Efficient Methods for Large Resistor Networks

Joost Rommes and Wil H. A. Schilders

***Abstract*—Large resistor networks arise during the design of very-large-scale integration chips as a result of parasitic extraction and electro static discharge analysis. Simulating these large parasitic resistor networks is of vital importance, since it gives an insight into the functional and physical performance of the chip. However, due to the increasing amount of interconnect and metal layers, these networks may contain millions of resistors and nodes, making accurate simulation time consuming or even infeasible. We propose efficient algorithms for three types of analysis of large resistor networks: 1) computation of path resistances; 2) computation of resistor currents; and 3) reduction of resistor networks. The algorithms are exact, orders of magnitude faster than conventional approaches, and enable simulation of very large networks.**

***Index Terms*—Approximate minimum degree, Cholesky decomposition, electro static discharge analysis, graph algorithms, large-scale systems, model order reduction, parasitic extraction, path resistance, resistor networks, strongly connected components, two-connected components.**

I. Introduction

ELECTRO STATIC discharge (ESD) analysis is of vital importance during the design of large-scale integrated circuits and related products. A human touch charged by walking across a carpet, for instance, can affect or destroy a device containing electric components. The costs involved may vary from a few cents to millions if, due to interconnect failures, a respin of the chip is needed. An example of a damaged piece of interconnect that was too small to conduct the required amount of current is shown in Fig. 1.

ESD analysis [1], [2] requires knowledge on how fast electrical charge on the pins of a package can be discharged, and whether the metal interconnect can handle the expected current densities. In many cases, the discharge is done through the power network, the interconnect, and the substrate, which are assumed to be resistive. Furthermore, diodes are used to protect transistors on a chip against peak charges. The discharge paths must be of low resistance to allow for sufficient discharge.

The interconnect and resistance networks are typically modeled by resistors, and diodes are used to connect different parts of the networks. The resulting resistive networks may contain up to millions of resistors, hundreds of thousands of internal nodes, and thousands of external nodes (nodes with connections to diodes and other elements). Simulation of such large networks within reasonable time is not possible, and including such networks in full system simulations may be even unfeasible. Hence, there is a need for much smaller networks that accurately or even exactly describe the resistive behavior of the original network, yet allow for fast analysis.

The main contribution of this paper is that we propose efficient algorithms for three types of analysis of large resistor networks: 1) computation of path resistances; 2) computation of resistor currents; and 3) reduction of resistor networks. We show how insights from graph theory, numerical linear algebra, and matrix reordering algorithms can be used to solve the large linear systems that arise in the dc problems 1) and 2). Similarly, we use these techniques for the reduction of large resistor networks 3); we construct an *equivalent* network with the same number of external nodes, but much fewer internal nodes and resistors, which can be simulated much faster. The algorithms are exact (i.e., no approximation error is made), and ten to thousand times faster than conventional approaches.

Although in the examples we mainly focus on large resistor networks arising in ESD analysis, the proposed techniques are general in the sense that they can be applied to problems, arising in other types of analysis as well, e.g., electromigration and voltage drop (for which special purpose simulators exist [3]).

This paper is organized as follows. In Section II, we summarize the modeling of resistor networks. In Section III, we describe two types of analysis that are common for resistor networks (computation of path resistances and computation of resistor currents), and present efficient algorithms for performing these analyses. A new method for reducing large resistor networks is presented in Section IV. Several practical issues related to reduction of large networks are discussed in Section V. In Section VI, we show numerical results for developed algorithms. Section VII concludes.

II. Modeling of Resistor Networks

In this paper we will assume that the resistor network, which only contains resistors, is a subnetwork of a circuit that may contain elements other than resistors. This circuit, also referred to as the top-level circuit, may consist of just a single current source, as in some cases of electro static discharge analysis, or may consist of a complete design, including nonlinear elements such as transistors and diodes (see Fig. 2). Nodes connecting a resistor and a non-resistor element are called terminals (or external nodes), while nodes connecting only resistors are called internal nodes. Throughout this paper we

Manuscript received December 18, 2008; revised May 7, 2009 and July 14, 2009. This work was supported by the EU Marie-Curie project O-MOORE-NICE! FP6 MTKI-CT-2006-042477. This paper was recommended by Associate Editor L. M. Silveira.

The authors are with NXP Semiconductors, 5656 AE Eindhoven, The Netherlands (joost.rommes@nxp.com; wil.schilders@nxp.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2034402

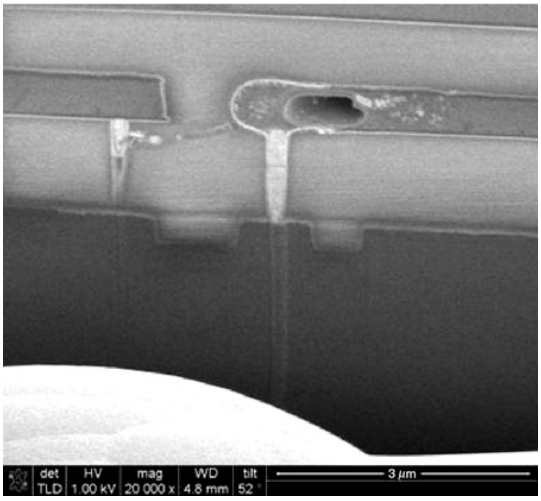

Fig. 1. Example of a piece of interconnect that was damaged because it was too small to conduct the amount of current caused by a peak charge.

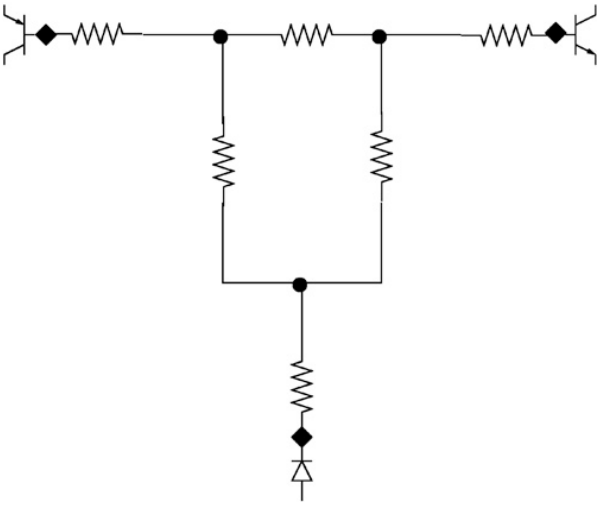

Fig. 2. Resistor network connecting non-resistor elements. All nodes connecting a resistor and a non-resistor element are terminals (or external nodes, here denoted by diamonds) of the resistor network, while nodes connecting only resistors are called internal nodes (circles).

will assume that current can only be injected in terminals. Typical networks from realistic designs have 1–10^4 terminals and 10^4–10^6 internal nodes and resistors.

Although this paper focuses on resistor networks, the methods for fast dc computations presented in Section III can be readily generalized to ac analysis of linear *RCLk* networks. The reduction algorithm described in Section IV can also be extended to *RCLk* networks; however, the reduction will then no longer be exact.

A. Circuit Equations and Matrices

For the purpose of this paper it is sufficient to consider networks consisting of resistors only (see, e.g., [4], [5] for more details on circuit modeling).

Applying Kirchhoff's current law and Ohm's law for resistors leads to the following system of equations for a resistor network with N resistors (resistor i having resistance r_i) and n nodes ($n < N$):

$$\begin{bmatrix} R & P \\ -P^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{i}_b \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{i}_n \end{bmatrix} \tag{1}$$

where $R = \text{diag}(r_1, \ldots, r_N) \in \mathbb{R}^{N \times N}$ is the resistor matrix, $P \in \{-1, 0, 1\}^{N \times n}$ is the incidence matrix, $\mathbf{i}_b \in \mathbb{R}^N$ are the resistor currents, $\mathbf{i}_n \in \mathbb{R}^n$ are the injected node currents, and $\mathbf{v} \in \mathbb{R}^n$ are the node voltages.

The modified nodal analysis method formulation [5] can be derived from (1) by eliminating the resistor currents $\mathbf{i}_b = -R^{-1}P\mathbf{v}$

$$G\mathbf{v} = \mathbf{i}_n \tag{2}$$

where $G = P^T R^{-1} P \in \mathbb{R}^{n \times n}$ is symmetric positive semidefinite. Since currents can only be injected in external nodes, and not in internal nodes of the network, system (2) has the following structure:

$$\begin{bmatrix} G_{11} & G_{12} \\ G_{12}^T & G_{22} \end{bmatrix} \begin{bmatrix} \mathbf{v}_e \\ \mathbf{v}_i \end{bmatrix} = \begin{bmatrix} B \\ 0 \end{bmatrix} \mathbf{i}_e \tag{3}$$

where $\mathbf{v}_e \in \mathbb{R}^{n_e}$ and $\mathbf{v}_i \in \mathbb{R}^{n_i}$ are the voltages at external and internal nodes, respectively ($n = n_e + n_i$), $\mathbf{i}_e \in \mathbb{R}_e^n$ are the currents injected in external nodes, $B \in \{-1, 0, 1\}^{n_e \times n_e}$ is the incidence matrix for the current injections, and $G_{11} = G_{11}^T \in \mathbb{R}^{n_e \times n_e}$, $G_{12} \in \mathbb{R}^{n_e \times n_i}$, and $G_{22} = G_{22}^T \in \mathbb{R}^{n_i \times n_i}$. The block G_{11} is also referred to as the terminal block.

A current source (with index s) between terminals a and b with current j results in contributions $B_{a,s} = 1$, $B_{b,s} = -1$, and $\mathbf{i}_e(s) = j$. If current is only injected in a terminal a (for instance if a connects the network to the top-level circuit), the contributions are $B_{a,s} = 1$ and $\mathbf{i}_e(s) = j$.

Finally, systems (1)–(3) must be made consistent by grounding a node *gnd*, i.e., setting $\mathbf{v}(gnd) = 0$ and removing the corresponding equations. In the following we will still use the notation G for the grounded system matrix, if this does not lead to confusion.

III. Analysis of Resistor Networks

In this section we describe two types of analysis that frequently occur in the design process, and efficient algorithms to perform these analyses: computation of path resistances, and computation of resistor currents. A third type, reduction of resistor networks, is described extensively in Section IV.

A. Path Resistances

The path resistance or effective resistance between two nodes a and b is defined as the ratio of voltage across a and b to the current injected into them [6]. In practice, one is interested in the path resistances from the output of one device to the input of one or more other devices for several reasons: the path resistances can be used to analyze the power dissipation, and in ESD analysis they can be used to check

INPUT: conductance matrix $G \in \mathbb{R}^{n\times n}$, source $\mathbf{i}_n \in \mathbb{R}^n$
OUTPUT: resistor currents
1: Reorder G to minimize fill-in with AMD [8];
2: Compute the Cholesky decomposition $G = LL^T$
3: Compute the DC node voltages $\mathbf{v} = G^{-1}\mathbf{i}_n$:
   1) Solve $\mathbf{x}$ from $L\mathbf{x} = \mathbf{i}_n$
   2) Solve $\mathbf{v}$ from $L^T\mathbf{v} = \mathbf{x}$
4: Compute resistor currents $\mathbf{i}_b = -R^{-1}P\mathbf{v}$

Fig. 3. fastR: computation of resistor currents.

whether unintended peak currents are conducted well enough through the resistive protection network to prevent damage to the chip.

The path resistance between nodes a and b is defined as

$$R_{ab} = (\mathbf{e}_a - \mathbf{e}_b)^T G^{-1}(\mathbf{e}_a - \mathbf{e}_b) \tag{4}$$

where $\mathbf{e}_a$ and $\mathbf{e}_b$ are the ath and bth unit vectors, respectively (i.e., unit current is injected). As described above, in practice one is interested in the path resistances between terminal a and one or more terminals b_i. Since the dimension of G can easily exceed millions, explicit computation of G^{-1} is not an option. Instead, since G is symmetric positive-definite, we use the Cholesky decomposition $G = LL^T$ [7], where $L \in \mathbb{R}^{n\times n}$ is a lower triangular matrix (the rows and columns of G are reordered first using approximate minimum degree reordering [8]). Furthermore, only a single backward substitution is needed to compute the path resistances.

1) Reorder G to minimize fill-in with approximate minimum degree (AMD) [8].
2) Compute the Cholesky decomposition $G = LL^T$.
3) Solve $\mathbf{u}$ from $L\mathbf{u} = (\mathbf{e}_a - \mathbf{e}_b)$.
4) Compute $R_{ab} = \mathbf{u}^T\mathbf{u}$.

If we reorder the unknowns in such a way that the equations for nodes a and b are in the last rows, only a partial backward substitution needs to be done.

B. Resistor Currents

The resistor current is defined as the amount of current flowing through the resistor. In ESD analysis, one is interested in the (dc) current densities of *all* resistors in the network (recall that the resistor network arises from a parasitic extraction on the layout). The current density in a resistor is defined as the current flowing through a given surface perpendicular to the current flow (in this case, this surface would be the area of the cross-section of the resistor or metal track). The current densities are important in ESD analysis, since they are used to check whether the metal interconnect is thick enough to conduct the amount of current. The dimensions of the resistors are known from the layout, so current densities can be computed easily if the currents are known.

The resistor currents can be computed with the algorithm, called fastR, described in Fig. 3.

In steps 1–3, we use again the Cholesky decomposition $G = LL^T$ to efficiently solve the dc state $\mathbf{v}$ from $G\mathbf{v} = \mathbf{i}_n$.

Since R is a diagonal matrix and P has only two nonzero elements per row, the operation in step 4 has complexity $O(n)$. This way, the currents through *all* resistors can be computed with two cheap matrix-vector operations only (when $\mathbf{v}$, the dc operating point, is available). Note that for this analysis, reducing the resistor network is not an option, since the currents through *all* resistors are needed.

Another approach is to solve the resistor currents $\mathbf{i}_b$ directly from (1), but this would require the solution of a large saddle point system that might be harder to solve [9].

C. Remarks on the Cholesky Factorization

Computing the Cholesky factorization $A = LL^T$ of a symmetric positive-definite matrix A is feasible even for very large matrices, as long as they are sparse and allow for effective column reordering (with, e.g., AMD [8]), the main reason being that the factor L is also sparse (but less sparse than the original matrix). It is well known that for matrices that arise in circuit simulation, the complexity for computing the Cholesky factorization is typically $O(n^\alpha)$, where $1 < \alpha \leq 2$ [10]. In Section VI, we show results for networks with several hundreds of thousands of unknowns and resistors that were all computed on desktop computers.

If, however, the number of resistors and nodes becomes too large and/or the matrix becomes too dense, one could consider iterative methods such as conjugate gradients [7] for the solution of (2).

IV. Reduction of Resistor Networks

In the previous section, we considered analysis of networks that consist of only resistors. In general, however, these resistor networks are part of larger systems that contain other elements (diodes, transistors) as well. Full system simulation (e.g., dc analysis and transient analysis) of such complex circuit designs with large (extracted) resistor subnetworks can be time consuming or unfeasible. Hence, there is need for much smaller networks that accurately or even exactly describe the resistive behavior of the original resistor subnetworks, but allow for fast full system analysis, when replacing the original (cf. Fig. 4).

This leads to the problem of reducing large resistor networks, which is defined as follows: given a very large resistor network described by (3), find an equivalent network that:

(a) has the same terminals;
(b) has exactly the same path resistances between terminals;
(c) has $\hat{n}_i \ll n_i$ internal nodes;
(d) has $\hat{N} \ll N$ resistors;
(e) is realizable as a netlist so that it can be (re)used in the design flow as subcircuit of large systems (see Fig. 4 for an example use of a reduced netlist).

Simply eliminating all internal nodes will lead to an equivalent network that satisfies conditions (a)–(c), but violates (d) and (e): for large numbers n_e of terminals, the number of resistors $\hat{N} = (n_e^2 - n_e)/2$ in the dense reduced network is in general much larger than the number of resistors in the sparse original network (N of $O(n)$), leading to increased

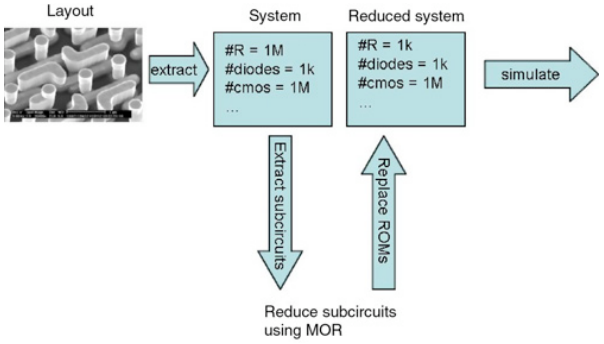

Fig. 4. Typical (re)use of reduced equivalent network in the design flow: the original network is reduced to a smaller network that replaces the original network in the complete system.

memory and central processing unit (CPU) requirements (see Section IV-B).

One might wonder whether interconnect modeling by resistor-only networks is accurate enough in practice and hence why we need reduction of resistor networks. Indeed, for high frequencies one needs to take capacitive and inductive effects into account as well. However, for our applications, mainly in ESD analysis, we are interested in low frequencies [compared to the resistor—capacitor (RC) cutoff frequencies] close to dc and hence resistor-only networks are accurate enough. Consequently, we need accurate reduced networks to speed up or enable full system simulation.

Full circuit simulations, e.g., transient simulation, based on both iterative and direct solvers can take advantage of replacing the resistor networks by reduced resistor networks: for iterative solvers, the matrix–vector multiplications become cheaper, while for direct solvers fewer elimination steps (with less fill-in) are needed (this pays off when more than one solve is needed, but this is generally the case for full circuit simulations).

A. Existing Approaches and Limitations

There are several approaches to deal with large resistor networks. If the need for an equivalent reduced network can be circumvented in some way, this is usually the best to do. To see this, one has to take into account that due to sparsity of the original network, memory usage and computational complexity are *in principle* not an issue, even not for networks containing millions of resistors. Solving linear systems with the related conductance matrices is typically of complexity $O(n^\alpha)$, where $1 < \alpha \leq 2$, instead of the traditional $O(n^3)$ [10], and hence the path resistance and resistor current problems can be solved directly, as we have described in the previous section. Of course, α depends on the sparsity and will rapidly increase as sparsity decreases. This also explains why eliminating all internal nodes does not work in practice: the large reduction in unknowns is easily undone by the enormous increase in number of resistors, mutually connecting all external nodes.

General purpose circuit simulation tools most likely use techniques that exploit the (general) sparsity of the circuit equations. Nevertheless, as we will see in Section VI, the running times increase rapidly with the size of the networks, and even for moderately sized networks they are no longer able to perform the simulations. Although the exact causes of these limitations are not clear (the source code is not available in all cases), obvious reasons are that: 1) the fact that the networks consist of resistors only is not exploited; and (2) the specific type of analysis (path resistances, resistor currents) is not implemented in a way that takes full advantage of (mathematical and/or graph theoretic) structure in problems. The algorithms described in the previous section for these problems can be implemented as stand-alone tools, but can also easily be implemented in general purpose simulators.

However, if we want to (re)use the network in full system simulations including nonlinear elements like transistors, a reduced equivalent network is needed to limit simulation times or make simulation possible at all. There is software [11] available for the reduction of parasitic extracted networks, but this software produces approximate reduced networks while in many cases an exact reduced network is needed. In [12], approaches based on large-scale graph partitioning packages such as (h)METIS [13] are described, but only applied to small networks. Structure preserving (Krylov subspace) projection methods for model reduction [14], [15], finally, have the disadvantage that they lead to dense reduced-order models if the number of terminals is large.

The availability of efficient sparse matrix algorithms has led to a paradox in model order reduction [16], [17]: we can solve linear systems with sparse matrices and millions of unknowns on desktop computers, but reduction of dynamical systems with the same system matrices leads to dense reduced system matrices that are smaller in terms of unknowns, but often (much) larger in terms of elements, especially when the number of inputs and outputs is large. As a result, in practice these reduced systems do not lead to smaller simulation times and less memory usage, but to netlists and simulation times that are even larger than the original.

B. Reduction by Eliminating Internal Nodes

Looking at (3), it may seem attractive to eliminate *all* the internal unknowns by computing the Schur complement of G_{22}, resulting in the much smaller, equivalent system

$$(G_{11} - G_{12}G_{22}^{-1}G_{12}^T)\mathbf{v}_e = B\mathbf{i}_e.$$

Since the number n_e of terminals is usually much smaller than the number n_i of internal unknowns, in terms of unknowns this leads in general to a huge reduction. However, in terms of resistors (and hence network size), the reduction, if any, depends on the number of resistors in the original network. Since the original network is in general very sparse (typically three resistors per node on average) and the number of terminals can be large (up to $\sim 10^4$, see Table II), this approach will lead to network expansion instead of reduction in many cases: the Schur complement will be dense, resulting in $O(n_e^2)$ resistors. Of course, if $n_e \ll n_i$ and $n_e(n_e-1)/2 \ll N$, elimination of all internal nodes may still be attractive (note $n_e(n_e-1)/2$ is the upper bound on the number of resistors in

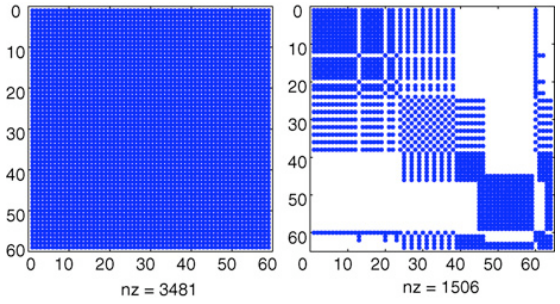

Fig. 5. Two reduced conductance matrices of the same network with 59 terminals. The matrix on the left results when eliminating all internal nodes; on the right, all but five internal nodes are eliminated. The matrix on the right has fewer nonzeros, and only five more rows, than the first matrix, and hence the corresponding network is smaller: 721 versus 1711 resistors. By selecting specific internal nodes (five in this case) to be preserved in the reduced network, fill-in during elimination of the other internal nodes is limited considerably. The difference becomes even bigger as the number of terminals increases.

the reduced network), and since this can be computed before doing the actual elimination we always include a check on this. If it is decided to eliminate all internal nodes, the reduced conductance matrix G_k can be computed efficiently using the Cholesky factorization $G_{22} = LL^T$

$$G_k = G_{11} - QQ^T$$

where $Q = L^{-1}G_{12}^T$.

In many cases, however, elimination of all internal nodes leads to a dramatic increase in the number of resistors and is hence not advisable. On the other hand, we know that the sparse circuit matrices allow for sparse Cholesky factorizations, provided clever reordering strategies such as approximate minimum degree [8] are used. Our algorithm for reducing large networks heavily relies on reordering of rows and columns to minimize fill-in. Before going into all the details in the following sections, we briefly describe the key idea behind our approach.

The idea is best explained by referring to Fig. 5. Here two reduced conductance matrices of the same network are shown. The matrix on the left results when eliminating all internal nodes; on the right all but five internal nodes are eliminated. It is clear that the matrix on the right contains a few more internal nodes, but considerably fewer resistors, thanks to the fact that these specific internal nodes are *not* eliminated. In other words, since these nodes are connected to many of the remaining nodes (after the elimination of other internal nodes; in the original network they are connected to only a few other nodes), eliminating these nodes would cause fill-in in a large part of the matrix. Hence, at the costs of an additional unknown, we gain on sparsity. The question is now how to find these specific internal nodes, that cause the most fill-in, in an efficient way. In the following sections, we will explain how graph algorithms and matrix reordering algorithms can be used to answer this question.

The idea of reduction by node elimination is not new, see, for instance, [18]–[23]. We present a specialized method for resistor networks and by making use of fill-in reducing matrix reordering algorithms it can be applied to very large-scale networks.

INPUT: conductance matrix G, list of terminals
OUTPUT: reduced conductance matrix G

1: Compute connected components G_i of G
2: **for** every connected component G_i in G with one or more terminals **do**
3: Compute the two-connected components $G_i^{(2)}$ of G_i
4: **for** every two-connected component $G_i^{(2)}$ of G_i **do**
5: **if** $G_i^{(2)}$ has no terminals and exactly one articulation node a **then**
6: Remove all resistors and nodes in $G_i^{(2)}$ from G_i, but keep articulation node a
7: **else if** $G_i^{(2)}$ has no terminals and exactly two articulation nodes a and b **then**
8: Replace the network between a and b by a single equivalent resistor
9: **else if** $G_i^{(2)}$ has exactly one terminal t and exactly one articulation node a **then**
10: Replace the network between a and t by a single equivalent resistor
11: **else**
12: Keep $G_i^{(2)}$ (will be reduced in step 15–16)
13: **end if**
14: **end for**
15: Reorder rows and columns of G_i using AMD
16: Eliminate internal nodes one by one, keeping track of reduced system with the fewest resistors
17: Replace G_i by its reduced equivalent in G
18: **end for**

Fig. 6. reduceR: Reduction of large resistor networks.

C. reduceR: Effective Reduction of Resistor Networks

The new approach, called reduceR, that we will describe next, combines well-established techniques for sparse matrices with graph algorithms. As will become clear, this approach has several advantages: the running times do not depend on the number of terminals and are linear (at most quadratic for advanced reduction) in the number of states, both the number of states and number of elements are decreased dramatically, the reduced networks are exact (i.e., no approximations are made), and the approach can be implemented with robust existing software libraries. An outline of the algorithm is described in Fig. 6. We use a divide-and-conquer approach to reduce large networks: first the network is partitioned in smaller parts that can be reduced individually (steps 1–14). In the second and last phase (steps 15–17), the partially reduced network is reduced even further by a global approach. In the following sections, we will describe each of the steps in more detail. Finally, we would like to stress that although in this paper we describe and apply the approach for resistor only networks, it can be applied to RC and RCLk networks as well. The key observation, as will become clear in the following, is that internal nodes are identified that partition the network in disconnected parts that are easier to reduce. We give more details for reduction of general networks in Section V-E.

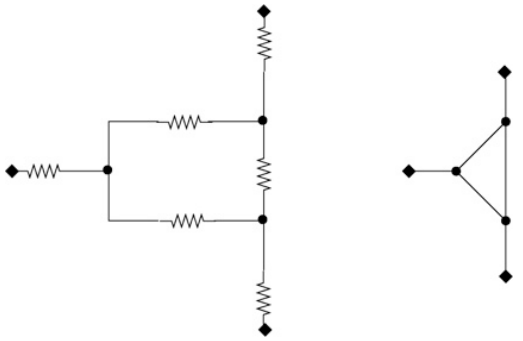

Fig. 7. Example of a simple resistor network and its graph representation. The nodes are the vertices of the graph, the resistors are the edges. Terminals (external nodes) are denoted by diamonds, internal nodes by circles.

In the following, we make extensive use of graphs [24]. A graph $\mathcal{G} = (V, E)$ consists of a set V of vertices (or nodes) and a set E of edges between nodes. If all edges in a graph are undirected, the graph is called undirected. It is well known that a circuit can be represented as an undirected graph: the nodes of the circuit are the vertices, and the resistors are the edges. Hence, for resistor networks we have $V = \{0, 1, 2, 3, \ldots, n-1\}$ (there are n nodes, assumed to be numbered consecutively starting at 0) and $E = \{(i, j)|$ there is a resistor between nodeS i and $j\}$. Furthermore, $|V| = n$ (the number of nodes) and $|E| = N$ (the number of resistors). The degree of a vertex is defined as the number of edges incident to it. See Fig. 7 for an example graph.

D. Steps 1, 2: Strongly Connected Components

A graph is called connected if there exists a path from any vertex to any other vertex in the graph [24]. It is easy to see that if a network is not connected, a first simplification can be obtained by considering the connected subgraphs separately. Computing the strongly connected components of an undirected graph is a well-known graph problem and can be solved in $O(|V|+|E|) = O(n+N)$ time using two depth first searches [24]. There are many standard implementations available; in our experiments we have used the function `dmperm` of MATLAB [25].

The presence of connected components is very natural in extracted networks: a piece of interconnect connecting two other elements such as diodes or transistors, for instance, results in an extracted network with two terminals, disconnected from the rest of the extracted circuit.

It might also happen that some connected components do not contain any terminals at all. It is clear that such so-called floating nets can safely be removed from the network and do not return in the reduced netlist (cf. the check in step 2).

Fig. 8 shows a collection of connected components that belongs to one circuit. They are connected to each other through diodes in the original circuit.

Eliminating all internal nodes, in general, will lead to a dense matrix, except in cases when strongly connected components occur: there a less dense matrix results, that is however still far from sparse. Therefore, we always first compute the strongly connected components and then deal with the components one by one, as described in the following.

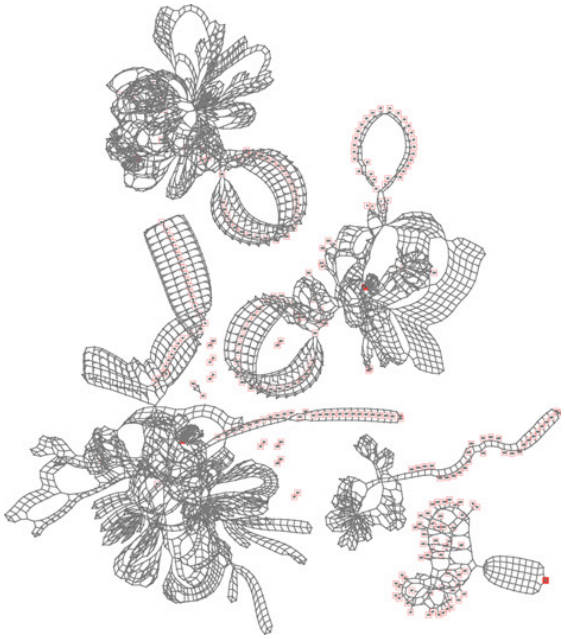

Fig. 8. Collection of connected components that belong to the same circuit. The components are connected to each other through diodes between the terminals (squares).

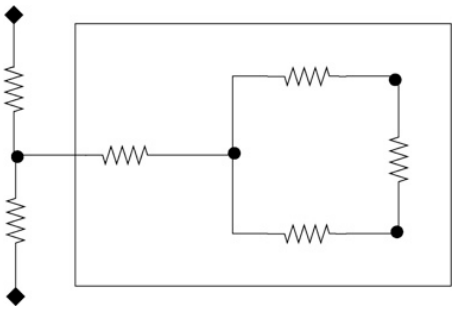

Fig. 9. Example of a simple resistor network with a relatively large dangling part (boxed). Since no current is flowing in the dangling part, it can be removed from the network.

E. Steps 2–14: Two-Connected Components

The next phase is to compute the two-connected (or biconnected) components of every connected component. A connected component is called two-connected if it becomes disconnected by removing one vertex [26]. The vertices in a connected component with this property are called articulation nodes. Similar to the classification of connected components, the two-connected components and articulation nodes can be used to obtain a next reduction. The two-connected components of a connected graph can be computed in $O(|V| + |E|) = O(n + N)$ time. See [27] for an efficient implementation of graph algorithms in MATLAB.

We describe here some of the options, but several other options are possible as well.

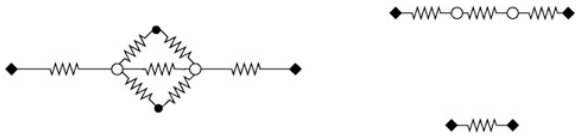

Fig. 10. Resistor network (left) with two terminals (diamonds), two internal nodes (circles), and two articulation nodes (open circles). A first simplification is to replace the network between the two articulation nodes by a single equivalent resistor (top right). A further simplification is obtained by replacing the remaining resistors by a single equivalent resistor between the terminals. Note that all networks are equivalent.

1) *Steps 5, 6. Dangling Networks:* If a two-connected component is connected to the rest of the graph by only one articulation node, and if there are no terminals in this two-connected component, it can be removed from the network (except for the articulation node) without making an error. Such parts of the network are also called dangling in electrical engineering, see also Fig. 9.

2) *Steps 7, 8. No Terminals and Two Articulation Nodes:* If a two-connected component is connected to the rest via two articulation nodes and has no terminals, it can be replaced by a single resistor connecting the two articulation nodes (cf. Fig. 10). The resistance of this resistor can be computed using the elimination procedure described in Section IV-B.

3) *Steps 9, 10. One Terminal and One Articulation Node:* If a two-connected component is connected to the rest via exactly one articulation node and has exactly one terminal, it can be replaced by a single resistor connecting the terminal and the articulation node (cf. Fig. 10). The resistance of this resistor can be computed using the elimination procedure described in Section IV-B.

4) *Steps 11, 12. More than Two Articulation Nodes:* If a two-connected component has multiple articulation nodes and terminals, one can choose to eliminate all non-terminals and non-articulation nodes (if this gives a large enough reduction in number of resistors, cf. Section IV-B), to apply a more advanced reduction scheme, or to keep the component for the moment and reduce it in the next phase together with the other components.

Although articulation nodes typically do not occupy much of the network, it makes sense to check on their presence: they help to identify the global structure of the network, whereas fill-in minimizing reordering algorithms, as described next, tend to focus more on local properties.

F. Steps 15, 16: Elimination of Selected Internal Nodes

This phase is the crucial phase in the algorithm, since here it is determined which internal nodes to eliminate and which not. The paradigm is: eliminate those internal nodes that give the least fill-in. As discussed before, eliminating all internal nodes is usually not an option. Furthermore, it is well known that determining the optimal order of elimination is nondeterministic polynomial time (NP)-complete [8]. Fortunately, there exist efficient algorithms to determine a suboptimal order. In our approach, we make use of the approximate minimum degree [8] algorithm.

The AMD algorithm reorders the rows and columns of a matrix G such that the fill-in generated during the computation of the Cholesky factorization $G = LL^T$ (or lower upper factorization) is minimal. Since finding the optimal order is NP-complete, AMD uses an approximate degree to estimate the fill-in generated by eliminating a node (i.e., the approximate degree of a node during factorization). In an iterative process, AMD selects the nodes with the least approximate degree for elimination, and updates the approximate degree of the remaining nodes. Hence, AMD finds a suboptimal order. Selection of nodes and updates of approximate degrees is done efficiently by using quotient graphs. For sparse matrices, AMD runs in linear time, and it is successfully used in many applications [28]. For more details, see [8].

There is one important difference, however, between the usual application of AMD for solving linear systems and the network reduction problem: in our case, the terminals (a subset of all unknowns) cannot be eliminated and hence they have to be treated in a special way.

We have considered four alternatives for this.

1) Apply AMD to the original matrix and eliminate internal nodes one-by-one in the order computed by AMD. Skip terminals that might have mixed with the internal nodes. Since this turned out to give only a modest reduction in the number of internal nodes and resistors, we will not consider this option any further. See also Fig. 11.
2) Apply AMD to the block corresponding to internal nodes only [cf. block G_{22} in (3)]. Numerical experiments have shown that not taking into account the fill-in caused in (and by) the blocks corresponding to terminals (the blocks G_{11} and G_{12} are neglected) results in a reordering that is far from optimal and hence we will not consider this option any further.
3) Use constrained AMD [28] so that the terminal nodes will be among the last to eliminate. To find the optimal reduction, eliminate internal nodes one-by-one in the order computed by AMD, and keep track of the reduced system with fewest resistors. The rationale behind this is that we would like the rows corresponding to terminals to be considered as well in the reordering process, but that they should be among the last unknowns to be eliminated. Nevertheless, this keeps the possibility open for internal nodes that cause even more fill-in to be detected by AMD (and hence to be kept in the reduced network). See Fig. 12.
4) *Recursive Reordering*. Apply AMD to the original matrix and eliminate internal nodes one-by-one in the order computed by AMD. As soon as a terminal is encountered, move it to the last row and reorder the partially reduced matrix. Repeat this procedure until all internal nodes are eliminated, while keeping track of the reduced system with fewest resistors. The idea behind this recursive approach is that AMD for the original network gives the most optimal reordering. However, the terminals can end up anywhere in this order. By eliminating the internal unknowns up to the first occurrence of a terminal, moving the corresponding row(s) to the last row(s) and then computing a new order for the partially reduced matrix and repeating the procedure we might expect to arrive at

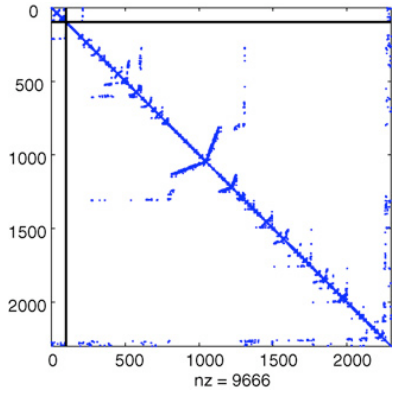

Fig. 11. Unreduced conductance matrix (59 terminals) after reordering with AMD. The first terminal nodes appear already in row 97 (indicated by the solid lines), while the last terminal is in row 1335. Hence, using this reordering will not result in a good reduction in nodes and resistors. However, after having eliminated the first 96 internal nodes, a new reordering can be computed, and a new set of internal nodes can be eliminated. Repeating this leads to good reduction rates for some networks.

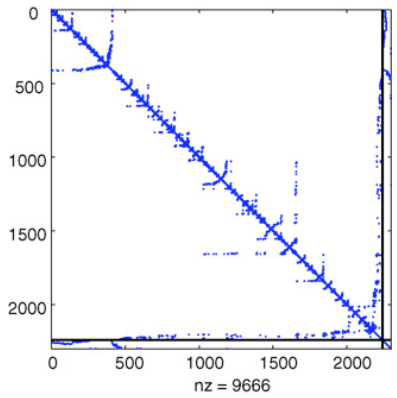

Fig. 12. Unreduced conductance matrix (59 terminals) after reordering with constrained AMD. The terminal nodes appear in the last 59 rows and columns (indicated by the solid lines). This approach leads to good reduction rates in most of the cases (the recursive approach, cf. Fig. 11, can be used in case of poor reduction rates).

a reduced network that is much smaller. The worst-case running time for this approach depends quadratically on the number of nodes and resistors. See Fig. 11.

In reduceR we have implemented alternatives 3 and 4, including an option to do both and take the smallest reduced network. Our experience in practice is that alternative 3 satisfies in 95% of the cases; in 5% of the cases alternative 4 gives a huge reduction in the number of nodes and resistors, while the additional computational effort is limited.

G. Complexity

The connected components and the two-connected components can be detected with $O(n+N)$ operations. The total costs for eliminating the internal nodes are $O(n^\alpha)$, where $1 < \alpha \leq 2$, since the original conductance matrix is very sparse [10]. The worst case costs for the recursive approach, where AMD is called repetitively, are $O(n^2)$ [8]. However, in practice we observed much lower costs (nearly linear complexity) in all cases. Hence, the reduction algorithm is perfectly scalable to very large sparse networks, as is also confirmed by the experiments in Section VI.

INPUT: Netlist
OUTPUT: fastR: resistor currents
reduceR: reduced netlist

```
Parse netlist
Extract resistor network and determine terminals
Setup equations and form conductance matrix G
if fastR mode then
    Compute resistor currents with fastR (Fig. 3)
    Output voltages and currents
else if reduceR mode then
    Reduce network with reduceR (Fig. 6)
    Output reduced netlist
end if
```

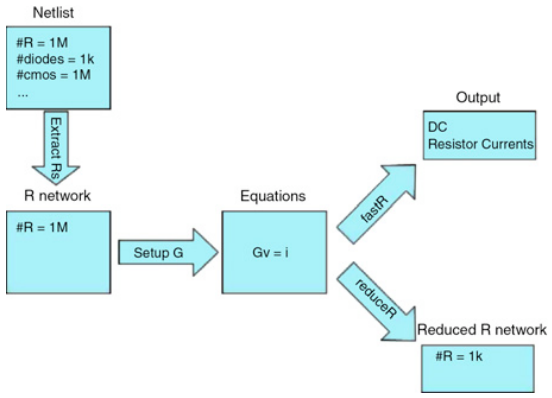

Fig. 13. Framework for fastR and reduceR.

V. Practical Issues

A. Integration in Design Flow

The algorithms presented in the previous sections can easily be integrated in the design flow. A framework that can be used is shown in Fig. 13.

For computing resistor currents (and/or just the dc operating point, which is required for the resistor currents anyway), fastR can replace a general purpose simulator. When reduced networks are needed, reduceR can be used as shown in Fig. 4.

B. Implementation Details

The matrix–vector operations and other computational work can be implemented readily in any matrix manipulation software (e.g., MATLAB [25]) or any other language, since all required functions are available (see [28] for robust implementations of AMD and Cholesky factorizations).

Parsing the netlists can be done efficiently for both resistor only networks and netlists, containing other elements such as transistors, since we only need to keep track of the resistors, internal nodes, and the nodes that connect resistors to non-resistor elements (terminals).

C. Reduction by Approximation

The algorithms for computing resistor currents and reduced networks are exact in the sense that no approximations are made. Especially for reduced resistor networks, an obvious way to reduce the network even further would be to remove selected resistors from the network. Although for some examples an additional reduction of about 15% of the resistors can be obtained, we do not apply this kind of approximation for the following reasons. First, upper bounds on the error made that are known from matrix perturbation theory [29], [7] are much too loose to be practical. Second, explicit computations using the Sherman–Woodbury formula [7] for rank-one updates do give exact insight into the error made and hence provide a way to select resistors to remove, in practice these computations can be expensive and again result in only minor reduction. Third, intuitive or local approximation techniques, e.g., removing relatively small conductances (large resistors), have the big disadvantage that there is no control on the global error. We refer to [29], [30] for more details on approximation techniques.

D. Number of Nodes Versus Number of Resistors

It is not clear beforehand what the optimal ratio of number of nodes against number of resistors is, since this depends on many factors. Looking at the size of the netlist, the number of resistors should be minimized (every resistor is one line in the netlist). The time needed for simulation, however, depends completely on the datastructures and algorithms used in the circuit simulator, and this information is usually not available. Having in mind that solving linear equations with sparse (unreduced) circuit matrices (the most common operation in circuit analysis) has complexity $O(n^\alpha)$, where $1 < \alpha \leq 2$ [10] and that reduced circuit matrices tend to be more dense, driving $\alpha \to 3$, a strategy focused on finding a trade-off between number of internal nodes and resistors seems most appropriate.

In practice we obtained the best results when minimizing the number of resistors, since this also tends to minimize the number of internal nodes. However, our reduction algorithm can easily be adapted for other criteria like sparsity or any function of the number of nodes and resistors: provided the use of a fill-in minimizing reordering still makes sense, the only part that needs to be changed is step 16 in reduceR (Fig. 6), where we keep track of the best reduced network sofar.

E. Application to General Networks

The algorithms for computing the dc operating point and resistor currents can easily be generalized to fast and specialized ac solvers for general RCLk networks, since the key ingredient is the exploitation of structure and usage of sparse factorization algorithms. A discussion on whether such a solver should be a stand-alone tool or part of a circuit simulator is beyond the scope of this paper.

The authors are currently working on a generalization of the resistor network reduction algorithm to general RCLk networks. Here the crucial observation is that the determination of internal nodes that have to be preserved in the reduced network to ensure sparsity is, in principle, a graph problem that does not depend on the types of elements in the network.

F. Advanced Algorithms

The process of determining the internal nodes that need to be preserved in the reduced network can be seen as a structure-revealing process. Typically, networks that result from parasitic extraction show a lot of the structure of the originating layout. The difficulty in practice is that one cannot assume to have this layout information available: the resistor networks are considered individually. It is clear that any knowledge about the layout could be used to find even smaller reduced networks. It is, therefore, not unthinkable that a combination of parasitic extraction techniques with the algorithms presented in this paper leads to even better results.

One could ask whether the first phase—partitioning in connected and two-connected components—could be skipped, since AMD might also detect these structural properties. In fact, AMD does detect connected components, but, as will be illustrated in Section VI-B, being an *approximate* minimum degree algorithm, and inherently local, it may not detect more global properties such as articulation nodes and two-connected components. Making use of articulation nodes may lead to better reductions, and since the costs for detecting articulation nodes depend only linearly on the number of nodes and edges, it is worthwhile to do this before running AMD. A step further would be to detect tri-connected components [31] as well, but this is beyond the scope of this paper.

One also recognizes similarities to matrix partitioning for parallel computing, where the matrix needs to be divided over the available processors in such a way that communication is minimized [13], [32]. Translated to our problem, this would mean to divide the network in subnetworks in such a way that the terminals are evenly divided over the subnetworks, and that the number of nodes connecting subnetworks is minimal.

VI. Numerical Results

In this section, we present the results of the algorithms for large resistor networks that come from realistic designs of very-large-scale integration (VLSI) chips. In Section VI-A, we compare the performance of fastR, our implementation for computing resistor currents in very large networks, to the performance of general purpose circuit simulators. In Section VI-B, we show results for the reduction of very large resistor networks with many terminals with our implementation reduceR.

Both fastR and reduceR are implemented in MATLAB [25] 7.5 (R2007B), relying heavily on the functions `amd` and `symamd` [33], [34]. For the graph algorithms we used the MATLAB library MatlabBGL [27]. All experiments were carried out on Linux platforms on Intel Xeon 2.66 GHz machines with 16 GB RAM. For the circuit simulations we used Spectre (ver. 6.1.1.378) [35] and Pstar (ver. 5.4.2) [36].

TABLE I
CPU TIMES FOR SPECTRE, PSTAR, AND FASTR, AND SPEEDUPS

Circuit	#nodes	#resistors	Spectre (dc)	Pstar (dc)	fastR (dc)	Speedup fastR/Spectre	Speedup fastR/Pstar
vssco1.gnds!	5k	6k	3 s (0.1 s)	2.15 (0.33 s)	0.2 s (0.01 s)	15	10
C65LR7402_APIO5VTF3V3.vdde5v	19k	46k	180 s (1 s)	149 s (2.69 s)	0.8 s (0.02 s)	225	186
supco3v3.VDDCORE3V3	32k	99k	600 s (3 s)	400 s (1.68 s)	1.5 s (0.04 s)	400	267
CLN45ESDTC1_notiles.23	36k	188k	2900 s* (–)	481 s (3.71 s)	7 s (0.1 s)	414*	68
A_VX_2XFULL.A_VX_2XFULL	72k	476k	17700 s (252 s)	6600 s (380 s)	8 s (0.3 s)	2212	825
ohmchck.vssi	344k	701k	400 s* (–)	33000 s* (–)	14 s (3.14 s)	28*	2350*
NLM14SWPTCF1.vssi	560k	917k	64 ks* (–)	88 ks* (–)	21 s (5.78 s)	3060*	4190*
C90L065ESD8.gnd	586k	1.5M	* (–)	* (–)	33 s (13.1 s)	∞	∞
C45LR74020_LVDSTXPHY	2.2M	6.2M	* (–)	* (–)	330 s (300 s)	∞	∞

Times denoted with an asterisk (*) denote termination with error. Between parentheses, CPU times for dc computation are listed.

TABLE II
RESULTS FOR THE REDUCTION ALGORITHM REDUCER: ORIGINAL NETWORKS VERSUS REDUCED ORDER MODELS (ROMS)

	Network I		Network II		Network III		Network IV		Network V	
	Original	ROM	Original	ROM	Original	ROM	Original	ROM	Original	ROM
#terminals	3260		1978		15299		8500		8000	
#int nodes	99k	8k	101k	1888	1M	180k	48k	26k	46k	6k
#resistors	161k	56k	164k	39k	1.5M	376k	75k	60k	67k	26k
#other elements	1874		1188		8250		7500		29k	
#other unknowns	0		0		0		8k		11k	
CPU time reduction	130 s		140 s		1250 s		80 s		75 s	
CPU time simulation	67 h	6 h	20 h	2 h	–	120 h	–	470 s	–	392 s
Speed up	11×		10×		∞		∞		∞	

A. Computing Resistor Currents with fastR

We have tested fastR on a number of large resistor networks that resulted from ESD analysis on designs of VLSI chips. As can be seen in Table I, general purpose circuit simulators like Spectre [35] and Pstar [36] need much more CPU than fastR and fail even for the largest test cases (these were exactly the reasons for developing fastR). The currents computed by fastR are exact and equal to the currents computed by Spectre and Pstar (if available).

We stress that for Spectre and Pstar more than 90% of CPU time is used for parsing the netlist, setting up equations, and, probably, computing the currents through resistors (the costs for computing the dc solution also seem to increase rapidly for Spectre and Pstar as the number of resistors increases). The advantage of fastR is that we can make use of optimized parsers and sparse matrix datastructures and computations, while Spectre and Pstar possibly have to work with slower, but more general, datastructures.

Taking this into account, the comparison of special purpose solver fastR to general purpose simulators such as Spectre and Pstar might seem unfair. However, one of the goals here is to show that it is indeed worthwhile to develop special purpose simulators and that, for specialized large-scale applications, one has to rely on special purpose simulators.

B. Reduction of Resistor Networks

Table II shows results for resistor networks that result from ESD analysis of realistic chip designs and interconnect layouts. Networks I–III contain diodes as well, and one is interested in the path resistances between certain terminals. Since simulation of the original networks is time consuming or not possible at all, reduced networks are needed. Networks IV and V also contain other nonlinear elements, such as transistors. Here the original networks could not be simulated.

The reduction factors for the number of internal nodes vary from 5× for network III to 50× for network II. The reduction factors for the number of resistors vary from 2.5× for network IV to 4.5× for network III. As a result, the computing time for calculating path resistances, including the computation of the reduced network, is more than ten times smaller.

Although the times for computing the reduced network do not depend on the number of terminals, the obtained reduction rates do: network I and network II are quite similar, except for the number terminals (3260 versus 1978). This difference in terminals is also visible in their respective connected components. Following the reasoning in Section IV-B, reducing a component with many terminals is in general harder than reducing a component with only a few terminals: the more terminals, the more fill-in, or, equivalently, the fewer options for eliminating internal nodes. This explains why the reduction for network II is much better, both in terms of resistors and in terms of internal nodes.

1) *Eliminating All Internals Nodes Versus Partial Elimination:* In Table III, we compare the CPU and memory costs for reduced networks that were obtained with reduceR to reduced networks where all internal nodes were eliminated. Except for the first (and smallest) test case, we see that the memory usage is much lower for the networks reduced with reduceR. As the number of terminals increases, also the CPU times for the networks reduced by reduceR are smaller, even when a considerable amount of internal nodes are preserved. Note that here we used 20 dc solves of the resistor network only

TABLE III
RESULTS FOR THE REDUCTION ALGORITHM REDUCER COMPARED TO ELIMINATION OF ALL INTERNAL NODES

	Original					Reduced with reduceR				Elimination of all internal nodes		
	#terminals	#int nodes	#R	CPU (s)	MEM (kb)	#R	#int nodes	CPU (s)	MEM (kb)	#R	CPU (s)	MEM (kb)
1	39	1542	2476	0.017	84	658	8	0.1	21	741	0.004	18
2	59	2300	3683	0.03	125	534	53	0.04	14	1711	0.007	42
3	76	1210	1936	0.023	654	1304	293	0.15	37	2850	0.02	69
4*	430	13 168	21 209	1.1	719	15 795	7201	1.75	501	92 235	1.1	2200
5*	509	16 053	25 594	1.5	871	12 633	652	0.35	321	129 286	1.6	3100
6*	626	10 202	15 740	3.2	540	14 819	6279	1.5	466	195 625	3.1	4705

Cases marked with an asterisk (*) required the recursive reordering as described in Section IV, alternative 4. The number of resistors is denoted by #R.

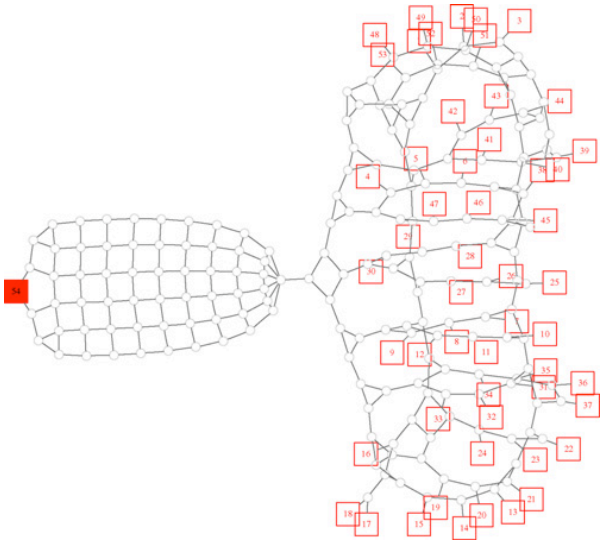

Fig. 14. Network (54 terminals, 190 internal nodes, and 333 resistors) with a large two-connected component at the left. Since this two-connected component contains a single articulation node (center of the graph) and a single terminal (leftmost square), it can be replaced by a single equivalent resistor connecting the articulation node and the terminal.

to compare CPU times; when we insert the reduced networks back into the original systems including active components the differences become even larger, as can be seen in Table II. Finally, we remark that for cases 4–6 the recursive reordering as described in Section IV-F, alternative 4, was required to obtain a significant reduction.

2) *The Effect of Two-Connected Components:* A small but illustrative case for computing two-connected components is the network (54 terminals, 190 internal nodes, 333 resistors) shown[1] in Fig. 14. It is immediately clear that the net in the left of the graph is a two-connected component, with the two center-nodes as articulation nodes. It is also clear that we can replace this two-connected component by a single equivalent resistor connecting the articulation node and the terminal at the left. AMD, however, in all the variants that we have described, does not detect this and the best reduction (in terms of number of resistors) it can find is a network with 188 internal nodes and 333 resistors. If we now first compute the two-connected components and apply reduction as described in Fig. 6, followed by AMD, we obtain a reduced network with 132 internal nodes and 225 resistors.

[1]This graph and the graph in Fig. 8 are drawn using Neato [37]. Neato uses mass-spring based modeling to draw graphs in a natural way. Although not scalable to very large networks, it gives some insight into the structure of the networks.

VII. CONCLUSION

Two algorithms, fastR and reduceR, for efficient computations with large resistor networks have been presented. With fastR one can efficiently compute the currents through all resistors in large resistor networks that typically arise in electro static discharge analysis. Making use of sparse matrix computations, fastR is much faster for large networks than general purpose simulators.

If a circuit consists of one or more large resistor subnetworks and also contains many nonlinear elements such as transistors, full circuit simulation is often time consuming or not possible at all. In that case reduceR can be used to reduce the large resistor networks to equivalent but much smaller resistor networks. These reduced networks can be (re)used in full circuit simulation and since the reduced networks are exactly equivalent, no approximation error is made. The reduction rates in the number of nodes and resistors vary from 80% to 98% and from 60% to 80%, respectively. As a result, full circuit simulations, including the time needed for reducing the networks, are more than ten times faster or possible with the reduced networks only.

ACKNOWLEDGMENT

The authors would like to thank J. Fleurimont, M. Stoutjesdijk, H. van Walderveen, and H. van Zwol of NXP Semiconductors for stimulating discussions about ESD and resistor networks, and for providing the large resistor networks. The authors are also grateful to M. Sevat and T. Beelen of NXP Semiconductors for their helpful comments on earlier drafts of this paper.

REFERENCES

[1] J. M. Kolyer and D. Watson, *ESD: From A To Z.* Dordrecht, The Netherlands: Kluwer, 1999.
[2] *Electrostatic discharge association* [Online]. Available: http://www.esda.org
[3] Cadence. *VoltageStorm* [Online]. Available: http://www.cadence.com
[4] L. O. Chua and P. Lin, *Computer Aided Analysis of Electric Circuits: Algorithms and Computational Techniques*, 1st ed. Englewood Cliffs, NJ: Prentice-Hall, 1975, ch. 4.
[5] W. J. McCalla, *Fundamentals of Computer Aided Circuit Simulation*, 1st ed. Dordrecht, The Netherlands: Kluwer, 1988, ch. 1.
[6] A. Ghosh, S. Boyd, and A. Saberi, "Minimizing effective resistance of a graph," *SIAM Rev.*, vol. 50, no. 1, pp. 37–66, 2008.

[7] G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: John Hopkins University Press, 1996.
[8] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *Soc. Ind. Appl. Math. J. Matrix Anal. Appl.*, vol. 17, no. 4, pp. 886–905, 1996.
[9] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, vol. 14, no. 1, pp. 1–137, 2005.
[10] J. R. Phillips and L. M. Silveira, "Poor Man's TBR: A simple model reduction scheme," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 1, pp. 43–55, Jan. 2005.
[11] Edxact. *Jivaro* [Online]. Available: http://www.edxact.com
[12] P. Miettinen, M. Honkala, and J. Roos, "Using METIS and hMETIS algorithms in circuit partitioning," Circuit Theory Lab., Helsinki Univ. Technol., Helsinki, Finland, Rep. CT-49, 2006.
[13] G. Karypis and V. Kumar. (1998, Nov. 30). *METIS, A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, ver. 4.0.1 [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis
[14] R. W. Freund, "SPRIM: Structure-preserving reduced-order interconnect macromodeling," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2004, pp. 80–87.
[15] Z. Bai, R. Li, and Y. Su, "A unified Krylov projection framework for structure-preserving model order reduction," in *Model Order Reduction: Theory, Research Aspects and Applications* (Mathematics in Industry Series), vol. 13. W. H. A. Schilders, H. A. van der Vorst, and J. Rommes, Eds. Berlin, Germany: Springer-Verlag, 2008.
[16] J. Rommes, P. Lenaers, and W. H. A. Schilders, "Reduction of large resistor networks," presented at Scientific Comput. Electr. Eng., Espoo, Finland, 2008.
[17] Z. Ye, Z. Zhu, and J. Phillips, "Model order reduction for large scale and many-terminal systems," presented at Manage. Organization Rev. Workshop, Shanghai, China, 2008.
[18] B. N. Sheehan, "Realizable reduction of extracted *RC* networks," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design Integr. Circuits Syst.*, 1999, pp. 200–203.
[19] B. N. Sheehan, "Realizable reduction of *RC* networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 8, pp. 1393–1407, Aug. 2007.
[20] M. H. Chowdhury, C. S. Amin, Y. I. Ismail, C. V. Kashyap, and B. L. Krauter, "Realizable reduction of rlc circuits using node elimination," in *Proc. Int. Symp. Circuits Syst.*, vol. 3. 2003, pp. 494–497.
[21] C. S. Amin, M. H. Chowdhury, and Y. I. Ismail, "Realizable reduction of interconnect circuits including self and mutual inductances," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 2, pp. 271–277, Feb. 2005.
[22] Z. Qin and C. Cheng, "Realizable parasitic reduction using generalized Y-Δ transformation," in *Proc. Design Autom. Conf.*, 2003, pp. 220–225.
[23] C. S. Amin, M. H. Chowdhury, and Y. I. Ismail, "Realizable rlck circuit crunching," in *Proc. Design Autom. Conf.*, 2003, pp. 226–231.
[24] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 1st ed. Cambridge, MA: MIT Press, 1990, ch. 22.
[25] Mathworks, Inc. *MATLAB*, ver. 7.5.0.338 [Online]. Available: http://www.mathworks.com
[26] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. Berlin, Germany: Springer-Verlag, 2003, ch. 15.
[27] D. Gleich. (2008, Oct. 21). *MatlabBGL: A MATLAB Graph Library*, ver. 4.0 [Online]. Available: http://www.stanford.edu/dgleich/programs/matlab-bgl
[28] T. A. Davis. *Suite Sparse: A Suite of Sparse Matrix Packages*, ver. 3.4.0 [Online]. Available: http://www.cise.ufl.edu/research/sparse/SuiteSparse
[29] F. Yang, X. Zeng, Y. Su, and D. Zhou, "RLC equivalent circuit synthesis method for structure-preserved reduced-order model of interconnect in VLSI," *Commun. comput. Phys.*, vol. 3, no. 2, pp. 376–396, 2008.
[30] V. Rao, J. Soreff, R. Ledalla, and F. Yang, "Aggressive crunching of extracted RC netlists," in *Proc. Tau Workshop*, Dec. 2002, pp. 72–77.
[31] J. E. Hopcroft and R. E. Tarjan, "Dividing a graph into triconnected components," *SIAM J. Comput.*, vol. 3, no. 3, pp. 135–158, Mar. 1973.
[32] A. I. Zečević and D. D. Šiljak, "Balanced decompositions of sparse systems for multilevel prallel processing," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 41, no. 3, pp. 220–233, Mar. 1994.
[33] P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 381–388, 2004.
[34] T. A. Davis, J. R. Gilbert, S. Larimore, and E. Ng, "Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 377–380, 2004.
[35] Cadence. *Spectre*, ver. 6.2.0.493 [Online]. Available: http://www.cadence.com
[36] NXP Semiconductors. *Pstar*, ver. 6.0 [Online]. Available: http://www.nxp.com
[37] J. Ellson, E. Gansner, Y. Hu, and A. Bilgin, *Graphviz: Graph Visualization Software*, ver. 2.16 [Online]. Available: http://www.graphviz.org

Joost Rommes received the M.Sc. (cum laude) degree in computational science, the M.Sc. (cum laude) degree in computer science, and the Ph.D. degree in mathematics from Utrecht University, Utrecht, The Netherlands, in 2002, 2003, and 2007, respectively.

From 1999–2003, he worked as a Software Engineer at Ordina Technical Automation, Nieuwegein, The Netherlands. He is currently a Researcher at NXP Semiconductors, Eindhoven, The Netherlands, where he works on model reduction. His research interests include model order reduction, specialized eigenvalue methods, and algorithms for problems related to circuit design and simulation.

Wil H. A. Schilders received the M.Sc. degree in mathematics (cum laude) from Radboud University, Nijmegen, The Netherlands, in 1978, and the Ph.D. degree from Trinity College Dublin, Dublin, Ireland, in 1980.

He has been employed with Philips Electronics, Amsterdam, The Netherlands, and has done extensive work on simulation methods for semiconductor devices, electronic circuits, electromagnetics, and other problems related to the electronics industry since 1980. He has been a Part-Time Professor of Numerical Mathematics for Industry at the Eindhoven University of Technology, Eindhoven, The Netherlands, since 1999. In 2006, Philips' semiconductor activities were transferred to NXP Semiconductors, Eindhoven, The Netherlands, where he currently leads a Mathematics Group.

Dr. Schilders is the Vice-President of the European Consortium for Mathematics in Industry, and the Editor-in-Chief for the Dutch Journal for Mathematicians.