Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
INTEGRATION, the VLSI journal 34 (2003) 65–86
INTEGRATION theVLSI journal
www.elsevier.com/locate/vlsi

Balanced multi-level multi-way partitioning of analog integrated circuits for hierarchical symbolic analysis ☆

Sheldon X.-D. Tan[a,*], C.-J. Richard Shi[b]

[a] *Department of Electrical Engineering, University of California, Bourns Hall A223, Riverside, CA 95134, USA*
[b] *Department of Electrical Engineering, University of Washington, Seattle, WA 98195, USA*

Received 7 May 2002; received in revised form 15 January 2003; accepted 15 January 2003

Abstract

This paper considers the problem of partitioning analog integrated circuits for hierarchical symbolic analysis based on determinant decision diagrams (DDDs). The objective is to use DDDs with the minimum number of vertices to represent all the symbolic expressions. We show that the problem can be formulated as that of multi-level multi-way hyper graph partitioning with balance constraints, and be solved in two phases by connectivity-oriented initial clustering and iterative improvement. Our new contribution consists of a fast and effective heuristic for constructing a balanced initial partition, a potential gain formulae that can be computed efficiently, and a multiple-vertex moving strategy for relaxing and enforcing balance constraints. The proposed algorithm has been implemented and applied to symbolic analysis of several practical analog integrated circuits. Experimental results are described and compared to the contour tableau method of Sangiovanni-Vincentelli, Chen and Chua, and the SCAPP algorithm of Hassoun and Lin. The resulting hierarchical symbolic analyzer outperforms SPICE in numerical evaluations for a number of large analog circuits.
© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Symbolic analysis; Partitioning; Determinant decision diagrams

1. Introduction

Symbolic analysis is to find the behavior or the characteristic of a circuit in terms of symbolic parameters. It is important for many applications such as optimum topology selection, design

☆Some preliminary results of this paper were presented at Asia and South Pacific Design Automation Conf. (ASP-DAC'99), Hong Kong, January 1999. This work was sponsored by US Defense Advanced Research Projects Agency (DARPA) under grant number F33615-96-1-5601 from the United States Air Force, Wright Laboratory, Manufacturing Technology Directorate.

*Corresponding author.
E-mail addresses: stan@ee.ucr.edu (S.X.-D. Tan), shi@ee.washington.edu (C.-J.R. Shi).

0167-9260/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0167-9260(03)00002-6

space exploration, behavioral model generation, and fault detection in analog design and test automation [1]. For example, the basic idea of topology selection is to generate a large number of alternative topologies consisting of elementary building blocks using some selection schemes (like artificial intelligence). For each generated topology, the formula associated to some performance criteria is derived automatically by symbolic analyzers. Then the performance criteria is optimized through repetitive evaluations of the formula and the best topologies can be selected based on the outcome of the optimization procedure.

Unfortunately, traditional symbolic analysis techniques can only be used to very small circuits. The two main limiting factors of for performing symbolic analysis on a large circuit are computer execution time and memory requirements. One way to cope with large circuits is by means of hierarchical decomposition. Hierarchical decomposition is to generate symbolic expressions in a nested form or sequence of expression form [2–4]. The decomposition can be performed on a graph representing the circuit of interest [3] or directly on circuit equations as done in *Network Formulation* method or determinant decision diagrams (*DDD*)-*based* method [2,4]. DDD-based approach shows promising results as it can exploit the sharing among different sub-expressions in a systematic way via DDD graph manipulations. The size of the resulting nested-form expressions from hierarchical decomposition, however, depends crucially on how a circuit is partitioned.

As the primary goal of the hierarchical analysis is to generate very compact nested-form expressions or sequence of expressions, an obvious optimization problem we need to solve is how to partition the circuits such that the resulting DDD-based nested symbolic expressions are minimized in terms of number of DDD vertices. The circuit partitioning problem arises virtually in all aspects of electronic design automation. Research works in the area of partitioning for circuit-level simulation and analysis includes the work of Sangiovanni-Vincentelli, Chen and Chua for node-tearing nodal analysis [5], Hassoun and Lin for symbolic analysis [6], and Yeh and Rao for parallel circuit simulation on multiprocessors [7].

The partitioning algorithm in [5] is based on a clustering-based technique (called contour tableau) to find the clusters (subcircuits) of closely coupled circuit elements, so that the size of every cluster is less than a user-specified upper bound, and the interconnection among clusters is minimized. The algorithm has linear time complexity, however, it may yield an unbalanced partition, and thus becomes less attractive in applications where a balanced partition is desired. The partitioning algorithm proposed in [6] partitions a circuit into a binary tree hierarchy by the permutations on the rows of the circuit matrix based on some interconnection information. The method, however, fails to consider balance constraints and explores only the binary tree circuit hierarchy. The method in [7] seeks a balanced bisection of a circuit by applying the Kernighan–Lin algorithm [8], which has quadratic time complexity. A balance partition is achieved by incorporating the balance constraints into the cost function. It has to be pointed out that so far no effective partitioning results have been reported for symbolic analysis on such less regular-structured circuits as $\mu A741$ Opamps [1].

Multiple-way partitioning for digital VLSI circuits have been intensively studied in the past. Some of works include recursive bipartitioning Kernighan–Lin (KL) algorithm [8], generalized Fiduccia–Mattheyses (FM) algorithm with look-ahead scheme (K-FM) [9], primal-dual method based on K-FM [10], dual net-based method [11], improved look ahead method [12], pairwise cell movement-based method [13] and simple greedy strategy [14]. Among them, K-FM, dual net-based, improved look-ahead method and simple greedy algorithms are *native* multiple-way

partitioning algorithms where all the nodes are directly shifted to different partitions instead of applying 2-way partitioning recursively. K-FM uses a $k(k-1)$ priority queues, one for each moving direction and the algorithm selects a move with highest gain from $k(k-1)$ queues subject to balance constraints. A sorted list is used in [10] for each partition to quickly select the best candidate move. Similar sorted list with better look-ahead scheme was used in [12]. In [14], the best move in partition is computed on the fly and is selected in a greedy way. However, all those methods mainly targeted at partitioning digital circuit netlists subject to balanced area constraints. No special attentions were paid to analog circuit partitioning for efficient circuit simulations.

In this paper, we formulate the partitioning problem such that the size of resulting DDD is a minimum based on a simple model of the DDD size for dense matrices. We adopt a multi-way partitioning strategy with balance constraints in terms of internal nets of subcircuits. It consists of two phases: (1) connectivity-oriented initial partitioning phase and (2) iterative refinement partitioning phase. The resulting algorithm has time complexity $O(n(k+\log_2 n + k^2 \log_2 k))$, where k is the number of parts a circuit to be divided and n is the total number of nodes in a circuit. To obtain a partition of tree hierarchy, the algorithm is simply used recursively until the upper bound constraint on the number of internal variables of leaf circuits is attained.

The rest of the paper is organized as follows: Section 3 formulates the partitioning problem for DDD-based hierarchical symbolic analysis. Section 4 presents a new multi-way multi-level partitioning heuristic. Section 5 describes some experimental results. Section 6 concludes the paper.

2. DDD-based hierarchical symbolic analysis

A linear(ized) time-invariant analog circuit can be described by a set of equations via modified nodal analysis in the following matrix form:

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

where $\mathbf{x}$ is a vector of circuit unknowns (node voltage and branch current variables), $\mathbf{A}$ is the modified nodal admittance matrix, or simply circuit matrix, and $\boldsymbol{b}$ is a vector of known values.

Consider a subcircuit with some internal structures and terminals as shown in Fig. 1. The circuit is divided into two device-disjointed parts and the variables $\mathbf{x}$ are divided into three disjoint groups: $\mathbf{x}^{\mathrm{I}}$, $\mathbf{x}^{\mathrm{B}}$, and $\mathbf{x}^{\mathrm{R}}$, where the sup-scripts I, B, R stand for, respectively, *internal* variables, *boundary* variables and the *rest* of variables. Then the system-equation set can be rewritten in the following form:

$$\begin{bmatrix} \mathbf{A}^{\mathrm{II}} & \mathbf{A}^{\mathrm{IB}} & \\ \mathbf{A}^{\mathrm{BI}} & \mathbf{A}^{\mathrm{BB}} & \mathbf{A}^{\mathrm{BR}} \\ & \mathbf{A}^{\mathrm{RB}} & \mathbf{A}^{\mathrm{RR}} \end{bmatrix} \begin{bmatrix} \mathbf{x}^{\mathrm{I}} \\ \mathbf{x}^{\mathrm{B}} \\ \mathbf{x}^{\mathrm{R}} \end{bmatrix} = \begin{bmatrix} \mathbf{b}^{\mathrm{I}} \\ \mathbf{b}^{\mathrm{B}} \\ \mathbf{b}^{\mathrm{R}} \end{bmatrix}. \tag{1}$$

The basic idea that underlines all the hierarchical analysis methods is to eliminate some equations from the equation-set above until the remaining set of equations involves only the desired

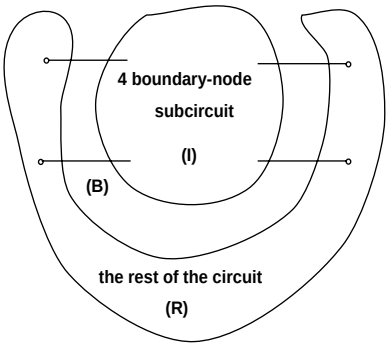

Fig. 1. Partition of a circuit.

variables. The resulting set of equations (1) by eliminating $\mathbf{x}^{\mathrm{I}}$ can be written as follows:

$$\begin{bmatrix} \mathbf{A}^{\mathrm{BB}*} & \mathbf{A}^{\mathrm{BR}} \\ \mathbf{A}^{\mathrm{RB}} & \mathbf{A}^{\mathrm{RR}} \end{bmatrix} \begin{bmatrix} \mathbf{x}^{\mathrm{B}} \\ \mathbf{x}^{\mathrm{R}} \end{bmatrix} = \begin{bmatrix} \mathbf{b}^{\mathrm{B}*} \\ \mathbf{b}^{\mathrm{R}} \end{bmatrix}, \tag{2}$$

where

$$\mathbf{A}^{\mathrm{BB}*} = \mathbf{A}^{\mathrm{BB}} - \mathbf{A}^{\mathrm{BI}}(\mathbf{A}^{\mathrm{II}})^{-1}\mathbf{A}^{\mathrm{IB}} \tag{3}$$

and

$$\mathbf{b}^{\mathrm{B}*} = \mathbf{b}^{\mathrm{B}} - \mathbf{A}^{\mathrm{BI}}(\mathbf{A}^{\mathrm{II}})^{-1}\mathbf{b}^{\mathrm{I}}. \tag{4}$$

In our application, $\mathbf{b}^{\mathrm{I}}$ is a zero vector, we rewrite (3) in the following expanded form:

$$a_{u,v}^{\mathrm{BB}*} = a_{u,v}^{\mathrm{BB}} - \frac{1}{det(\mathbf{A}^{\mathrm{II}})} \sum_{k_1,k_2=1}^{l} a_{u,k_1}^{\mathrm{BI}} \Delta_{k_2,k_1}^{\mathrm{II}} a_{k_2,v}^{\mathrm{IB}}, \tag{5}$$

where $u, v = 1, \ldots, k$, k is size of $\mathbf{A}^{\mathrm{BB}*}$, l is the size of $\mathbf{A}^{\mathrm{II}}$, $det(A^{\mathrm{II}})$ is the determinant of matrix A^{II}, $a_{u,v}^{\mathrm{BB}*}$ is the entry at row u and column v in $\mathbf{A}^{\mathrm{BB}*}$, $\Delta_{k_2,k_1}^{\mathrm{II}}$ is the first-order cofactor of A^{II} defined as $(-1)^{(k_2+k_1)}det(A_{k_2,k_1}^{\mathrm{II}})$ and $A_{k_2,k_1}^{\mathrm{II}}$ is obtained by eliminating row k_2 and column k_1 in A^{II}.

Note that we need the first-order cofactors $\Delta_{k_2,k_1}^{\mathrm{II}}$ only when both a_{u,k_1} and $a_{k_2,v}$ are non-zeros. In practice, given a good partitioning, a_{u,k_1} and $a_{k_2,v}$ are zero for most time due to the sparsity of A^{BI} and A^{IB}. The determinant $det(\mathbf{A}^{\mathrm{II}})$ and a few of its required first-order cofactors can be efficiently represented by a newly proposed special graph, DDD, described below.

A DDD example for a determinant is shown in Fig. 2. Its determinant is shown at the left-hand side.

Except for two special terminal vertices, namely the 0-*terminal* vertex and the 1-*terminal* vertex, each non-terminal vertex is labeled by a symbol in the determinant denoted by a_i, and a sign denoted by $s(a_i)$. Each non-terminal vertex *originates* two outgoing edges, called 1-*edge* and 0-*edge*. Each vertex a_i represents a symbolic expression $D(a_i)$ defined recursively as follows: $D(a_i) = a_i s(a_i) D_{a_i} + D_{\bar{a}_i}$, where D_{a_i} and $D_{\bar{a}_i}$ represent, respectively, the symbolic expressions presented by the vertices pointed by the 1- and 0-edges of a_i. A 1-*path* in a DDD corresponds a product term in the original DDD, which is defined as a path from the root vertex (A in our example) to the 1-terminal including all symbolic symbols and signs of the vertices that originate the 1-edges along

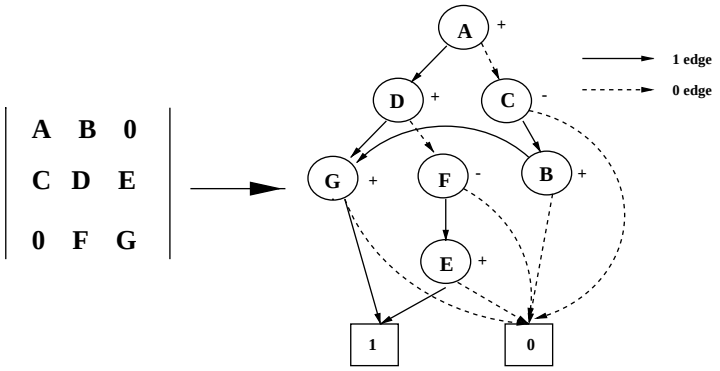

Fig. 2. A matrix determinant and its DDD.

the 1-path. In our example, there exist three 1-paths representing three product terms: ADG, $-AFE$ and $-CBG$. The root vertex represents the sum of these product terms.

3. Partitioning problem formulation

An analog circuit can be modeled as a hypergraph $G(V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_m\}$ and hyperedge set $E = \{e_1, e_2, \ldots, e_n\}$. Each vertex corresponds to a circuit device and each hyperedge represents a *node* or *net* in the circuit. A hyperedge e_j is a non-empty subset of two or more vertices, i.e. $e_j \in V$ and $|e_j|$ is the *size* of net e_j and $|e_j| \geqslant 2$.

The k-way hypergraph partitioning problem is to assign v_i, $i = 1, \ldots, m$ into k disjoint subsets $V_1, V_2, \ldots, V_k$. If $k > 2$, it is a *multi-way* partitioning problem. If subcircuit V_j, $j = 1, \ldots, k$ are further decomposed into smaller subcircuits, it is *multi-level* or *tree* partitioning; otherwise it is *two-level* partitioning.

A general circuit hierarchy is shown in Fig. 3. A subcircuit containing no subcircuits is a *leaf* subcircuit, otherwise it is a *middle* subcircuit. We further define the *span* of a net e_j, denoted as $span(e_j)$, as the number of the subcircuits e_j connects. If $span(e_j) > 1$, e_j is called *cut* net, otherwise it is an *internal* net. In nodal analysis, a node voltage variable corresponds to a node (net) in an analog circuit, therefore a hyperedge in the corresponding hypergraph. Similarly, a variable is an *internal* variable if its corresponding net is an internal net; otherwise, it is a *boundary* variable (its corresponding net is a cut net).

We first consider the effect of a circuit hierarchy on the representations of symbolic expressions by DDDs. From expression (5), we observe that for a middle circuit, the *internal* circuit matrix, A^{II}, of its subcircuit and a number of its first-order cofactors $\Delta^{\mathrm{II}}_{k_2,k_1}$ are required to be represented by DDDs. In order to reduce the number of those first-order cofactors and the fill-ins generated in the circuit matrix of the middle circuit during the suppressing of its subcircuits, the number of *boundary* variables, $a^{\mathrm{BI}}_{x,y}$ and $a^{\mathrm{IB}}_{x,y}$, of the subcircuit should be minimized. Boundary variables essentially are cut nets. Therefore, the total number of cut nets seen by each subcircuits should be minimized. This leads to the following problem: given a hypergraph $H(V, E)$, find $v[i] =$

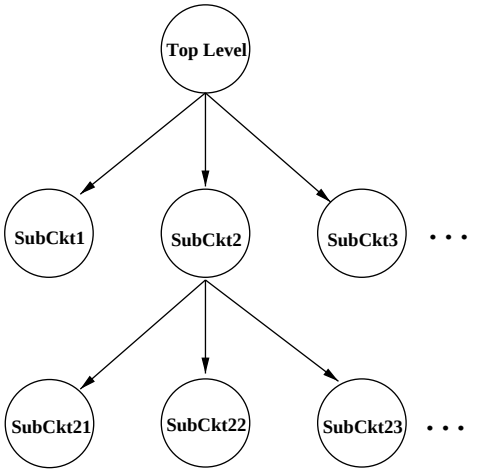

Fig. 3. Model of a circuit hierarchy.

$1, \ldots, k,\ i = 1, \ldots, |V|$ so as to

$$\textit{minimize} \sum_{e_j \in E} \textit{span}(e_j). \tag{6}$$

Min-span or *min-cut* partitioning commonly gives rise to very dense interconnection within each leaf subcircuit; thus the corresponding subcircuit matrices are dense. For a dense or full matrix, its DDD representation is still exponential. Fig. 4 shows the number of vertices of a DDD for the determinant of a full matrix and the number of product terms in the determinant vs. the size of the full matrix under the variable ordering given by the heuristic described in [15].

To simplify the problem formulation, we make the following two assumptions. First, we assume that the size of a DDD representing an $x \times x$ determinant, $det(T)$, of matrix T is an exponential function $g(x)$ in x, $g(x) = \alpha^x$, where base α is a positive constant. Our experimental results show that α is in the range $[2.3, 2.7]$ for all full matrices reported in Fig. 4. Second, we assume that the size of a DDD for $det(T)$ and the required first-order cofactors of $det(T)$ is $O(\alpha^x)$. The second assumption can be justified by the fact that the required first-order cofactors due to subcircuit suppression usually are few given a small number of boundary variables. A few of the required first-order cofactors can be represented by DDDs whose size is compared to that for $det(T)$, which is α^x, because of the sharing among the required cofactors and $det(T)$.

Under the two assumptions, the size of DDDs for hierarchical symbolic analysis of a circuit with k subcircuits will be bounded by

$$O(|DDD|) = O(\alpha^{(V_1)} + \alpha^{I(V_2)} + \cdots + \alpha^{I(V_k)} + \xi(S_{\text{cut}})), \tag{7}$$

where $I(V_i)$ is the number of internal variables in subcircuit V_i, i.e.

$$I(V_i) = \{e_j | e_j \subseteq V_i \quad \text{and} \quad \textit{span}(e_j) = 1\}, \tag{8}$$

and S_{cut} is the number of boundary variables, and $\xi(S_{\text{cut}})$ the size of DDDs due to the root circuit. Note that we have

$$I(V_1) + I(V_2) + \cdots + I(V_k) + S_{cut} = |E|. \tag{9}$$

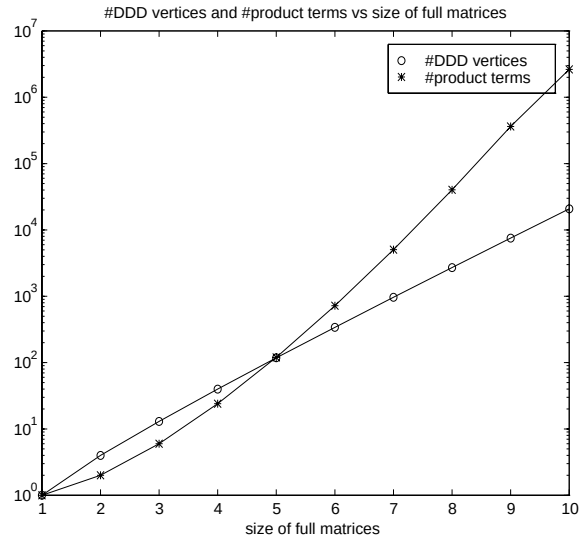

Fig. 4. #DDD vertices and #product terms vs. size of full matrices.

Since we take the minimum span objective, it is reasonable to assume the we should always get a very good quality partition and different good solutions should have similar total spans, which actually matters for the size of top DDDs. As a result, we can safely assume that S_{cut} will not change or change slightly under a given k, then $|DDD|$ will reach the minimum when

$$I(V_1) = I(V_2) = \cdots = I(V_k). \tag{10}$$

It is fairly easy to verify this when $k = 2$. Eq. (10) essentially calls for the balance on the internal variables of subcircuits. We solve the problem by introducing the following balance constraints:

$$I_{\max} - I_{\min} \leqslant \tau, \tag{11}$$

$$I_{\min} = min\{I(V_i)\}, \quad i = 1, \ldots, k,$$
$$I_{\max} = max\{I(V_i)\}, \quad i = 1, \ldots, k,$$

where τ is a measure of the offset from its balanced size (referred to as the *deviation factor*). The partitioning problem can be summarized as follows: Given a hypergraph $G(V, E)$, find a k-way partition of V subject to balance constraint (11), such that objective (6) is minimized.

For very large analog circuits, two-level partitioning may be still inadequate to reduce the overall DDD size. The reason is that S_{cut} will increase with the number of partitions - k, so does $\xi(S_{\text{cut}})$, which is the size of DDD vertices for the top circuit consisting of the cut nets. To keep the DDD size for each subcircuit and each non-leaf circuit manageable, multi-level partitioning is desired to bound the maximum number of internal variables of all the subcircuit and to limit the boundary variables of non-leaf circuits. The upper bound on the number of internal nets of leaf circuit j and on the boundary variables of non-leaf circuits can be expressed as

$$I(V_j) \leqslant \beta|E|, \quad S_{cut} \leqslant \beta|E|, \tag{12}$$

where β is a positive constant and $\beta \in [0, 1]$.

If the numbers of internal nets in some subcircuits or the number of top cut nets cannot be bounded by $\beta|E|$ properly selecting k, we have to use multiple level partitioning.

4. Multi-level multi-way balanced partitioning algorithm

We will use subcircuit and subset in an interchangeable way hereinafter for convenience.

Our approach to k-way constrained hypergraph partitioning is composed of initial partition construction and iterative refinement.

4.1. Initial partition construction algorithm

Iterative refinement algorithms such as Kernighan–Lin's algorithm usually trap to local minima [8,16]. A reasonable initial partition therefore is crucial to obtain good results. In addition, we need to find a feasible initial solution due to the balance constrains.

The initial partition construction is a connectivity-oriented clustering process where the k clusters are growing in such a way that each step the number of internal nets is maximized.

We begin with some definitions. A vertex v becomes *locked* after it is moved into a subset. A net is *Free* when all its vertices are free; otherwise, the net is *Assigned* if it still has free vertices and all the locked vertices are in one subset, or it is *Locked* if locked vertices on the net are in more than one subsets or all the vertices are locked. A *Locked* net, however, may still have free vertices.

We *move* a net (move all its free vertices) each time. The moving scheme first selects a *seed* (the first net) for each subset and then moves all the nets with the consideration of their interconnection with the *Locked* nets in all the subsets until all the vertices are locked, which marks the end of the moving process. The algorithm is described in Fig. 5. When a *Free* net is moved into a subset s, the number of *internal* nets of s is increased by at least one. This is also true for the *Assigned* net when it is moved into the subset in which it has locked vertices. Therefore, the move operations in steps (3) and (4) are restricted by the balance constrain (12) to guarantee a feasible solution. Once a net is moved and it becomes *Locked*, it will affect the status of its adjacent nets. We therefore have to adjust the status of those affected nets. In addition, $I_{\max}$ and $I_{\min}$ in the balance constraint (12) are also adjusted after a move operation. In step (3), the selection of subset for a *Free* net is based on its interconnection with subsets. Such an selection scheme, which emphasizes the clustering of the closely coupled vertices, typically yields better results.

If we assume the maximum number of vertices on a net and maximum number of nets incident on a vertex are bounded by a constant which is also significantly less than total number of vertices in the $G(V, E)$, we have following proposition.

Proposition 1. *The time complexity of* InitPart *algorithm is* $O(k|E| + |E| \log_2(|E|))$.

Proof. The sorting procedure in step (2) requires $O(|E| \log_2(|E|))$ to finish. In step (3), we visit each *Free* net just once. For each net, we need to compute its status, which takes $O(n_{\max,\text{net}}) = O(1)$, where $n_{\max,\text{net}}$ is the maximum number of vertices which a net connects. If it is a *Free* net, we calculate the changes of the number of internal nets for each subset when the net is moved into it.

```
InitPart (G(V, E), k)
  1. Set all the nets to Free and sort increasingly all the nets
     according to their size and the resulting sorted net list is
     L[i], i = 1, ..., |E|.
  2. for s = 1 to k do /* seed net for each cluster*/
        Find the first available Free net i (i.e. i is the small-
        est such that L[i].status = Free) and move net i into
        cluster s.
  3. for i = 1 to |E| do
        if (L[i].status = Free)
           Move net i into cluster s, subject to con-
           straint (12), such that the increase in the number
           of internal nets in s is a maximum among all the
           clusters.
  4. for i = 1 to |E| do
        if (L[i].status = Assigned)
           Move net i into the subset in which the net has
           the locked vertices subject to constraint (12).
  5. for i = 1 to |E| do
        if (net i has Free vertices)
           Randomly move all the free vertices in net i such
           that the net becomes a cut net.
```

Fig. 5. Initial partition construction.

Such a process will take $O(n^2_{\max,\text{net}} n_{\max,\text{ver}})$ to finish, as we must visit all the nets on all the vertices connected by the moved net and compute their status, where $n_{\max,\text{ver}}$ is the maximum of nets which a vertex connects. Hence the time complexity for this step is $O(kn^2_{\max,\text{net}} n_{\max,\text{vert}}|E|) = O(k|E|)$. Note that we treat item $O(n^2_{\max,\text{net}} n_{\max,\text{ver}})$ as a constant because $n_{\max,\text{net}}, n_{\max,\text{ver}}$ are topology parameters even though $O(n^2_{\max,\text{net}} n_{\max,\text{ver}})$ may become significant compared to $|E|$ in some situations. In step (4), we calculate status of each *Assigned net* and the change in terms of the number of internal nets in one subset, so the time complexity for this step is $O(n^3_{\max}|E|) = O(|E|)$. The last step can be finished in $O(n_{\max}|E|) = O(|E|)$ as we scan once all the vertices for each net that still has free vertices. By counting all the steps, the total time complexity of InitPart is $O(k|E| + |E| \log_2(|E|))$. □

4.2. Iterative refinement algorithm

An iterative refinement procedure is employed to further improve the quality of an initial partition. The algorithm is an extension of the iterative refinement heuristic due to Feduccia and Mattheyses (FM) [16] with an improved vertex-moving gain formula and a balance-relaxed vertex moving scheme described in this section.

The FMs algorithm begins with two initial subsets and proceeds in a series of *passes*. In each pass, it keeps moving vertices between two subsets until each vertex has been moved exactly once. After each pass, the best solution observed during the pass becomes the initial solution for a new pass. During each pass, the moved vertices are locked from further exchanging. The pass terminates when a pass does not improve upon the most recent solution.

4.2.1. Computation of gain and potential gain

Similar to FMs algorithm, we define the gain of moving a vertex into or out of a subset as the decrease in the span of a partition. Unless otherwise specified, we assume that a vertex v is moved from set A to set B, A and B are two subsets among k subsets and $v \in A$.

We further define an *incident number* of a net e with respect to a set of vertices A, denoted by $\alpha_A(e)$, as $\alpha_A(e) = |\{v|v \in A \text{ and } v \in e\}|$. A *binding force* of a net with respect to a set A, denoted by $\beta(e)$, is defined as $\beta_A(e) = \alpha_{A_F}(e)$ if $\alpha_{A_L}(e) = 0$ otherwise, ∞ if $\alpha_{A_L}(e) > 0$, where A_F and A_L denote the subsets that contains all *free* and *locked* vertices in A, respectively.

In order to efficiently calculate the *span*(e_j) of a net e_j, we divide the move operation of v into two steps:

1. The vertex v is selected from A and put into $\bar{A}$ ($\bar{A}$ is complement of A, $\bar{A} = V - A$); the gain of this move operation, denoted by $G^A_{\text{get}}(v)$, is

$$G^A_{\text{get}}(v) = |\{e \in E_v | \beta_A(e) = 1 \quad \text{and} \quad \beta_{\bar{A}}(e) > 0\}| - |\{e \in E_v | \beta_A(e) > 0 \quad \text{and} \quad \beta_{\bar{A}}(e) = 0\}|, \tag{13}$$

   where E_v denotes the set of nets that vertex v connects.
2. The vertex v is moved from the complement of B, $\bar{B}$ to B.[1] the gain, denoted by $G^B_{\text{put}}(v)$, is

$$G^B_{\text{put}}(v) = |\{e \in E_v | \beta_{\bar{B}}(e) = 1 \quad \text{and} \quad \beta_B(e) > 0\}| - |\{e \in E_v | \beta_{\bar{B}}(e) > 0 \quad \text{and} \quad \beta_B(e) = 0\}|. \tag{14}$$

Then the total gain of the move operation of v from A to B is given by

$$G_{\text{span}}(v) = G^A_{\text{get}}(v) + G^B_{\text{put}}(v). \tag{15}$$

To satisfy the balance constraint (12), we need to check the following two constraints at each moving step:

$$I^*_{\max} - \tau \leqslant I(A) - I_A(v) \quad \text{for set A}, \tag{16}$$

$$I^*_{\min} + \tau \geqslant I(B) + S_B(v) \quad \text{for set B}, \tag{17}$$

where $I_A(v)$ is the number of all the internal nets of A that become cut net after v is moved out of A, and where $S_B(v)$ is the number of all the cut nets that become internal nets of B after v is moved into B. $I^*_{\max}(I^*_{\min})$ are maximum (minimum) number of internal nets among all the subcircuits after v is moved from A into B, i.e.

$$I^*_{\max} = max\{I(V_i) \text{ for } V_i \neq A, (I(A) - I_A(v)) \text{ for } V_i = A\}, \quad i = 1, \ldots, k, \tag{18}$$

$$I^*_{\min} = min\{I(V_i) \text{ for } V_i \neq B, (I(B) + S_B(v)) \text{ for } V_i = B\}, \quad i = 1, \ldots, k. \tag{19}$$

Because each vertex v has $k-1$ moving directions thus $k-1$ gain values for each direction. We select the direction with the highest gain each time. So a vertex is sorted according to its highest gain. Once we select a vertex, we also know its destination.

One way to improve the partitioning quality is to break the tie situation where two vertices have the same gain [17–19]. We solve this problem by introducing a penalty function devised for multi-way partitioning into the vertex gain as a *potential* gain. Let $G_P(e)$ denote the potential gain of net e imposed on all its connected vertices. The potential gain also consists of two parts which

[1] Note that $\bar{A} \neq \bar{B}$, but their differences have no impact on the gain calculations.

correspond to the two aforementioned steps in a move operation:

$$G_{\mathrm{P}}(e) = G^{A}_{\mathrm{Pget}}(e) + G^{B}_{\mathrm{Pput}}(e), \tag{20}$$

where

$$G^{A}_{\mathrm{Pget}}(e) = \begin{cases} 0 & \text{if } \beta_A(e) = |e| \text{ or } \beta_A(e) = 0, \\ -p & \text{if } \beta_A(e) = \infty, \\ -p \times w(e, A) & \text{if } 0 < \beta_A(e) < |e| \end{cases} \tag{21}$$

and

$$G^{B}_{\mathrm{Pput}}(e) = \begin{cases} 0 & \text{if } \beta_B(e) = |e| \text{ or } \beta_B(e) = 0, \\ p & \text{if } \beta_B(e) = \infty, \\ p \times w(e, B) & \text{if } 0 < \beta_B(e) < |e| \end{cases} \tag{22}$$

and $w(e, X) = \alpha_X(e)/n_{\max}$ if $|e| \leqslant n_{\max}$; and $w(e, X) = \alpha_X(e)/|e|$ otherwise. p is a constant, and $n_{\max}$ is a user-specified upper bound on the number of vertices which a net connects. Both $G^{A}_{\mathrm{Pget}}(e)$ and $G^{B}_{\mathrm{Pput}}(e)$ favor the vertices on the cut nets which likely become internal nets of B if the vertices are moved. The total gain for the whole move operation is expressed as

$$G_{AB}(v) = G_{\mathrm{span}}(v) + \sum_{e \in E_v} (G^{A}_{\mathrm{Pget}}(e) + G^{B}_{\mathrm{Pput}}(e)). \tag{23}$$

Now we illustrate the gain calculation by means of a three-way partitioning example shown in Fig. 6. Let $n_{\max} = 5$ and $p = 3$, and all the vertices are free. Without considering the potential gain, moving vertex a, f and d to some subsets will reduce the span of the partition by one (non-potential gain equals 1). The potential gain for moving vertex a from A to B is calculated as follows: $G^{A}_{\mathrm{Pget}}(e_1) = 0$ and $G^{B}_{\mathrm{Pput}}(e_1) = 0$, $G^{A}_{\mathrm{Pget}}(e_2) = -3/5$ and $G^{B}_{\mathrm{Pput}}(e_2) = 6/5$, $G^{A}_{\mathrm{Pget}}(e_3) = -3/5$ and $G^{B}_{\mathrm{Pput}}(e_3) = 3/5$. Hence, the total gain for this move is $G_{AB}(a) = 1 + 3/5 (= 0 + 0 - 3/5 + 6/5 - 3/5 + 3/5) = 8/5$. On the other hand, the potential gain of moving f to either A or B and of moving d to either A and C is zero. Therefore, moving a from A to B has the highest total gain. Intuitively, the advantage of moving a over the other two moves is that *cut* net e_2 which is otherwise difficult to be removed, is first removed when a is moved first.

4.3. Balance constraint relaxation

Another problem with FMs algorithm is that moving a vertex is only feasible if the move operation does not violate the balance constraints. Such a moving scheme will severely confine the solution space especially when the constraints are strict [12,20]. This problem can be alleviated by temporarily relaxing the balance constraints and allowing a sequence of move operations of vertices, called *macro-step*, to be carried out as long as the balance constraints are restored after the macro-step.

Each macro-step begins with moving a vertex v from A to B without considering the balance constraint (17). Nevertheless, the move is still subject to constraint (16). The reason is that we should not limit the increase of the internal nets in subset B, while we limit the decrease of the internal nets in subset A. This constraint-relaxation scheme is consistent with the min-span partitioning objective. We observe that after the move operation, $I[B]$ always increases. In case the

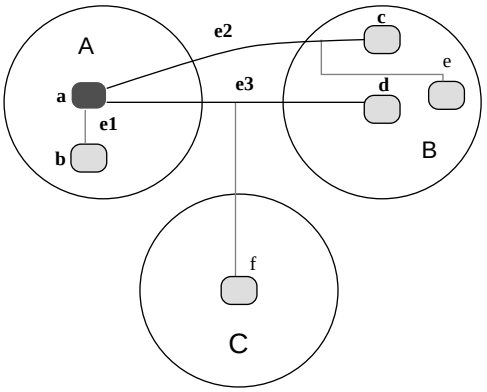

Fig. 6. Illustration of gain calculation in multi-way partitioning.

move operation causes the violation of the balance constraint in B, we just move some vertices out of B to restore the balance.

More specifically, let $F = \{v_1, v_2, \ldots, v_m\}$ be a set of free vertices. Let $G_{\max}[i]$ and $I[i]$ are the maximum gain of all the free vertices in subset i and the number of internal nets in subset i, respectively. The new balance-relaxed multi-way partitioning algorithm, BALRELAXMP, is described in Fig. 7.

A similar constraint-relaxation scheme has been shown to be effective on the digital circuit partitioning with the balanced area constraints [12].

4.3.1. Implementation and time complexity

In our implementation, the bucket data structure introduced in [16] is used. We modify this date structure to account for multi-way partitioning. Because the total gain of a vertex is still bounded after the introduction of the potential gains (21) and (22), the bucket sorting scheme is still applicable to our application. With this, re-sorting all the affected free vertices caused by moving a vertex will still take a constant time.

Under the same assumption that the number of vertices per net and the number of connections per vertex are limited by a constant, $n_{\max}$, we have the following proposition:

Proposition 2. *Given a hypergraph $H(V, E)$, the time complexity of* BALRELAXMP *for k-way partitioning is $O(|E|k^2 \log_2(k))$.*

Proof. Let $n(i)$ denote the number of vertices on net e_i, then the total number of gain calculations due to the vertices on net i during a pass is

$$n(i) + (n(i) - 1) + \cdots + 1 = O(n^2(i)). \tag{24}$$

Hence, the total number of gain calculations in a pass is given by

$$O\left(\sum_{i=1}^{|E|} n^2(i)\right), \tag{25}$$

```
BalRelaxMP(H(V, E))
  while(|F| ≠ 0) do /* begin macro-step */
    1. Sort G_max[i], i = 1, ..., k.
       for j = 1 to k do
         Try to select a free vertex v with the highest gain
         from subset j subject to constraint (16), and move
         v to its destination, subset t.
         if the free vertex v can be founded
             goto step 2.
       Current pass stops.
    2. while (constraint (17) is violated) do
         move a free vertex v_x with the highest gain out of
         subset t subject to constraints (17) and (16).
    3. Lock all the moved vertices and add them into current
       macro-step; record the current span size.
```

Fig. 7. Balance-relaxed multi-way partitioning.

which also equals

$$\sum_{i=1}^{|E|} n^2(i) \leqslant \sum_{i=1}^{|E|} n_{\max}^2 = |E| n_{\max}^2. \tag{26}$$

As a result, the time complexity of the algorithm is $O(|E| n_{\max}^2) = O(|E|)$. Within a vertex gain calculation, $k-1$ gain values for each moving direction must be recalculated as each vertex has $k-1$ move directions. Such a process takes $O(k)$ to complete. In BalRelaxMP algorithm, each macro-step begins by sorting k subsets. In the worst case, every macro-step corresponds to just one vertex move. Sorting process has the time complexity of $O(k \log_2 k)$. Therefore, the total time complexity is $O(|E| k^2 \log_2 k)$. □

Considering both initial partition construction and iterative refinement, the time complexity of the entire partitioning process is

$$O(|E|(k + \log_2(|E|) + k^2 \log_2(k))). \tag{27}$$

4.4. Multi-way tree partitioning

Two-level partitioning can be easily extended to tree partitioning by recursively applying the multi-way algorithm, until constraints (12) on the number of the internal nets of all the leaf circuits are satisfied.

In our implementation, multi-level partitioning is treated as a special sequence of multi-way partitioning processes where each multi-way partitioning is performed such that vertices are moved only from the subsets at a particular hierarchical level, and all the other vertices are locked during that pass. In this way, the gain calculations (23), and balance constraints (16) and (17) will still remain valid.

5. Experimental results

The proposed balanced multi-level multi-way partitioning algorithm has been implemented and linked to a DDD-based symbolic analyzer SCAD[3] [15,4]. The results from two examples are described. All the results are obtained on a SUN SPARCstation 5 with 32 MB memory.

For non-linear integrated circuits, DC analysis is first performed using SPICE, and the resulting small signal models from the output of SPICE are used in symbolic analysis. For the completeness, the small signal models used for bipolar and MOS transistors are described in Figs. 8(a) and (b), respectively.

5.1. $\mu A741$ opamps

The first example is the bipolar $\mu A741$ circuit with 26 transistors and 11 resistors shown in Fig. 9(a). We first perform 2-level multi-way partitioning of $\mu A741$. The total number of nets for $\mu A741$ is 24. For small-signal AC analysis, the power and ground nets are the reference node in the nodal formulation method. They will not appear in the circuit matrix, and ignored by

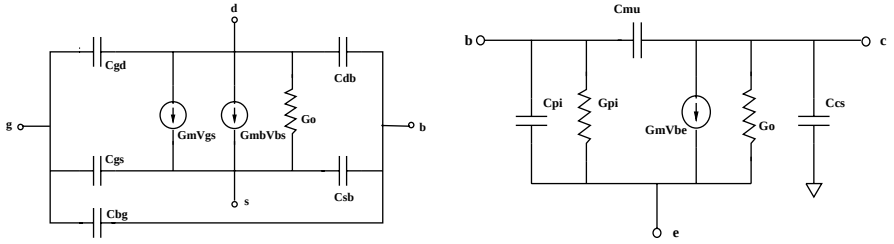

Fig. 8. (a) MOSFET small signal model and (b) bipolar transistor model.

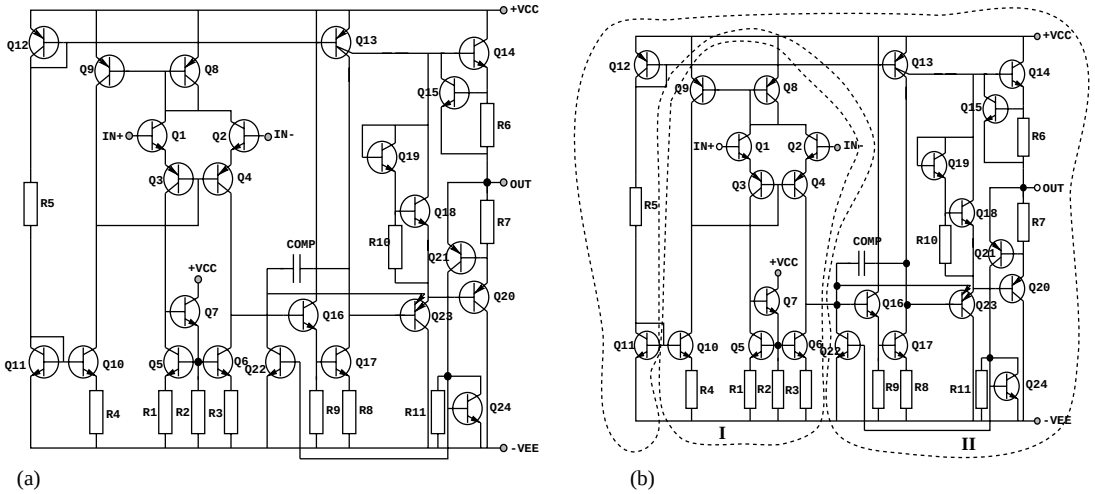

Fig. 9. (a) Bipolar $\mu A741$ opamp and (b) a 2-way partitioned $\mu A741$.

partitioning. All the nets corresponding to the circuit inputs and outputs are always cut nets. We select the *deviation factor*—τ to be 4.

Figs. 9(b), 10(a) and (b) show, respectively, the results of 2-, 3- and 4-way balanced partitioning of $\mu A741$, where each partitioned subcircuit is marked by an index (I–IV). Table 1 summarizes the statistics of these partitions. Columns 1, 2 and 3 list, respectively, the number of subcircuits, the span and the number of cut nets. Columns 4–7 list, respectively, the number of internal nets in each subcircuit and their corresponding DDD size. Column 8 describes the number of DDD vertices in the root circuit, $|topD|$; the last column is the total number of DDD vertices, $|totalD|$. The last two columns give the CPU times for evaluating 1000 frequency points from our program SCAD[3] and SPICE for each partitioned circuit. The CPU times are collected on a Linux system with 450 MHz Intel-Celeron CPU and 200 MB memory. We have the following observations:

- The total number of DDD vertices used for representing all the subcircuits decreases from 837 in the balanced 2-way partitioning to 239 in the balanced 4-way partitioning. Meanwhile, the number of DDD vertices used for the root circuit increases from 20 to 114. The total number of DDD vertices decreases as k increases from 2 to 4. If we further increase k, more nets will be cut nets and the size of the DDD for the root circuit will increase rapidly, and dominate the overall DDD size.

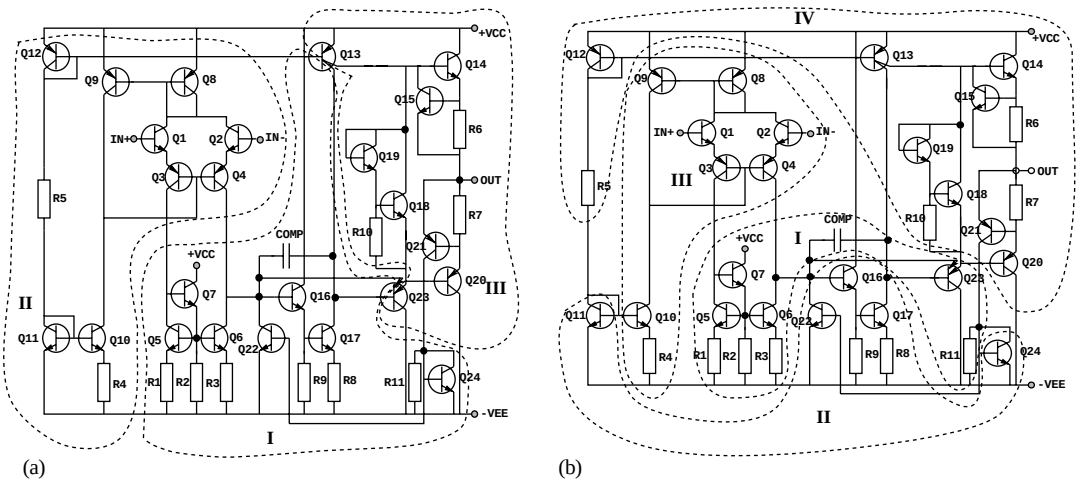

Fig. 10. (a) A 3-way partitioned $\mu A741$ and (b) a 4-way partitioned $\mu A741$.

Table 1
Statistics of 2-level multi-way partitioning on $\mu A741$

k	Span	#Cut	#Internal nets/$\|DDD\|$				$\|topD\|$	$\|totalD\|$	SCAD[3]	SPICE
			I	II	III	IV				
2	8	4	9/260	10/557	—	—	20	837	2.01	1.60
3	13	7	5/29	6/79	5/55	—	80	241	0.68	1.61
4	18	8	3/5	2/5	5/53	5/57	114	239	0.69	1.69

- As we have reported in [15], without partitioning and hierarchical symbolic analysis, the total number of DDD vertices to represent the $\mu A741$ is 6654, where the number of product terms in fully expanded classical symbolic analysis is 119011. Thus, hierarchical symbolic analysis leads to a very compact behavioral model (237 vs. 6654) thanks to automatic balanced partitioning. Since the number of multiplications is linear in the DDD size, hierarchical symbolic analysis with automated partitioning speeds up canonical symbolic analysis (without partitioning) by a factor of 28, where canonical symbolic analysis already boasts of several orders of magnitude speedup over fully expanded symbolic analysis (6654 multiplications vs. 119011 product terms).
- Smaller number of DDD nodes does results small CPU time for evaluating the resulting symbolic expressions than SPICE. The speedup will become more significant if multi-level partitioning is used as shown in Table 2.

We further perform a 3-level 2-way partitioning of $\mu A741$ based on the hierarchy tree shown in Fig. 11(b). The third-level partitioning is based on the 2-level 2-way partitioning and the resulting tree hierarchy is shown in Fig. 9(b). The resulting partitioning is shown in Fig. 11(a).

Table 2 summarizes the partitioning statistics. We can see that hierarchical symbolic analysis with automated 3-level 2-way partitioning reduces the number of DDD vertices from 6654 to 117! Since at most one multiplication is needed for one vertex, the total number of multiplications is bounded by 117. To compare with best-known hierarchical symbolic analyzer—SCAPP [7], row *SCAPP* lists the best result from SCAPP with automated partitioning, where #*mul* and #*add* are number of multiplications and additions, respectively. The last two rows give the CPU times for evaluating 1000 frequency points from our program SCAD^3 and SPICE for the partitioned circuit. Hence the speedup of numerical evaluation over SPICE is further improved compared with the 2-level partitioned circuits.

Table 2
Statistics for 3-level, 2-way partitioning on $\mu A741$

Leaf subcircuits	I1	I2	II1	II2
$\|DDD\|$	10	36	23	6
internal nets	3	4	4	3
Middle subcircuits	I		II	
$\|DDD\|$	4		18	
internal nets	2		3	
cut nets	5		6	
Root $\|DDD\|$	20			
Total $\|DDD\|$	117			
SCAPP	#mul:182, #add: 357			
SCAD^3	0.47 s			
SPICE	1.57 s			

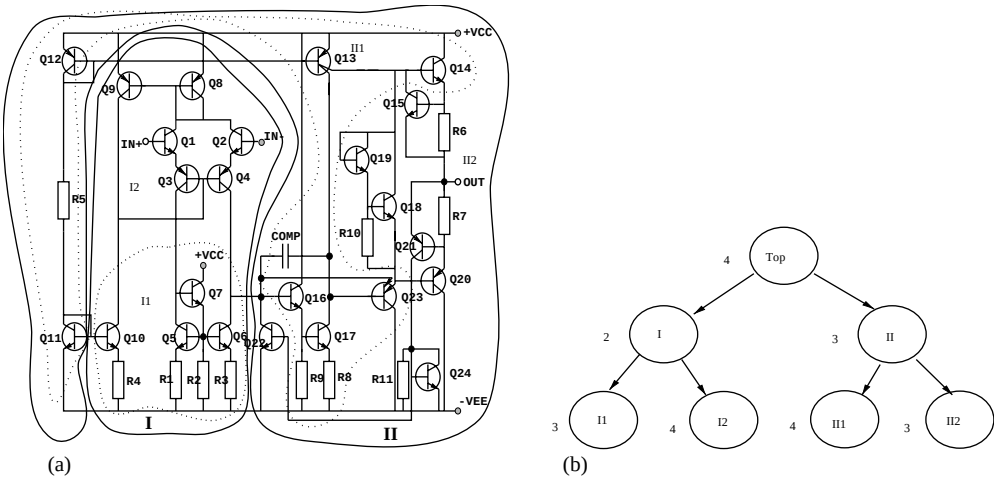

Fig. 11. (a) A 3-level, 2-way partitioned $\mu A741$ and (b) a 3-level, 2-way partition of $\mu A741$.

5.2. *A band-pass filter*

The second example is a band-pass filter, shown in Fig. 13(a), where each Opamp subcircuit is a miller-compensated two-stage MOS circuit, shown in Fig. 12.

An important feature of the proposed partitioning algorithm is its capability in performing higher level partitioning based on the existing subcircuit definitions. For example, we can view each Opamp circuit as a 3-terminal block in the band-pass filter circuit shown in Fig. 13(a) and perform partitioning at the top circuit level. The resulting partitioning is shown in Fig. 13(b). This contrasts with the partitioning algorithm in SCAPP where the automatic partitioning has to be carried out on the flattened circuits.

Table 3 summarizes the partitioning statistics along with the comparison with the best results from SCAPP with automated partitioning. Column *Op* shows the number of DDD vertices used to represent the Opamp subcircuit. Columns 3–5 describe the DDD size for each middle subcircuit. The last two columns give the DDD size at the root and the total number of DDD vertices. Rows 2 and 3 describe, respectively, the DDD size and number of internal nets for each subcircuit. Row $|DDD|$ (w/o) shows the DDD size without partitioning. Row *SCAPP* shows the best result from SCAPP with automated partitioning. Note that the difference in the DDD size with and without partitioning is quite marginal for this example, because the cascade structure in the band-pass circuit has been exploited by DDDs [15]. The last two rows still give the CPU times for evaluating 1000 frequency points from our program SCAD^3 and SPICE for the partitioned circuit. Comparing with Table 1, it appears that with larger circuits, the speedup of the new algorithm over SPICE will increase using the same level partition scheme.

5.3. *Effectiveness of* InitPart

We then study the impact of the proposed initial partition construction algorithm (InitPart) on the final quality of a partition. For comparison, we implement another initial partition

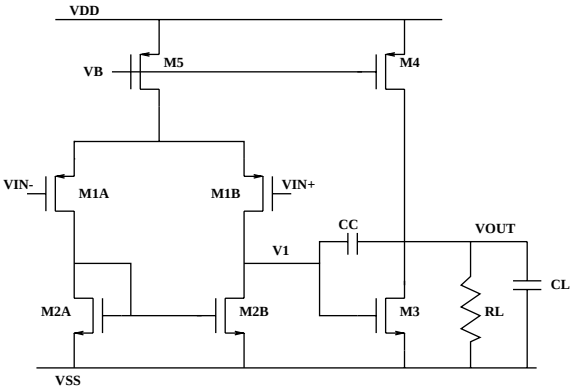

Fig. 12. Miller-compensated two-stage Opamp.

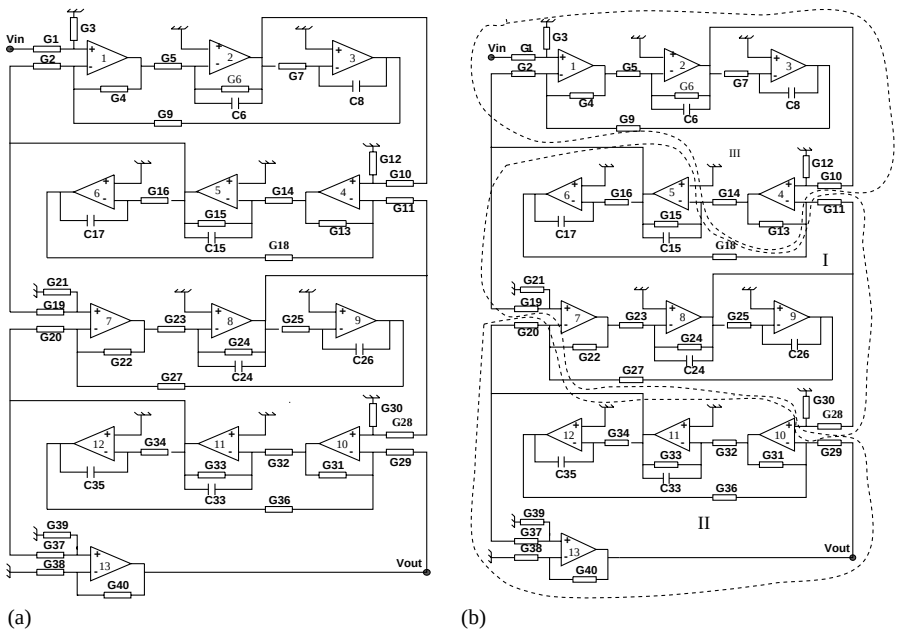

Fig. 13. (a) An active RC band-pass filter and (b) a 3-way partitioning of the band-pass filter.

Table 3
Statistics of 3-way tree partitioning of the band-pass filter

Subcircuits	Op	I	II	III	#top	#total
$\|DDD\|$	17	168	140	166	173	664
#internal net	3	8	8	9	7	—
$\|DDD\|$ (w/o)	767					
SCAPP	#mul:1639, #add:2479					
SCAD^3	1.96 s					
SPICE	5.21 s					

Table 4
Comparison between two initial partition construction algorithms

	InitPart			RandInitPart		
Span	Avg	Std	Min	Avg	Std	Min
$\mu A741$	26.5	3.58	21	28.3	3.76	23
Band-pass	27.2	5.7	18	32.4	8.9	21
#cut nets	Avg	Std	Min	Avg	Std	Min
$\mu A741$	9.5	1.2	7	12.8	1.9	9
Band-pass	10.2	1.5	8	16.5	5.1	9

construction algorithm, RandInitPart, which randomly move each net into a subset as long as no violations of balance constraints occur. The results from these two algorithms are then fed into the iterative refinement algorithm.

The experiment is conducted by performing 4-way partitioning on both the $\mu A741$ circuit and the band-pass filter for 20 runs with *deviation factor* $\tau = 4$. The total number of nets in $\mu A741$ and band-pass circuit are 26 and 33, respectively (we count power and ground nets in both circuits). Table 4 shows the statistics about the experimental results.

Rows 3–4 are the average (*avg*), the standard deviations (*std*) and the minimum (*min*) of the span of the resulting partitions by the InitPart algorithm (in columns 2, 3 and 4) and by the RandInitPart algorithm (in columns 5, 6 and 7) for circuits $\mu A741$ and band-pass filter, respectively. Rows 6 and 7 show the results in terms of cut nets for both circuits.

We have the following observations. First, the InitPart algorithm outperforms the RandInitPart algorithm in both circuits. Second, the InitPart algorithm leads to more stable results in the 20 runs on both circuits. This is reflected in the standard deviations of both span and numbers of cut nets in the resulting partitions. Hence, given the improved average results and reduced volatility, the InitPart algorithm can lead to much more saving in CPU time since we can obtain similar results with less number of trials in practice.

5.4. Comparison with contour tableau

Finally, we compare our method with the contour tableau method [5] on real analog circuits. In the contour tableau method, the partitioning procedure is directed performed on a graph $\Omega(N, B)$ representing an analog circuit, where N is a set of nodes in the circuit and B is a set of edges representing circuit elements connecting nodes in the analog circuits. A clustering procedure is performed by selecting one node at a time. The cluster is identified such that the minimum adjacent nodes outside the cluster is observed before the user-specified upper bound on the number of nodes, n_{up}, in the cluster is reached. This method is efficient at finding closely coupled circuit elements [5]. Unfortunately it may yield very unbalanced partitions for some circuit structures as shown in the following example.

We consider the band-pass filter circuit shown in Fig. 13(a). We select several upper bounds, n_{up}, on the clusters and the results are shown in Table 5.

Column 2 is the number of clusters (subcircuits) generated. Columns 3 and 4 are the span and the number of cut nets in resulting partitions. Column 5 is the size of resulting DDDs.

Table 5
Partitioning results on band-pass filter by the contour tableau method

n_{up}	k	Span	#Cut	$\|DDD\|$	#nodes in each cluster					
					I	II	III	IV	V	VI
5	5	22	11	601	5	1	5	5	5	—
7	6	26	9	927	7	1	2	5	7	1
9	5	17	5	568	10	2	10	2	3	—
11	5	17	5	568	10	2	10	2	3	—
13	5	17	5	568	10	2	10	2	3	—
15	3	11	5	837	10	2	15	—	—	—
—	5	25	8	509	3	2	7	7	5	—

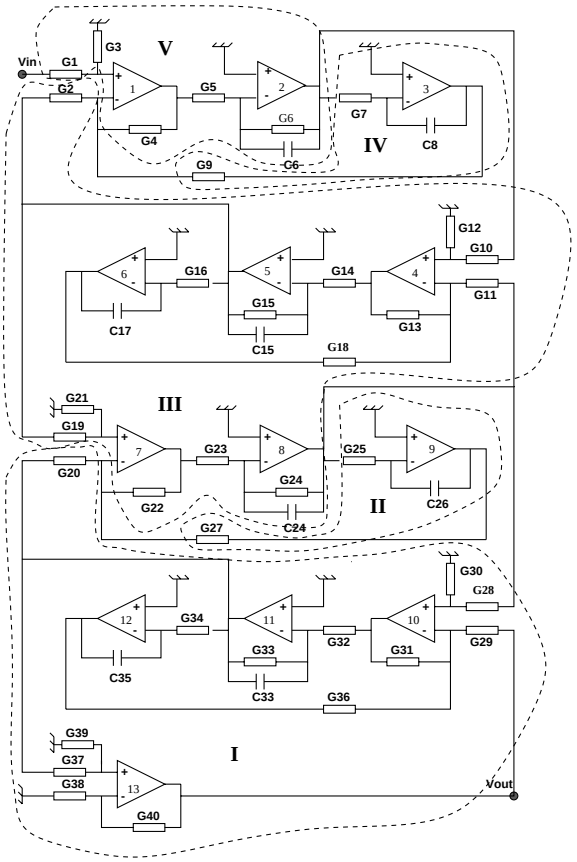

Fig. 14. A 5-way partition of band-pass filter by the contour tableau method.

Columns 6–11 are the number of nodes (also the number of internal nets) in each cluster. The last row show the results from the new partitioning algorithm. It is worth pointing out that when $n_{up} = 9, 11, 13$, the differences between the maximum number of internal nets and the minimum

number of internal nets are quite remarkable. The partitions with $n_{up} = 9, 11, 13$ are drawn in the Fig. 14. The reason for such unbalance is that after a cluster has been identified and removed from the graph along with the cut nets, the remaining circuit will becomes a set of separated graphs (the remaining graph is not a single connected graph). If some of the separated graphs are very small, the algorithm will soon identify them as clusters and thus results in the unbalanced partitions. In our example, after cluster I is identified and removed, cluster II becomes a separated graph. This is also true for cluster IV after cluster III is identified. We note that this inherent problem cannot be solved by simply increasing n_{max} as shown in Table 5. In contrast, results from a more balanced partition obtained by the new partitioning algorithm, shown in the last row of the table, give the smallest DDD size.

Symbolic hierarchical decomposition methods mainly aims at speeding up applications where repeated evaluation of a circuit performance (symbolic expressions) is required. The major advantage of hierarchical decomposition is its ability to process fairly large analog circuits. On the other hand, non-decomposition symbolic circuit analysis with approximation strategies still suffers the circuit-size limitation problem. It is encouraging to see some promising results from combining both hierarchical decomposition and approximation strategies to deal with more large analog circuits [21]. We like to look at how to apply approximation during hierarchical decomposition in our future investigations.

6. Conclusions

An efficient algorithm for balanced multi-level multi-way partitioning of large analog circuits is presented. Similar to many recent algorithms for multi-way partitioning, for example [17,9], the proposed algorithm is based on iterative vertex moving scheme due to Fiduccia and Mattheyses. The novelty is the introduction of an effective potential gain and multiple-vertex moving strategy for relaxing balance constraints. In this paper, we have also described its application to hierarchical analog symbolic analysis, and shown its superiority over the best analog symbolic analysis program SCAPP. The new partitioning scheme also compares favorably with the contour tableau method in partitioning analog circuits for DDD-based hierarchical symbolic analysis. The resulting hierarchical symbolic analyzer outperforms SPICE in numerical evaluations for a number of large analog circuits.

Acknowledgements

The authors thank three reviewers for their valuable and constructive comments of this work that improve the representation of this paper.

References

[1] G. Gielen, P. Wambacq, W. Sansen, Symbolic analysis methods and applications for analog circuits: a tutorial overview, Proc. IEEE 82 (2) (1994) 287–304.

[2] M.M. Hassoun, P.M. Lin, A hierarchical network approach to symbolic analysis of large scale networks, IEEE Trans. Circuits Systems 42 (4) (1995) 201–211.
[3] J.A. Starzky, A. Konczykowska, Flowgraph analysis of large electronic networks, IEEE Trans. Circuits Systems 33 (3) (1986) 302–315.
[4] X.-D. Tan, C.-J. Shi, Hierarchical symbolic analysis of large analog circuits via determinant decision diagrams, IEEE Trans. Comput. Aided Design Integrated Circuits Systems 19 (4) (2000) 401–412.
[5] A. Sangiovanni-Vincentelli, L.-K. Chen, L.O. Chua, An efficient heuristic cluster algorithm for tearing large-scale networks, IEEE Trans. Circuits Systems CAS-24 (12) (1977) 709–717.
[6] M.M. Hassoun, P.M. Lin, An efficient partitioning algorithm for large-scale circuits, in: Proceedings of the IEEE International Symposium on Circuits and Systems, 1990, pp. 2405–2408.
[7] D.C. Yeh, V.B. Rao, Partitioning issue in circuit simulation on multiprocessors, in: Proceedings of the IEEE International Symposium on Circuits and Systems, 1988, pp. 300–303.
[8] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (1970) 291–307.
[9] L.A. Sanchis, Multiple-way network partitioning, IEEE Trans. Comput. C38 (1) (1989) 62–81.
[10] C.W. Yeh, C.K. Cheng, T.T. Lin, A general purpose multiple-way partitioning algorithm, in: Proceedings of the 36th ACM/IEEE Design Automation Conference, 1991, pp. 421–426.
[11] J. Cong, W. Labio, N. Shivakumar, Multi-way VLSI circuit partitioning based on dual net representation, IEEE Trans. Computer-Aided Des. Integrated Circuits Systems 15 (4) (1996) 396–409.
[12] X.-D. Tan, J.-R. Tong, P.-S. Tang, F. Lombardi, An efficient multi-way algorithm for balanced partitioning of VLSI circuits, IEEE Internat. Conf. Computer Design (ICCD), 1997, pp. 608–613.
[13] J. Cong, S.K. Lim, Multiway partitioning with pairwise movement in: IEEE International Conference on Computer Aided Design (ICCAD), 1998, pp. 512–516.
[14] G. Karypis, V. Kumar, Multilevel k-way hypergraph partitioning, in: Proceedings of the 36th ACM/IEEE Design Automation Conference, June 1999, pp. 343–348.
[15] C.-J. Shi, X.-D. Tan, Canonical symbolic analysis of large analog circuits with determinant decision diagrams, IEEE Trans. Comput.-Aided Design Integrated Circuits Systems 19 (1) (2000) 1–18.
[16] C.M. Fiduccia, R.M. Mattheyses, A linear time heuristic for improved network partitions in: Proceedings of the 19th ACM/IEEE Design Automation Conference, 1982, pp. 175–181.
[17] S. Dutt, W. Deng, A probability-based approach to VLSI circuit partitioning, in: Proceedings of the 33th IEEE/ACM Design Automation Conference, 1996, pp. 100-1-5.
[18] B. Krishnamurthy, An improved min-cut algorithm for partitioning VLSI networks, IEEE Trans. Comput. C-33 (1984) 438–446.
[19] C. Sechen, D. Chen, An improved objective function for mincut circuit partitioning, in: Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD) 1994, pp. 501–505.
[20] J. Cong, L. Hagen, A. Kahng, Net partitioning yields better module partitions, in: Proceedings of the 29th IEEE/ACM Design Automation Conference, 1992, pp. 47–52.
[21] O. Guerra, E. Roca, F.V. Fernández, A. Rodríguez-Vázquez, Approximate symbolic analysis of hierarchically decomposed analog circuits, Analog Integrated Circuits Signal 31 (2002) 131–145.