PALACE: A Parallel and Hierarchical Layout Analyzer and Circuit Extractor

F. Scherber and E. Barke

Department of Electrical Engineering
University of Hanover
Hanover, Germany

W. Meier

Corporate Research and Development
Siemens AG
Munich, Germany

Abstract

Layout verification of VLSI circuits can be speeded up significantly by parallel execution. The approach described in this paper combines parallel and hierarchical verification of cells ***and*** *cell areas using geometrical partitioning. In contrast to earlier approaches, design rule check and netlist extraction are performed in parallel without any functional restriction. This is accomplished by a new concept called* ***multiple execution switching****. Thus, industrial leading edge VLSI circuits can be handled. High speedups are obtained for large real-world layouts. A productive use is possible and will reduce time-to-market considerably.*

1 Introduction

Complete layout verification of today's leading edge VLSI circuits is only possible by exploiting the inherent hierarchy and regularity of a chip design [1]. Using hierarchical design rule checking and netlist extraction decreases computation time and data volume substantially. In addition, the number of reported errors is reduced significantly. Design rule violations found in a specific cell are reported only once, independently from the number of repetitions of the cell in the whole layout. Moreover, a hierarchically extracted netlist is a fundamental requirement for applying one of the very fast *hierarchical* netlist consistency checking tools [2].

According to growing circuit complexity, computation time for verification will further increase unless innovative solutions will be applied. Since designing a VLSI circuit is an iterative process, verification time is directly time-to-market relevant. One possible solution is to take advantage of the locality of verification. Instead of using one workstation to execute verification sequentially, it can be performed in parallel on a set of processors. There are four strategies for parallel verification that can be applied separately or in combination:

- Edge-based parallelization: Parallel handling of different polygon edges.
- Cell-based parallelization: Parallel verification of independent layout cells.
- Area-based parallelization: Parallel verification of layout areas using geometrical partitioning.
- Operation-based parallelization: Parallel execution of independent verification operations.

Because of its high degree of parallelism and intensive interprocess communication, the first – fine-grained – approach is only applicable on massively parallel systems. This may speedup verification by up to two orders of magnitude [3]. However, high investment costs, the expense for a complete reimplementation of the verification software and limited utilization of the machine for other computational work make this solution rather unattractive.

The other – coarse-grained – approaches seem to be much more reasonable, because they can utilize multiprocessors [4, 5, 6, 7, 8] or even general-purpose workstation clusters [9]. Since workstation clusters usually are at hand and not very busy, investment costs are low. However, using these clusters effectively makes it necessary to combine parallelization strategies mentioned above. Especially when using loosely coupled workstations, in many cases interconnected by Ethernet, interprocess communication has to be minimized to obtain sufficient speedup.

This paper describes the theoretical framework and implementation of PALACE, a **Pa**rallel and hierarchical **L**ayout **A**nalyzer and **C**ircuit **E**xtractor. PALACE is able to utilize almost all kinds of present general-purpose workstations. These workstations may be only loosely coupled by a local area network without significant performance drawbacks. PALACE executes verification tasks hierarchically. In this way, design rule violations are reported without repetition in every cell instance as described above. In addition, PALACE may be combined with other hierarchical tools e. g. hierarchical netlist consistency and electrical rule checking programs.

In contrast to earlier proposals for parallel verification, PALACE performs a complete layout analysis and circuit extraction. Complete refers to the absence of functional restrictions, as for example the disability of executing selection operations or extracting bipolar transistors, or even the limitation to Manhattan layouts [8, page 344]. For this purpose, the concept of *multiple execution switching* has been applied to parallel layout verification. In addition, a *wrapper* concept is used for implementation, which preserves all features of the proven sequential verification algorithms and their ability of handling all-angle geometry. In this way, real-world layouts of industrial leading edge circuits can be verified.

ED&TC '96
0-89791-821/96 $5.00

This paper is structured as follows. Sections 2 and 3 discuss parallelization strategies of PALACE regarding special problems concerning non-local operations and utilization of workstation clusters. Section 4 describes techniques used in the implementation. Sections 5 and 6 discuss some results and the areas of our further research.

2 Cell-Based Parallelization

The most natural strategy for parallel hierarchical verification is cell-based parallelization. Sequential hierarchical verification verifies the chip layout represented by the cell tree in a bottom-up order cell by cell (Figure 1 a). A cell-based parallel hierarchical verifier may handle all cells of one hierarchy level in parallel (Figure 1 b).

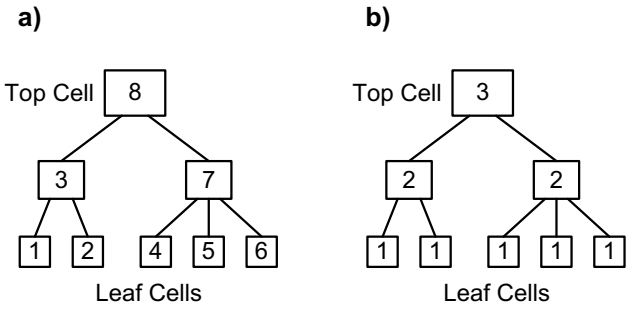

Figure 1: Sequential (a) and cell-based parallel verification (b) – Numbers correspond to the order of execution

The strategy of cell-based parallel verification offers some important features:

First, there is no data dependency between cells on the same hierarchy level. Therefore, there is no need for interprocess communication or synchronization except one "order"-message and one "ready"-message for each cell. These messages contain only a small amount of data.

Second, a cell-based parallel verifier can be implemented by a simple modification of the sequential program. One master process with knowledge about the cell tree triggers the verification of a cell by passing an "order"-message to an appropriate slave process. Sequential verification of the cell is performed by the slave process. After completion the slave process returns a "ready"-message. In this manner the complete cell tree is verified bottom-up from the leaf cells to the top cell.

However, there is one important drawback of cell-based parallelization. As can be derived from Figure 1, the degree of parallelism decreases considerably on higher levels of hierarchy. Moreover, cells on higher levels of hierarchy often consume most of the computation time. For example, if the top cell consumes half of the sequential runtime, speedup is less than two, independently of the number of available processors. In Section 5 this fact is illustrated by some runtime measurements.

To overcome this problem, the basic approach has to be extended by at least one of the other parallelization strategies mentioned in Section 1. Area-based parallelization of cells discussed in the next section is an efficient strategy.

3 Area-Based Parallelization

Area-based parallelization utilizes the locality of verification. The area of a cell is geometrically partitioned into tiles that hereafter are verified in parallel. The tile sizes can be chosen in two ways:

- Tiles of same size are simple to compute but may lead to load balancing problems if the density of layout polygons varies. This may cause a considerable speedup decrease.
- Tile sizes can be chosen in a computation time dependent way. Considering the geometrical distribution of layout polygons, computation time for a certain region can be estimated and a partition may be determined which distributes work uniformly over the involved processors. Developing such estimation algorithms will be an item of our further research.

A lot of false DRC-errors may be introduced by geometrical partitioning. To avoid this, design rule check is performed on an extended region called the *verification window* which is larger than the corresponding tile (Figure 2). The verification window belonging to a specific tile is determined by expanding the tile by the maximum design rule interaction distance. In case of netlist extraction, an expansion by one grid unit is sufficient for merging of broken polygons, devices and electrical connections.

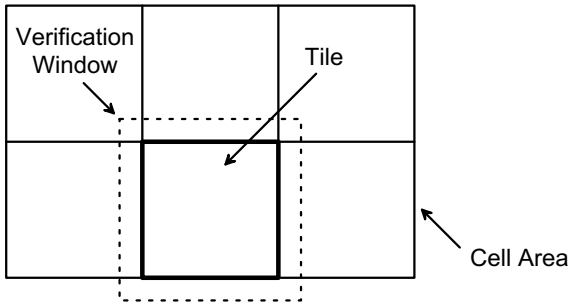

Figure 2: Verification window vs. tile

After completing parallel processing, a merging step has to be executed to combine the tile results (e. g. completing global connectivity information) and to correct deficiencies (e. g. merging of broken devices, avoidance of design rule violations reported several times).

There are two main advantages of area-based parallelization:

First, the number of tiles is limited only by parallelization overhead. This overhead is caused by the overlap of the verification windows described above and by the merging step. If it is possible to perform the merging step in a short time, area-based parallelization offers a considerable potential for high speedups.

Second, the computational complexity of layout verification is higher than linear. For example, let n be the number of polygon edges in the layout. Then, the computational complexity of the scanline algorithm that is used for various verification steps is $O(n \log n)$ in best case.

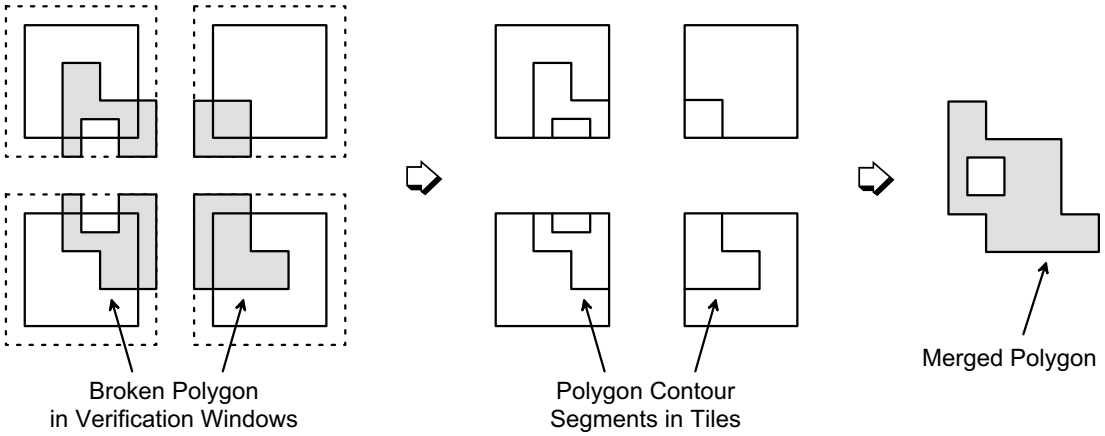

Figure 3: Merging of a broken polygon by connecting contour segments

Therefore, due to the linearizing effect of partitioning the reduced number of polygon edges handled by one processor decreases the *totally* consumed computation time of verification.

Area-based parallelization utilizes the locality of the executed operations. For example, design rule checks have a maximum interaction distance that is small compared to the size of a tile. Also, the presence of a – possibly broken – MOS-transistor is determined by a boolean AND of the diffusion, the polysilicon and the implant layer. This operation can be performed locally.

However, there are some non-local operations. Examples are selection operations or the extraction of bipolar transistors. In earlier approaches for area-based parallel verification such non-local operations have not been considered. Only boolean operations, design rule checking, connectivity analysis and extraction of MOS-transistors have been supported.

For PALACE, a new concept has been developed that allows all kinds of operations. It is based on the fact, that the non-local operations which have to be executed sequentially often consume only a small amount of computation time. Therefore, effective parallel verification may be performed by the following algorithm:

1. Resort the order of the operations considering mutual dependencies to build maximum instruction chains of either local or non-local operations.
2. Execute an area-based parallelizable instruction chain without any synchronization or communication between the involved processors.
3. Synchronize the involved processors and merge broken polygons.
4. Execute non-local operations on a single processor.
5. Repeat steps 2 to 4 until all operations are executed.
6. Merge tile results, correct deficiencies and return "ready"-message to the master.

This concept, called *multiple execution switching*, may cause several merging steps. To avoid large parallelization overhead when using a workstation cluster we developed a partition and merging concept with following advantage:

Local operations are applied to *all* polygons that overlap the verification window. Even devices that belong to more than one window are extracted in parallel by the corresponding tile processors. Resulting "half" devices and all related connectivity informations are automatically combined by the fast and simple polygon merging step described below. In this way, no parallelizable work is done by a single processor and the amount of sequentially executed work is minimized.

The polygon merging algorithm works as follows (see Figure 3):

First, polygons that overlap at least one edge of their tile are selected in parallel by the tile processors. References of these polygons, uniquely numbered over all tiles, are collected by the merge processor.

Second, the broken polygons are merged by this processor by connecting contour segments. For this purpose, all polygon contour segments are determined which are inside the tile. The endpoint coordinates of these segments (which are always on a tile edge) are stored in separate lists for each edge of a tile and each layer. References to the corresponding polygon points are also stored. The lists are sorted by their x- and y-values, respectively.

Finally, coinciding endpoints of segments in adjacent tiles are determined and the corresponding contour segments are connected. If the connected segments form a complete outer contour, the polygon is finished. Net numbers assigned to conducting layer polygons by the tile processors are unique over all tiles. Replacing the net numbers of all parts of a broken polygon by one common net number for the merged polygon automatically completes the global connectivity information.

In a separate step design rule violations detected several times are determined and eliminated. False DRC-errors that may occur at the window boundary are deleted.

Let n again be the number of polygon edges in the layout and m the number of contour segments to be con-

nected. Assuming uniform distribution of polygons over the cell area, m is proportional to the perimeter of a tile, and thus m is $O(\sqrt{n})$. Since runtime of the merging step is nearly linear in m, its computational complexity is $O(\sqrt{n})$. Therefore, the merging step is fast and causes no significant parallelization overhead.

As discussed in Section 1, hierarchical verification reduces the number of reported design rule violations and is a basic requirement for a hierarchical netlist consistency check. Therefore, even a parallel verification system has to be able to handle verification in a hierarchical way, independently of the obtainable speedups in flat mode. In our verification system, subcell instances are represented by polygons in a special symbolic layer. These instance polygons point to the connecting polygons in the parent cell. Therefore, subcell instances can be handled like primitive devices with additional properties. After area-based parallel verification of a cell, broken subcell instances are merged by the polygon merging step described above. Also, intercell connectivity is completed by merging the connecting polygons and the corresponding net numbers in the parent cell.

4 Implementation on a Workstation Cluster

It is crucial for a parallel verification system, that it can benefit from all current features and future improvements of the sequential algorithms. Therefore, PALACE is implemented using a *wrapper* concept. The sequential program is extended by a well-defined interface. This interface allows the execution of particular functions, for example the verification of a specific cell or cell area or the execution of a certain operation. The sequential algorithms need almost no knowledge whether verification is performed sequentially or in parallel. Therefore, modifications of these algorithms are minimized. The sequential algorithms, adapted to parallel execution, can be used in the sequential verification system without any drawback.

To execute verification in parallel, PALACE uses a master-slave scheme. One master process – started by a UNIX-command – initiates a set of slave processes. The master process sends "order"-messages to the slave processes. When recognizing completion of verification, the master process terminates the slave processes and hereafter itself.

Currently, task scheduling is performed by the master process in the order of arrival. When all subcells of a cell are verified, this cell may be scheduled. If no slave process is idle, the cell is stored in a queue. After receiving a "ready"-message from one of the slave processes, the master process triggers verification of the first cell in the queue. Computation time for verifying a specific cell is unknown until its verification is completed. Therefore, at present cells in the queue cannot be sorted by their computation time in order to minimize verification runtime.

Estimates of computation times of specific cells and certain regions, respectively, are a base requirement for an appropriate task scheduling scheme especially when area-based parallelization is applied. Therefore, development of such estimating algorithms will be a main item of our further research.

Table 1: Cell-based parallel netlist extraction (WS = workstations)

Circuit	Description	Sequential Extr. [Sec]	Speedup Extraction 1 WS	2 WS	4 WS
ACHIP	64 M-DRAM	38,256	1.12	1.67	1.84
MZFO	16 M-DRAM	435	0.96	1.36	1.50
CTAG	ASIC	13,544	1.04	1.64	2.06

Table 2: Cell-based parallel design rule check

Circuit	Description	Sequential DRC [Sec]	Speedup DRC 1 WS	2 WS	4 WS
ACHIP	64 M-DRAM	45,339	1.04	1.33	1.35
MZFO	16 M-DRAM	1,619	0.98	1.49	1.55
CTAG	ASIC	21,598	1.01	1.23	1.27

5 Results

PALACE has been implemented on Hewlett Packard and Sun workstations. To make results comparable, workstations with identical configurations have been used for runtime measurements. Up to four HP 9000/710 workstations complying with this requirement were used. These workstations were interconnected by Ethernet and provided with 32 MB main memory each.

Applying cell-based parallelization alone, the speedups presented in Tables 1 and 2 have been obtained. The results for cell-based parallel verification correspond with the expectations discussed in Section 2. All measured times are elapsed times and given in seconds.

At present, partitioning of a specific cell in PALACE is performed by a user command in a special file. In addition, all tiles of one cell are of the same size. These restrictions cause load imbalance between the involved workstations in most cases. Since an appropriate task scheduling scheme is not yet implemented for reasons discussed in Section 4, at considerable periods some workstations run idle. Therefore, the speedups obtained by cell- *and* area-based parallelization are nearly worst-case results of combining these parallelization strategies and will be improved by future extensions (Table 3).

To demonstrate the potential of geometrical partitioning of cells for parallel verification on workstation clusters, the results of some large cells are presented in Tables 4 and 5.

6 Conclusions

PALACE, a new system for parallel and hierarchical layout verification on clusters of loosely coupled workstations was presented in this paper. It combines cell- *and* area-based parallelization. Applying the concept of *multiple*

Table 3: Cell- and area-based parallelization

Circuit	Speedup Extraction	Speedup DRC
	4 WS	4 WS
ACHIP	2.78	2.57
MZFO	2.40	2.88
CTAG	3.14	2.65

Table 4: Speedups obtained by geometrical partitioning of single large cells (netlist extraction)

Cell	Circuit	Sequential Extr. [Sec]	Speedup Extraction 4 WS
TOPCELL	ACHIP	3,451	3.60
TOPCELL	MZFO	142	3.82
CIRCUIT	CTAG	1,270	3.74

execution switching, layout analysis and circuit extraction are performed in parallel without any functional restriction. Even all-angle geometry causes no problems, since proven sequential verification algorithms are exploited by a *wrapper* concept in the implementation. To prevent large parallelization overhead, a fast merging algorithm based on the connection of contour segments has been developed. The usability of the verification system has been proven by verifying industrial leading edge VLSI circuits, in particular large DRAM and ASIC layouts. High speedups have been obtained reducing time-to-market significantly.

In future, PALACE will be improved as follows:

First, independent operations will be executed in parallel. In this way, even considerably larger workstation clusters may be utilized effectively.

Second, parallelization strategies mentioned above will be combined considering properties of the current cell. In this way, speedup can be maximized and parallelization overhead can be minimized.

Finally, algorithms for runtime estimation and task scheduling will be developed as discussed in Section 4. Load caused by other processes running on the workstations can be taken into consideration. Moreover, in heterogeneous clusters workstations with higher computational power may be used in a more effective way.

Acknowledgement

The authors would like to thank Siemens AG for support of this work.

Table 5: Speedups obtained by geometrical partitioning of single large cells (design rule check)

Cell	Circuit	Sequential DRC [Sec]	Speedup DRC 4 WS
TOPCELL	ACHIP	2,359	3.28
TOPCELL	MZFO	933	3.59
CIRCUIT	CTAG	13,315	3.46

References

[1] W. Meier, ''Hierarchical Layout Verification for Submicron Designs,'' *Proc. 1st European Design Automation Conference*, Glasgow, Scotland, March 1990, pp. 382–386

[2] W. Meier, ''VLSI Logic Verification by Hierarchical Netlist Comparison,'' *Proc. GME Fachtagung Mikroelektronik*, Baden-Baden, Germany, March 1995, pp. 101–106

[3] E. C. Carlson and R. A. Rutenbar, ''Design and Performance Evaluation of New Massively Parallel VLSI Mask Verification Algorithms in JIGSAW,'' *Proc. 27th Design Automation Conference*, Orlando, Florida, June 1990, pp. 253–259

[4] F. Gregoretti and Z. Segall, ''Analysis and Evaluation of VLSI Design Rule Checking Implementation in a Multicomputer,'' *Proc. Int'l Conf. on Parallel Processing*, Bellaire, Michigan, August 1984, pp. 7–14

[5] G. E. Bier and A. R. Pleszkun, ''An Algorithm for Design Rule Checking on a Multiprocessor,'' *Proc. 22th Design Automation Conference*, Las Vegas, Nevada, June 1985, pp. 299–303

[6] B. A. Tonkin, ''Circuit Extraction on a Message-Based Multiprocessor,'' *Proc. 27th Design Automation Conference*, Orlando, Florida, June 1990, pp. 260–265

[7] K. P. Belkhale and P. Banerjee, ''A Parallel Algorithm for Hierarchical Circuit Extraction,'' *Proc. Int'l Conf. on Computer-Aided Design*, Santa Clara, California, November 1990, pp. 236–239

[8] P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design*, Englewood Cliffs, New Jersey: Prentice Hall, 1994

[9] J. D. Marantz, ''Exploiting Parallelism in VLSI CAD,'' *Proc. Int'l Conf. on Computer Design*, Port Chester, New York, October 1986, pp. 442–445