## 3.5. Coupling Measure

A clustering algorithm selecting the next nodes to merge only by the maximum edge weight favors the growth of a single big cluster as shown in Figure 3a, where a graph should be partitioned into three partitions.

To avoid this, a size limitation for the clusters is introduced in Figure 3b. Now the strongest connections are exploited to grow one cluster until the limit is reached, as shown by the dark grey clusters in Figure 3b. This is repeated with several clusters, but usually there are more clusters left than the desired $k$ partitions as in the third picture of Figure 3b. The remaining clusters are combined in a postprocessing step, yielding partitions weakly connected inside. Not using the maximum edge weight, but a coupling measure enforces the clusters to grow more simultaneously causing to reach the size limit later as shown in Figure 3c. This yields clusters with lots of inner connections and, therefore, the partitioning yields few cut edges.

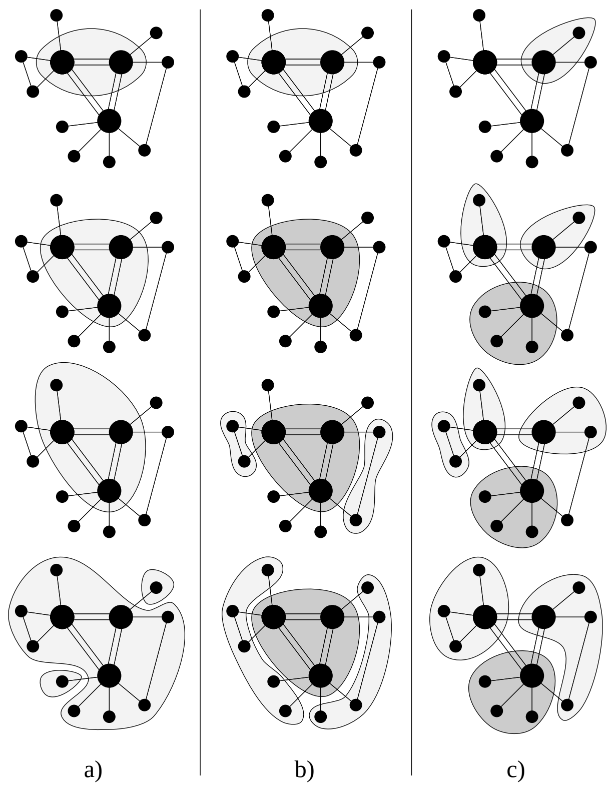


Figure 3. Growing of Clusters

If there are still more clusters than $k$, these are combined in a postprocessing step, which is described in Section 3.7.

The new coupling measure $c_{ij}$ of COPART is the edge weight between node $i$ and $j$ divided by the minimal sum of the edge weights of both nodes connected to the edge.

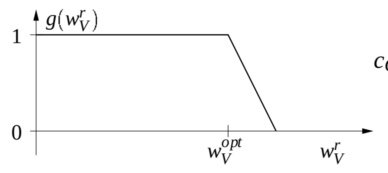
$$c_{ij} = \frac{w_{E_{ij}}}{\min\left(\sum_{l=1,l\neq i}^{n_V} w_{E_{il}},\ \sum_{l=1,l\neq j}^{n_V} w_{E_{jl}}\right)}$$

The advantage of this new measure is that nodes connected to only a few edges get a high $c_{ij}$ and are preferably combined. Thus, a situation as shown in the third picture of Figure 3b is avoided, where remaining parts with no direct connection have to be grouped together. After postprocessing, these unconnected parts would cause most likely poor partitions with many cut signals.

Especially when the algorithm starts, there are usually several edges with the same maximal coupling measures. From these values one is randomly selected, causing COPART to be a probabilistic algorithm. The randomness is reasonable reduced by the new signal modelling, which reduces significantly the number of equal maximal coupling measures.

## 3.6. Limitation of Cluster Size

To achieve equal sized partitions, a size limit for cluster growth is necessary. An optimal solution would be to control cluster growth in a way, that the size limit is first reached near the end of the clustering and thus exactly $k$ clusters remain after clustering.



$$c_{G_{ij}} = c_{ij} * g(w_V^r)$$

Figure 4. Ramp Function

Extensive experimental investigations showed poor performance for restricting cluster growth below the desired cluster size in any way. The best result has been achieved by multiplying the coupling measure with the ramp function $g(w_V^r)$ of Figure 4 yielding $c_{G_{ij}}$, where $w_V^r$ would be the resulting node weight after merging and $w_V^{opt}$ is the optimal cluster size.

Thus, COPART begins restricting cluster growth smoothly above the optimal cluster size and increases restriction up to a hard limit, which can be selected by the user. Typically, a 10 % ramp is reasonable.

## 3.7. Assigning Clusters to Partitions

After clustering by node merging there may be more clusters than the desired $k$ partitions. The reasons for that are two restrictions while clustering: First, only connected nodes can be merged, and second, a merging is rejected, if the resulting node would violate the size limit. Thus, a deadlock situation arises, if two neighboring nodes are too big for merging, or if two nodes small enough for merging are not directly connected. In a postprocessing step, the remaining clusters are combined regardless of their connections. The clusters are first ordered by size and second the cluster with the $k'th$ largest size is combined with the $(k+1)'th$ largest one. Both steps are repeated until there are only $k$ clusters, i.e. partitions, left.

As this postprocessing takes only the cluster weights into account and ignores the connecting signals, it can produce a suboptimal solution. But nevertheless, this fast algorithm does the cluster assignment to partitions quite well.

## 3.8. Complexity

For complexity considerations, the worst case would be a fully meshed graph. Then merging two nodes causes $(n_V - l) - 2$ edge weight additions in the adjacency matrix **A** (Figure 5), where $n_V = |V|$ is the number of nodes and $l$

| circuit | industry 1 | | industry 2 | | industry 3 | | industry 4 | |
|---|---|---|---|---|---|---|---|---|
| number of MOSFET's | 13852 | | 14413 | | 19995 | | 34869 | |
| **1 processor** | | | | | | | | |
| simulation runtime | 2:13:36 | | 4:19:17 | | 7:51:34 | | 14:47:39 | |
| partitioning tool | TiPart | CoPart | TiPart | CoPart | TiPart | CoPart | TiPart | CoPart |
| **4 processors** | | | | | | | | |
| partitioning runtime | 0:04:18 | 0:01:36 | 0:05:57 | 0:04:57 | 0:10:39 | 0:11:17 | 0:07:39 | 0:03:39 |
| number of cut signals | 275 | 257 | 237 | 156 | 203 | 112 | 382 | 241 |
| weight balance (%) | 7.3 | 4.8 | 10.9 | 4.7 | 3.8 | 4.6 | 4.3 | 4.1 |
| simulation runtime | 0:41:13 | 0:33:29 | 3:00:03 | 2:20:46 | 4:25:48 | 3:54:41 | 7:15:44 | 6:00:47 |
| speedup | 3.24 | 3.99 | 1.44 | 1.84 | 1.77 | 2.00 | 2.03 | 2.46 |
| **8 processors** | | | | | | | | |
| partitioning runtime | 0:06:27 | 0:01:31 | 0:05:51 | 0:04:42 | 0:08:24 | 0:10:54 | 0:07:27 | 0:03:37 |
| number of cut signals | 314 | 284 | 415 | 177 | 343 | 193 | 455 | 255 |
| weight balance (%) | 10.6 | 2.7 | 19.1 | 9.7 | 25.3 | 14.6 | 6.3 | 4.2 |
| simulation runtime | 0:24:05 | 0:25:38 | 1:48:57 | 1:27:21 | 2:37:47 | 2:13:53 | 5:24:49 | 3:44:23 |
| speedup | 5.96 | 5.21 | 2.37 | 2.96 | 2.98 | 3.52 | 2.73 | 3.95 |

**Table 1. Partitioning and simulation results (times in h:m:s)**

is the number of already performed merges. **A** is symmetric and the diagonal elements are not needed. The total number of operations in **A** is

$$\sum_{l=0}^{n_V-k-1} ((n_V - l) - 2) = \frac{1}{2}(n_V{}^2 - 3n_V - k^2 + 3k).$$

The same number of operations is necessary to compute the coupling measures. Thus, the complexity for all the edge weight operations is $\mathbf{O}(n_V^2)$, assuming that the number of partitions $k \ll n_V$.

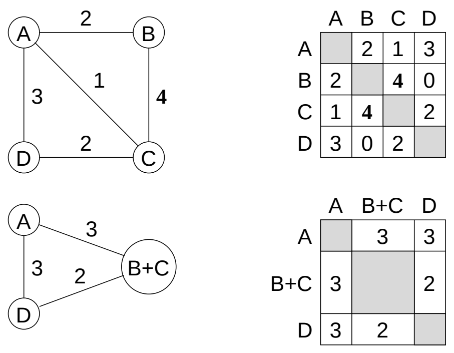


**Figure 5. Matrix Additions**

As the number of connections of a single element is independent from the circuit size, the adjacency matrix of a circuit structure on transistor level is sparse. For the initial circuit graph a node usually represents one circuit element and on average there are five connections to a node, i.e. five matrix entries per row. This number increases during clustering, but usually remains under 15. Let $b$ be the average number of entries per row while clustering. Now the number of additions per merge is $2b-2$. There are $n_V - k$ node merges and the number of additions in **A** is equal to the number of coupling measure calculations. Thus, the real total number of all the edge weight operations is $2(2b-2)(n_V - k)$, yielding a complexity $\mathbf{O}(b\,n_V)$.

The coupling measures are stored sorted in a *priority queue* with a complexity for changing or deleting an entry of $\mathbf{O}(\mathrm{ld}\ m)$, if there are $m$ entries. At each $n_V - k$ node merge $2b-2$ changes in the *priority queue* are performed, which has $b\,n_V$ entries. There are $(2b-2)(n_V-k)\mathrm{ld}(b\,n_V)$ operations for managing the coupling measures, which is a complexity of $\mathbf{O}(b\,n_V \mathrm{ld}(b\,n_V))$.

For large circuits algorithm runtime is dominated by the coupling measure handling. Experimental results proved an overall complexity for CoPart of $\mathbf{O}(b\,n_V \mathrm{ld}(b\,n_V))$.

## 4. Experimental Results

The performance of our new partitioning approach on transistor level has been evaluated on several large VLSI circuits. Table 1 shows partitioning and simulation results of four industrial circuits. The second row gives the number of MOSFET's of each circuit, which characterizes their size. The circuits are all critical parts of larger CMOS designs. *Industry1* is a part of a Dual Port RAM. *Industry2*, *Industry3* and *Industry4* are parts of a 16 and a 256 Mb DRAM design.

TiPart is implemented in C, CoPart in C++. All partitioning runs and simulations are performed on a R10000/195Mhz SGI Power Challenge with 12 CPU's.

In Table 1 results of CoPart are compared against TiPart. It shows simulation runtimes for a single processor simulation and for parallel simulations using 4 and 8 processors. The resulting speedups are also presented. For the partitioning the runtimes for the partitioning itself, the number of cut signals and a measure for the equality of the partition weights are given. The weight balance is the relative error in size of the largest partition to the optimal partition size.

The results show that COPART produces fewer cut signals, which result in significantly lower simulation runtimes especially for the largest circuit. As the connection network grows cubically with each additional cut signal, the advantage of fewer cut signals for a shorter simulation runtimes increases for larger circuits. Balance of partition weights is no problem for both tools. The runtimes for the partitioning are very low in contrast to the simulation runtimes. They seem to be comparable for both tools, but there are some advantages for COPART. As described in Section 3.5, COPART is a probabilistic method. Experience showed, that there is a high probability to have at least one very good partitioning result within five runs. The given runtimes include these five runs. TIPART is not probabilistic, but to achieve the requested number of partitions with similar size, the algorithm has to be run about three to five times while varying some parameters. The presented runtimes include three runs. Additionally, TIPART exploits the hierarchy information given in the circuit description. It builds clusters containing subcircuits up to a certain size, which is assigned by a partitioning parameter, whereas COPART always works on a circuit with all subcircuits flattened out.

We have conducted experiments with circuits generated from the layout mask, which contain no hierarchy information at all. Runtimes for a COPART partitioning have been about half an hour for an 150,000 elements circuits, whereas TIPART partitioning lasted about 24 hours. Number of cut signals have been about 2,000 by COPART against about 4,000 by TIPART. We cannot present simulation runtimes for these circuits, as the 1.5 GB RAM of our parallel machine has not been sufficient to simulate these circuits. Soon a bigger parallel machine will be available for us. Then we will be able to present a full simulation data set for these circuits.

## 5. Conclusion

In this paper a new approach for partitioning VLSI circuits on transistor level has been presented. It is based on a clustering algorithm with a new signal model concept and a special coupling measure. This enables a very efficient cluster growth. Selection of the next cluster merge by considering the whole circuit keeps the global view. Experimental results applying the new algorithm COPART on state-of-the-art industrial circuits show a significant reduction of parallel simulation runtimes on transistor level compared to another partitioning method [12] based on *Node Tearing*. Thus, the new partitioning method replaces the *Node Tearing* partitioning tool, which is actually used in the current design flow at Infineon Technologies combined with the excellent transistor level parallel simulation tool TITAN.

## 6. Acknowledgement

The authors would like to thank Georg Denk and the TITAN group from Infineon Technologies for providing TITAN and large industrial circuits. With their support, we were able to apply our partitioning techniques to current industrial problems.

## References

[1] P. Cox, R. Burch, and B. Epler. Circuit partitioning for parallel processing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 186–189, 1986.

[2] P. Debefve, F. Odeh, and A. Ruehli. Waveform techniques. In A. Ruehli, editor, *Circuit Analysis, Simulation and Design, Part 2*, volume 3 of *Advances in CAD for VLSI*, chapter 8, pages 41–127. North-Holland, 1985.

[3] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *ACM/IEEE Design Automation Conference (DAC)*, volume 19, pages 175–181, 1982.

[4] N. Fröhlich, B. M. Riess, U. A. Wever, and Q. Zheng. A new approach for parallel simulation of vlsi circuits on a transistor level. *IEEE Transactions on Circuits and Systems CAS*, 45(6):601–613, June 1998.

[5] V. Glöckel. Effizientes Partitionieren digitaler Schaltungen auf Transistorebene. Master's thesis, Lehrstuhl für Rechnergestütztes Entwerfen, Technische Universität München, November 1996.

[6] U. Hübner, H. Vierhaus, and R. Camposano. Partitioning and analysis of static digital cmos circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(11):1292 – 1310, Nov. 1997.

[7] W. John, W. Rissiek, and K. Paap. Circuit partitioning for waveform relaxation. In *European Conference on Design Automation (EDAC)*, pages 149–153, Amsterdam, Feb. 1991.

[8] T. Kage, F. Kawafuji, and J. Niitsuma. A circuit partitioning approach for parallel circuit simulation. *IEICE Transactions on Fundamentals*, E77-A(3):461–466, 1994.

[9] H. Onozuka, M. Kanoh, C. Mizuta, T. Nakata, and N. Tanabe. Development of parallelism for circuit simulation by tearing. In *European Conference on Design Automation (EDAC)*, pages 12 – 17, 1993.

[10] A. Sangiovanni-Vincentelli, L.-K. Chen, and L. O. Chua. An efficient heuristic cluster algorithm for tearing large-scale networks. *IEEE Transactions on Circuits and Systems CAS*, CAS-24(12):709–717, Dec. 1977.

[11] A. V. Vasquez, S. W. Director, and K. A. Sakallah. Primo: A vlsi circuit partitioner for simulation applications. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1075 – 1078, 1985.

[12] O. Wallat. Partitionierung und Simulation elektrischer Netzwerke mit einem parallelen mehrstufigen Newton-Verfahren. PhD thesis, Universität Hamburg, 1998.