jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 3, PP. 285–295
Manuscript received July 4, 2012; revised August, 2012. DOI: 10.2478/v10177-012-0040-4

DC Large-Scale Simulation of Nonlinear Circuits on Parallel Processors

Diego Ernesto Cortés Udave, Jan Ogrodzki, and Miguel Angel Gutiérrez de Anda

***Abstract*—Newton-Raphson DC analysis of large-scale nonlinear circuits may be an extremely time consuming process even if sparse matrix techniques and bypassing of nonlinear models calculation are used. A slight decrease in the time required for this task may be enabled on multi-core, multithread computers if the calculation of the mathematical models for the nonlinear elements as well as the stamp management of the sparse matrix entries is managed through concurrent processes. In this paper it is shown how the numerical complexity of this problem (and thus its solution time) can be further reduced via the circuit decomposition and parallel solution of blocks taking as a departure point the Bordered-Block Diagonal (BBD) matrix structure. This BBD-parallel approach may give a considerable profit though it is strongly dependent on the system topology. This paper presents a theoretical foundation of the algorithm, its implementation, and numerical complexity analysis in virtue of practical measurements of matrix operations.**

***Keywords*—circuit simulation, parallel computation, DC analysis, circuit decomposition.**

I. Introduction

Although the idea of having multiple processors for solving in a concurrent way computational problems which can be conveniently broken down into smaller tasks is rather old, relatively recent technological advancements have made possible the development of multi-core processors which are available to the average consumer. These processors have found their way in many engineering applications. Evidently, circuit simulation could not be left behind.

Long before the arrival of multi-core processors, several researchers were concerned with the applicability of parallel computers for the simulation of electronic circuits. Reference [1] describes the first advances made in the area for computing architectures with shared memory and a limited number of processors. In [2], a complete discussion on specific techniques which may be exploited to speed up the simulation of electronic circuits in a parallel computing environment is presented. Recent developments in the state of the art of the last two years may be found in [3]–[8].

Specifically, two strategies can be considered to accelerate the simulation process of a given circuit, namely, the use of parallel threads for:

1) the setup and solution of the linearized modified nodal equations required for the analysis,
2) the partitioning of the original simulation problem into a collection of simpler problems which can be concurrently solved.

The most essential routine in any circuit simulator devoted to the analysis of nonlinear circuits is the determination of at least a single DC solution [9], [10]. This solution may be used to establish a small-signal model suitable for AC analysis for a given circuit or can be used as initial condition which is congruent with the determination of its transient response through the integration of its associated differential-algebraic equations [11]. The most used method for the determination of DC solutions is the Newton-Raphson (NR) iterative method [12]. Unfortunately, the determination of DC solutions for nonlinear circuits based on the aforementioned method can become an extremely time-consuming process for large-scale systems. There are some well-known methods to alleviate this problem such as the step control for the NR iterative process, the use of sparse-matrix techniques for the solution of linear equations formulated through the course of the iterative process. Moreover, it is also possible to reduce the complexity of the matrix formulation process by bypassing the calculation of mathematical models of nonlinear elements when their branch voltages or currents have not changed significantly in previous iterations. These methods may lead to a reduction in the time required to find a given DC solution either by a reduction in the number of mathematical operations carried out or by a decrease in the number of iterations required. At this level, it is even possible to consider the parallel solutions of the linearized equations established during the solution process [13]–[15].

Another strategy is the partitioning of the original system of linearized equations used in the determination of a DC solution into a series of equation subsets which may, in principle, be taken as “decoupled” of each other. If this strategy is considered, the linearized equations of the circuit under analysis take a Bordered Block Diagonal (BBD) structure. In topological terms, such an approach is equivalent to decompose a circuit into simpler subcircuits which show a strong local connectivity for their internal nodes but are loosely connected to each other. In principle, each of these subcircuits could be treated as a separate problem and concurrently solved. In this strategy the decomposition procedure applied to the linearized equations of the circuit, as it takes place in our paper, or can be applied to its nonlinear equations to yield the Multi-level Newton-Raphson method as it has been considered in a number of works [16]–[19].

In this paper, a circuit simulator which combines the aforementioned strategies is presented. Moreover, an improvement

This work has been partially supported by Mexican CONACyT under grant contract 84819-Y. Moreover, D.E. Corts Udave received a scholarship from Mexican CONACyT under contract number 364221/243934 for the realization of this work.

Jan Ogrodzki is with Institute of Electronic Systems, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warszawa, Poland (e-mail: J.Ogrodzki@ise.pw.edu.pl).
D. E. C. Udave and M. A. Gutiérrez de Anda are with Instituto Nacional de Astrofísica, Óptica y Electrónica, Calle Luis Enrique Erro 1, Santa Maria Tonantzintla, C. P. 72840 San Andres Cholula, Puebla, Mexico (e-mails: lordecu@gmail.com; mdeanda@inaoep.mx).

of the formulation of the BBD decomposition required for the concurrent solution of subcircuits is also presented. The subsystems of equations formulated through the previous decomposition scheme are solved concurrently by means of parallel threads in a multi-core processor platform. Depending on the complexity of the circuit under analysis, the proposed approach is able to shorten its simulation time.

The algorithm presented in this paper is implemented in C++. It has been evaluated and compared against similar analysis approaches. This algorithm serves as the core of a rather simple circuit simulator which is devoted to the determination of the operating point of a circuit containing nonlinear elements. The circuit simulator formulated for this aim is endowed with a circuit description language which allows some flexibility in the description of circuits with two-terminal nonlinear elements.

II. The Parallelizable BBD-Based Algorithm

In order to parallelize the DC analysis of a large-scale nonlinear circuit (LSNC), it must be first partitioned into a collection of nonlinear subcircuits. The NR method applied to the resulting circuit configuration transforms the original problem into a simpler problem, namely, formulation of a sequence of solutions of large scale linear circuits (LSLC), each one decomposed into linear blocks. The decomposition of the LSLC into linear blocks is equivalent to the reordering of its linear equations and its unknown variables in such a way that the BBD structure of the matrix is attained [20]–[22]. In the reordering process we utilize a concept of internal variables and block variables. Since the internal variables (in groups corresponding to subsequent blocks) are first enumerated, they are also first eliminated during Gaussian Elimination (GE) of the solution process. The GE is stopped after the elimination of all the internal variables. Such a partial GE gives equations stretched over block variables only. These equations can be interpreted as the equations of a LSLC built of blocks represented by their Thévenin-Norton equivalents, i.e. the equations with eliminated internal variables.

Let the LSLC be composed of n_s blocks as it was introduced in [23]. The i-th block ($i = 1, \ldots, n_s$) contains n_i internal variables $\mathbf{x}_i$ and n_{0i} block variables $\mathbf{v}_{0i}$ which connect the block with other blocks. In Fig. 1 an example is shown where the circuit is composed of $n_s = 3$ blocks. For instance, subckt_1 has 3 block variables $\{v_1, v_2, \text{ground}\}$, a number of internal variables $\mathbf{x}_1$ and a vector $\mathbf{b}_1$ of internal excitations. The block variables belonging to all the circuit blocks as well as the nodal voltages of those elements which do not belong to a particular circuit block (e.g., the voltage source V and resistor R) form a set of block variables $\mathbf{v}_0$ of the LSLC. The LSLC, e.g. the circuit shown in Fig. 1, can be described by means of equations which have a matrix of the BBD form, namely:

$$\begin{bmatrix} \mathbf{Y}_1 & & & \mathbf{P}_1 \\ & \mathbf{Y}_2 & & \mathbf{P}_2 \\ & & \mathbf{Y}_3 & \mathbf{P}_3 \\ \mathbf{Q}_1 & \mathbf{Q}_2 & \mathbf{Q}_3 & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{v}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{b}_0 \end{bmatrix} \tag{1}$$

In equation (1), $\mathbf{Y}_i$ ($i = 1, 2, 3$) denote matrices of the blocks, $\mathbf{P}_i$ are responsible for connections of the blocks with the block terminals which have voltages $\mathbf{v}_0$, $\mathbf{Q}_i$ and $\mathbf{R}$ describe a current balance at the block nodes and $\mathbf{R}$ denotes a matrix created by elements connected to the block nodes. Among these elements we have elements belonging to the blocks as well as elements external to them (e.g., the resistor R in Fig. 1). As for the RHS of (1) it is composed of the internal excitations $\mathbf{b}_i$ of the blocks and of the excitations $\mathbf{b}_0$ connected to the block nodes. These excitations may belong to the circuit blocks or may be external to them (e.g., the voltage source V in Fig. 1).

If we separate the i-th block then its equations take the form:

$$\begin{bmatrix} \mathbf{Y}_i & \mathbf{P}_i \\ \mathbf{Q}_i & \mathbf{R}_i \end{bmatrix} \begin{bmatrix} \mathbf{x}_i \\ \mathbf{v}_{0i} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_i \\ \mathbf{b}_{0i} \end{bmatrix}. \tag{2}$$

where $\mathbf{R}_i$, $\mathbf{b}_{0i}$ are a matrix and an excitation vector created by those elements of the block which are connected to the block nodes. After elimination of the internal variables $\mathbf{x}_i$ from (2) the following relation is obtained:

$$\hat{\mathbf{R}}_i \mathbf{v}_{0i} = \hat{\mathbf{b}}_i, \tag{3}$$

where the matrix $\hat{\mathbf{R}}_i = \mathbf{R}_i - \mathbf{Q}_i \mathbf{Y}_i^{-1} \mathbf{P}_i$ and the vector $\hat{\mathbf{b}}_i = \mathbf{b}_{0i} - \mathbf{Q}_i \mathbf{Y}_i^{-1} \mathbf{b}_i$ constitute a Thévenin-Norton equivalent of the block. A description of this equivalent can be rewritten as:

$$\hat{\mathbf{R}}_i = \mathbf{R}_i - \mathbf{Q}_i \mathbf{M}_i, \quad \hat{\mathbf{b}}_i = \mathbf{b}_{0i} - \mathbf{Q}_i \mathbf{n}_i, \tag{4}$$

where the matrix $[\mathbf{M}_i, \mathbf{n}_i]$ is a solution of the multi-RHS equation:

$$\mathbf{Y}_i [\mathbf{M}_i, \mathbf{n}_i] = [\mathbf{P}_i, \mathbf{b}_i]. \tag{5}$$

Since the main circuit contains n_s blocks, therefore we have the structure as in (1). After the reduction of the internal variables, we may take into account only reduced equations of the main circuit, namely:

$$\hat{\mathbf{R}} \mathbf{v}_0 = \hat{\mathbf{b}}, \tag{6}$$

where the matrix $\hat{\mathbf{R}}$ and the vector $\hat{\mathbf{b}}$ are a superposition of contributions of elements constituting the main circuit, i.e. the blocks and the elements not belonging to any circuit block such as the voltage source V and the resistor R in Fig. 1. Hence eq. (6) may be formulated by means of stamps. The stamps of the blocks are calculated from eq. (4), whereas the stamps of distinct elements for the setup of modified nodal equations (MNE) are well known [11].

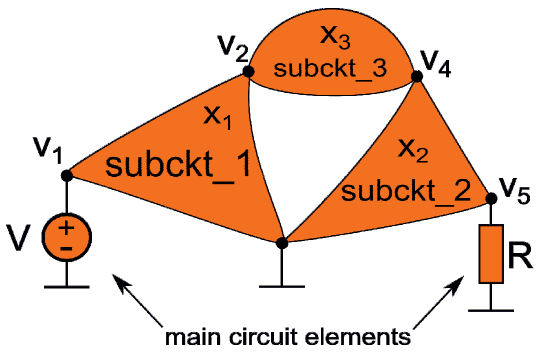

Fig. 1. Example of the main circuit built of 3 subcircuits and 2 distinct elements.

After the solution of the main circuit equations (6) with respect to $\mathbf{v}_0$ we select for each block a subset of relevant solutions $\mathbf{v}_{0i}$ and calculate internal variables from (7) resulted from the 2nd row of (2) after some algebra:

$$\mathbf{Y}_i\mathbf{x}_i = \mathbf{b}_i - \mathbf{P}_i\mathbf{v}_{0i}. \qquad (7)$$

The solution of equations (5) and (7) needs a single LU decomposition of $\mathbf{Y}_i$ followed by several forward-backward (FB) substitutions. A final NR Algorithm 1 is composed of the following 4 steps repeated in the NR loop until a convergence is reached. The solutions of (5), (6), (7) and the result of the matrix multiplications indicated in (4) are obtained with the help of the sparse matrix techniques. Moreover, a two-level bypassing process may be easily implemented: bypassing of nonlinear elements in step 1 of Algorithm 1 during the calculation of stamps for nonlinear elements and bypassing of blocks in step 4 during the calculation of their stamps indicated in (4).

Algorithm 1.

In subsequent NR iterations until convergence is reached:

1) For all blocks: formulation of submatrices with bypassing and solution of (5) (one LU factorization, several FB substitutions). This step is parallelizable.
2) For all blocks: matrix multiplications: $\hat{\mathbf{R}}_i = \mathbf{R}_i - \mathbf{Q}_i\mathbf{M}_i$, $\hat{\mathbf{b}}_i = \mathbf{b}_{0i} - \mathbf{Q}_i\mathbf{n}_i$. This step is parallelizable.
3) Formulation of the matrix and RHS of the main circuit using stamps of blocks; bypassing; solution of (6).
4) For all blocks: multiplications; solution of (7) (only FB substitutions). This step is parallelizable.

III. Analysis of Numerical Complexity

A. The Full Matrix Model

Let the LSNC circuit be composed of n_s blocks connected to n_0 block nodes and of N unknown variables of modified nodal equations. For simplicity and without any lack of generality let the blocks will be of the same size, i.e., each one has the same number $n_i = (N - n_0)/n_s$ of internal variables.

Each NR iteration of Algorithm 1 requires a sequence of the following operations:

- for n_s blocks: LU factorization of a $(n_i \times n_i)$-matrix, $n_0 + 1$ FB substitutions for RHSs of size n_i, multiplication of a $(n_0 \times n_i)$-matrix by a $(n_i \times n_0)$-matrix, two multiplications of an $(n_0 \times n_i)$-matrix by an $(n_i \times 1)$-vector, FB substitution for the RHS of size n_i;
- LU factorization and FB substitutions for the $(n_0 \times n_0)$-matrix of the main circuit.

If we take into account subtractions, multiplications and divisions then, for a $(n \times n)$-matrix, the numerical complexity for the operation of the solution process are as follows:

$$T_{LU}(n) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n,$$
$$T_{FB}(n) = n(2n-1)$$

and

$$T_{MUL}(p,q) = p^2r(2q-1)$$

(multiplication of a $(p \times q)$-matrix by a $(q \times p)$-matrix).

Let us compare three different versions of the algorithm:

- solution by means of one large system of equations without neither decomposition nor parallelization,
- solution via BBD decomposition without any parallelization,
- solution via BBD decomposition with parallelization.

In the first case (one set of equations), we have one LU factorization and one FB substitution, so the numerical complexity takes the form $T_{one} = T_{LU}(N) + T_{FB}(N)$. In the second case (circuit decomposition and serial solution), the numerical complexity is equal to

$$T_{ser} = n_sT_1 + T_2$$

where

$$T_1 = T_{LU}(n_i) + (n_0+2)T_{FB}(n_i) +$$
$$+ T_{MUL}(n_0, n_i, n_0) + 2T_{MUL}(n_0, n_i, 1),$$
$$T_2 = T_{LU}(n_0) + T_{FB}(n_0).$$

After some algebra this yields:

$$T_1 = \frac{2}{3}n_i^3 + \frac{7}{2}n_i^2 - \frac{13}{6}n_i + n_0^2(2n_i - 1) + n_0(2n_i^2 + 3n_i - 2),$$
$$T_2 = \frac{2}{3}n_0^3 + \frac{3}{2}n_0^2 - \frac{7}{6}n_0,$$

In the third case (circuit decomposition and parallel solution), let us assume infinite number of parallel processors with no latency issues for the creation of separate threads. Then the complexity can be written as

$$T_{paral} = T_1 + T_2 = \frac{2}{3}n_i^3 + \frac{7}{2}n_i^2 - \frac{13}{6}n_i + \frac{2}{3}n_0^3 +$$
$$+ n_0^2(2n_i + \frac{1}{2}) + n_0(2n_i^2 + 3n_i - \frac{19}{6}).$$

To describe an ideal, theoretical profit of the parallel implementation we introduce two figures of merit:

- the figure of merit $\eta = T_{ser}/T_{paral}$ normalized to a complexity of the serial implementation,

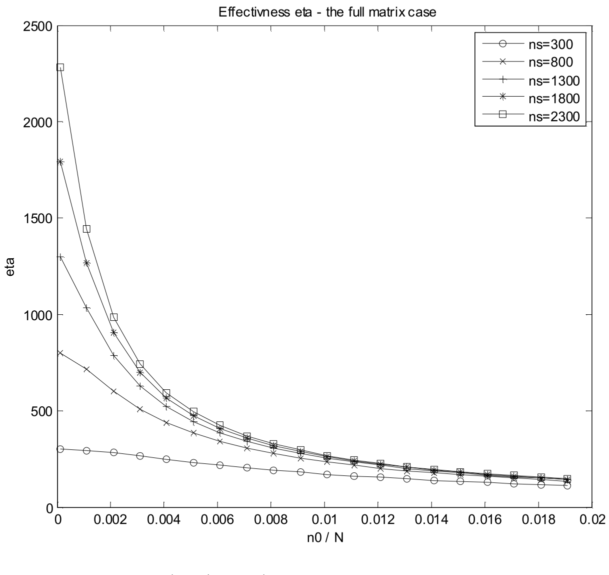

Fig. 2. Theoretical $\eta(n_0/N, n_s)$ of the full matrix algorithm.

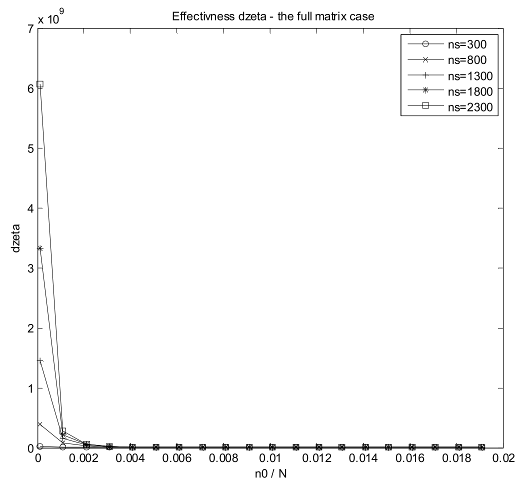

Fig. 3. Theoretical $\varsigma(n_0/N, n_s)$ of the full matrix algorithm.

- the figure of merit $\varsigma = T_{one}/T_{paral}$ normalized to a complexity of the one big matrix implementation.

Both figures of merit can be plotted against three parameters: $N = n_s n_i + n_0$, n_s and n_0. Let N be fixed, e.g. equal to 100000. The introduced figures of merit can be analyzed in the 2D space, namely $\eta(n_s, n_0)$ as it is shown in Fig. 2 and $\varsigma(n_s, n_0)$ as it is plotted in Fig. 3. We observe that both ideal figures of merit increase with n_s and decrease with n_0. If $n_0 \to 0$ then $\eta \to n_s$ and $\varsigma \to n_s^3$. If $n_0 \to N$ then $\eta \to 1$ and $\varsigma \to 1$. Both figures of merit are always greater than 1. This shows that in all cases of the full matrix implementation the parallel algorithm is better than a respective serial one, especially strongly when $n_0 \ll N$. Moreover we observe from η and ς that, for a n_s small in comparison with N, the efficiency of the algorithm is weakly dependent on n_0. If $n_0 \ll N$, the figures of merit approach n_s and n_s^3 respectively.

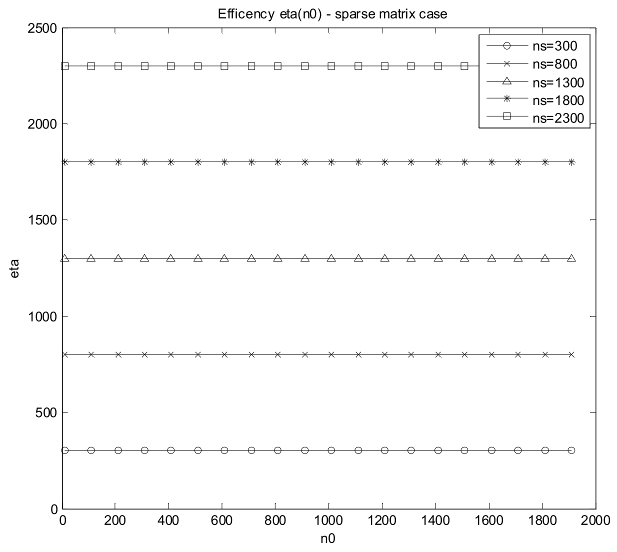

Fig. 4. Theoretical $\eta(n_0, n_s)$ of the sparse matrix algorithm.

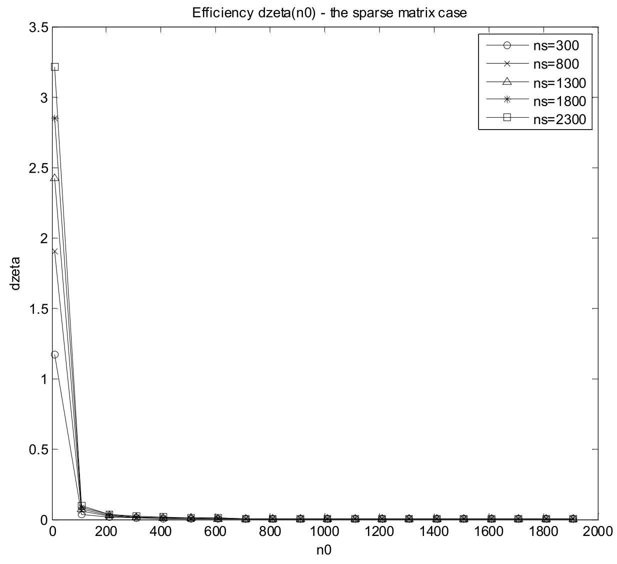

Fig. 5. Theoretical $\varsigma(n_0, n_s)$ of the sparse matrix algorithm.

B. Complexity Model for the Sparse Matrix Techniques

In this section we continue introductory results from [24]. The implementation of the algorithm proposed in this paper makes use of sparse matrix techniques for performing the matrix manipulations. The open-source KLU library has been used for this aim [25]–[27]. Since arithmetic operations could not be counted, an estimated number of operations involved for the matrix manipulations was obtained by means of dedicated experiments and computation time measurements. A surprising result of these experiments was that the library is so efficiently programmed that the LU factorization operates in the time $\Theta(n)$ while the FB substitution is carried out in the time $\Theta(1)$ (independent of the matrix size n to the measurement accuracy). Hence we introduce the complexity models: $T_{LU}(n) = \beta_1 n + \alpha_1$, $T_{FB}(n) = \alpha_2$, $T_{MUL}(n_i, n_0) = \alpha_3 n_i^{\beta_{31}} n_0^{\beta_{32}}$ (in the latter a $(n_0 \times n_i)$-matrix is multiplied by a $(n_i \times n_0)$-matrix). The introduced parameters are with a very good accuracy linear with respect to the matrix sparsity coefficient γ, i.e.:

$$\alpha_1 = 1.1831\gamma + 0.09593\mu s, \quad \beta_1 = 0.007695\gamma + 0.03337\mu s,$$

$$\alpha_2 = 0.0385\mu s, \quad \alpha_3 = 6.275\gamma + 4.3936\mu s,$$

$$\beta_{31} = 1.925\gamma + 0.3225, \quad \beta_{32} = 0.4\gamma + 1.4533.$$

on a computer with the I7 microprocessor. After some algebra, the figures of merit $\eta(n_s, n_0, \gamma)$ and $\varsigma(n_s, n_0, \gamma)$ introduced in Subsection III.A have been calculated for the case of sparse matrices. In Fig. 4 and Fig. 5 respectively these figures of merit have been plotted against n_0 for several values of n_s and with a sparsity coefficient $\gamma = 0.09$, while in Fig. 6 and Fig. 7 respectively they have been plotted against γ. The complexity models and evaluations of the figures of merit involved are much more realistic compared to ones introduced in Subsection III.A for the full matrices case. This complexity analysis realistically shows features of the proposed algorithm. Its efficiency η (advantage of the parallel algorithm over the serial one) in Fig. 4 and Fig. 6 is weakly dependent on n_0 and γ, and is very well approximated by n_s.

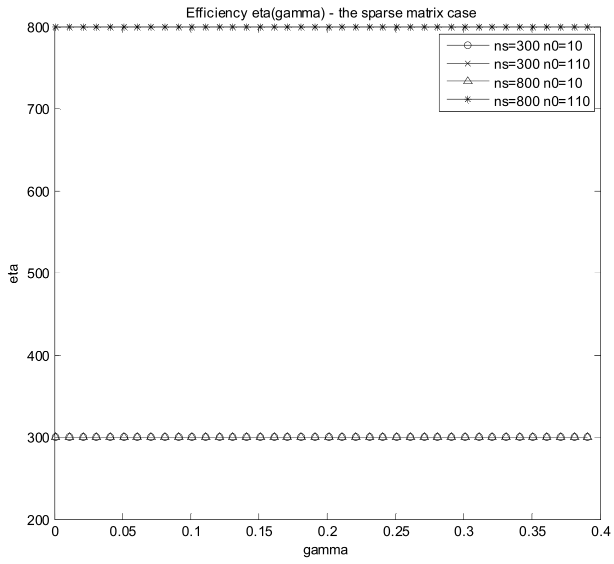

Fig. 6. Theoretical $\eta(\gamma, n_0, n_s)$ of the sparse matrix algorithm.

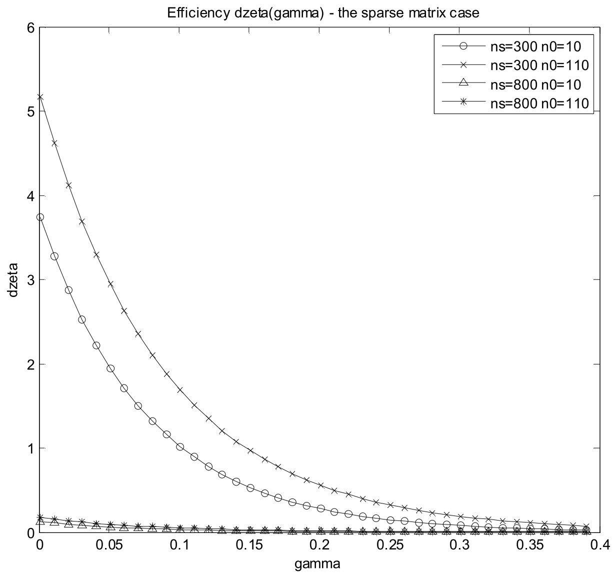

Fig. 7. Theoretical $\varsigma(\gamma, n_0, n_s)$ of the sparse matrix algorithm.

Also according to the figure of merit $\varsigma(n_s, n_0, \gamma)$ (comparison of the parallel algorithm with a decomposition-free one in Fig. 5 and Fig. 7) we can say that it is rather weakly dependent on n_0, and decreases with the sparsity coefficient, as well as with a number of subcircuits. If $\gamma < 0.1$ (realistic in large-scale circuits) and $n_s < 500$, then $\varsigma(n_s, n_0, \gamma)$ though greater than one, simultaneously it can reach, at most, 6. This level of efficiency with respect to the decomposition-free approach is all what we can expect from the algorithm.

IV. Implementation

A. Input Language and Parsing

The circuit description language is similar to the language used by SPICE [28] with specific modifications to enable the inclusion of user-defined nonlinear elements. An instance of a nonlinear resistor is defined as follows:

R<name> <positive_node> <negative_node> u(i)
'<expression_i>' <current_guess>

where the string <name> serves as a unique identifier for that instance in the input netlist, whereas strings <positive_node> and <negative_node> stand for identifiers of the nodes: a positive one and a negative one. The ordering of the nodes for a nonlinear resistor is essential since it establishes the direction of a positive current flowing through the element under consideration. Finally, <expression_i> (which is always enclosed between single quotes) is an algebraic expression which defines the branch voltage as a function of the branch current and with some numerical parameters or user-defined functions involved. Common mathematical functions such as the trigonometric functions are also available for their use in the formulation of the branch relation of a nonlinear resistor as well as of other nonlinear two-terminal elements. The nonlinear resistor thus defined is in fact a voltage defined, current-controlled element. Evidently, its current constitutes an unknown in the set of MNE which must be formulated to determine the bias point of a given circuit. It must be also noted that there is no restriction in the formulation of the algebraic expression used in the definition of the branch characteristic for a nonlinear resistor. Therefore, in order to facilitate convergence of the NR iterations when handling nonlinear resistors, the user may establish an initial guess of the branch current flowing through the nonlinear resistor by adding the <current_guess> token after the expression. If <current_guess> is not given, then the initial guess of its branch current will be equal to zero.

In a similar way, an instance of the nonlinear conductance is defined as

R<name><positive_node><negative_node> i(u)
'<expression_u>' <voltage_guess>

In this statement <expression_u> is an algebraic expression which describes the branch current as a function of the branch voltage. In this case, a definition of an initial guess of the branch voltage is possible as well. Unlike the nonlinear resistor, this element is fully compatible with the formulation of nodal equations for current defined, voltage-controlled elements. Note that the simulator can handle only explicit formulae either for the branch voltage of a resistor in terms of its branch current or for the branch current of a nonlinear conductance in terms of its branch voltage. Consequently, implicit mathematical relations for any of the aforementioned nonlinear elements are not allowed.

An instance of a nonlinear voltage-controlled voltage source (VCVS) takes the form

E<name><positive_node><negative_node>
<controlling_positive_node><controlling_negative_node>
'<expression>' <voltage_guess>

In this statement, the string <name> is used to identify the instance in the input netlist. Moreover, the strings

<positive_node> and <negative_node> indicate the nodes where the VCVS is connected. The controlling variable for this VCVS is given by a difference of nodal voltages of two nodes. These controlling nodes are respectively identified with the strings <controlling_positive_node> and <controlling_negative_node>. Finally, the string <expression> contains a valid algebraic expression involving the controlling voltage as well as user-defined parameters and functions. An initial guess definition for the controlling voltage is also available through the string <voltage_guess>. If this information is omitted by the user, the initial guess for the controlling voltage is assumed to be equal to zero.

For an instance of a current-controlled voltage source (CCVS), its definition has the following form:

```
H<name><positive_node><negative_node><element>
                          '<expression>' <current_guess>
```

In the previous statement, the string <element> contains the name of an element whose current controls the CCVS. This element cannot be considered in the formulation of nodal equations (NE) but it is acceptable by the modified nodal equations (MNE) where current of this branch is included as an unknown in the set of equations. Optionally, an instance of a current-controlled voltage source also may have indicated by <current_guess> an initial guess for the controlling current.

Similar definitions of instances to the controlled sources presented so far are available for voltage-controlled current sources (VCCS) and the current-controlled current sources (CCCS). Independent voltage and current sources are defined in the same way as in SPICE. The circuit language supports the definition of subcircuits. The definition of a subcircuit in the input netlist is made in the same way as in SPICE. The inclusion of subcircuits in the input netlist is very important since the simulator is not able to automatically identify circuit blocks with a high local connectivity. This task is left to the user and for this reason the inclusion of subcircuits in the input language is a required feature.

The parsing process of a given input netlist is driven by a finite state machine. The finite state machine was implemented with the aid of bison [29]. For this aim, a grammar of the input language for the simulator has been formulated. Particular care was taken in the formulation of the grammar in order to avoid reduce/reduce or shift/reduce conflicts to enable an automatic generation of the finite state machine with the aforementioned software tool. The parser required a routine for the extraction of tokens from the input netlist. For this aim, flex was used to build a lexical scanner [29]. The netlist was stored in a data structure which consists of a series of linked lists hierarchically organized for storing each of the elements present on it according to its type. In order to facilitate the creation process of the code required by each of these linked lists, the Tm tool was used [30]. The version of Tm used for the management of the code for the parser data structures was 2.2.1, whereas the versions of bison and flex used for the automatic generation of the parser code were, respectively, 2.4.1 and 2.5.35.

B. Architecture of the Main Solver

The implementation of the algorithm described in this section can be outlined in the following 6 steps:

- Parsing of the net-list (this step was already discussed in Subsection IV.A).
- Setup of data structures for each of the subcircuit instances.
- Building matrices of subcircuits.
- Extraction of matrix entries where updating is required during NR iterations.
- Iterative solution process including bypassing of elements and subcircuits.
- Generation of a text file with final results of the DC analysis.

The last five steps of this process will be now described with more detail.

The setup of data structures for each of the subcircuit instances contained in a given input netlist is an important step of the data processing required to determine the bias point of a nonlinear circuit using parallelization techniques as described in this paper. First of all, a subcircuit definition may be used several times to define a number of subcircuit instances in a given netlist. In fact, the subcircuit instances contained in the main circuit of an input netlist may be used to establish in a natural way a partition for a given circuit into a number of circuit blocks which are loosely connected to each other. For each subcircuit instance contained in the main circuit, the data structure containing its definition in terms of other elements must be copied. After this, all the circuit elements contained in that data structure must be renamed. This step is required to establish a difference between the constitutive elements of two different instances of the same subcircuit class. A similar process must be carried out for the internal nodes of each subcircuit instance. During this process, the relation between the external nodes of the subcircuit and the nodes of the main circuit must not be lost.

Once this process has been completed, submatrices representing each of the subcircuits instances must be built. Given that the linearized equations of the complete circuit must take a BBD form as indicated in equation (1), a set of matrices ($\mathbf{Y}_i$, $\mathbf{Q}_i$, $\mathbf{P}_i$, $\mathbf{R}_i$) must be generated for each subcircuit instance. These matrices are required to analyze a given subcircuit as indicated by (2) for the unknowns ($\mathbf{x}_i$ and $\mathbf{v}_{0i}$). According to the same expression, the excitation vectors ($\mathbf{b}_i$ and $\mathbf{b}_{0i}$) must be generated as well. In order to create the aforementioned matrices and vectors, all the variables belonging to the subcircuit must be extracted. For this aim, a vector which will contain, respectively, the names of all the nodes as well as the names of all the currents of all voltage-defined, current-controlled elements must be created. The vector associated to the node tags must be arranged in such a way that the internal nodes appear first. After these variables, the block nodes (i.e., nodes connected to the main circuit) should appear. Finally, the names of the currents of the elements which are not compatible with the nodal analysis must appear. After this, a reordering process for the internal variables must be performed. More specifically, the current belonging to a voltage source and one of its nodal voltages

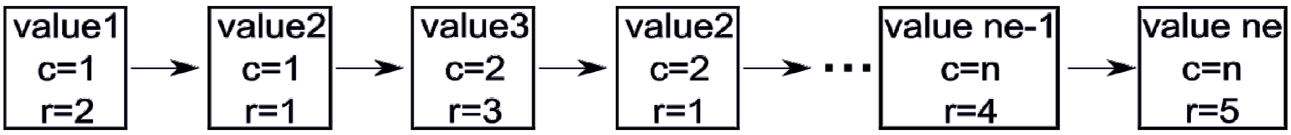

Fig. 8. Linked list representation of a sparse matrix. In this representation, for instance, $value3$ is located in its second column ($c = 2$) and in third row ($r = 3$).

must be exchanged. This reordering strategy, which is used in many simulators [11], [28], eliminates a diagonal structural zero and speeds up the LU factorization. At the very end, the vector containing the tags for all the nodes and the currents of all the voltage-defined, current-controlled elements of a given subcircuit will be used to establish indexes for the columns and rows of the matrices $\mathbf{Y}_i$, $\mathbf{Q}_i$, $\mathbf{P}_i$ and $\mathbf{R}_i$ which are required to describe it. The aforementioned matrices are stored as sparse matrices in the column-compressed form, i.e. they have elements stored in a linked list in column order while the rows in each column appear in an arbitrary order as it is shown in Fig. 8. The formulation process for these matrices adds the corresponding stamps for each of the elements present in the definition of a given subcircuit taking into account the indexes which were previously obtained for all the unknowns.

The subcircuit matrices stored in the linked list are translated into an alternative representation for its further processing with the routines provided by the KLU library. The KLU package [25]–[27] requires that a given sparse matrix is described in a compact way through the use of four different variables. The first one is an integer variable denoted as n. This variable contains the number of columns of a given sparse matrix. The second one is a real vector denoted as Ax which contains all the values of the structurally nonzero elements of the sparse matrix. This vector is organized such that the structurally nonzero elements are stored according to their order of appearance in the sparse matrix columns. In order to indicate the start of each column, an integer vector known as Ap is used. This vector is of size $n + 1$ and its first n entries represent indexes which indicate where each column begins. These indexes point to specific locations in the array Ax. The last entry of vector Ap is equal to the number of structurally nonzero elements contained in the sparse matrix. This information is required to track the size of vector Ax in a simple way. Finally, an integer vector known as Ai contains the row indexes of all the structurally nonzero elements of the sparse matrix. Evidently, the length of Ai is equal to the number of the structurally nonzero elements of the sparse matrix. According to the representation scheme previously introduced, matrix:

$$\begin{bmatrix} 1 & 0 & 5 \\ & 4 & \\ & 3 & 6 \end{bmatrix}$$

could be described as follows: $n = 3$, $Ax = [1, 0, 3, 4, 5, 6]$, $Ap = [0, 1, 4, 6]$, $Ai = [0, 0, 2, 1, 0, 2]$. Note that $Ap(4) = 6$ denotes the number of structurally nonzero elements. In this example, it can be seen that $Ax(2) = 0$. Although this element does not need to be stored in a sparse matrix, it may be still treated as a structurally nonzero entry. Therefore, it can also be stored in this representation scheme.

When the KLU representation is ready, stamps of the subcircuits ($\hat{\mathbf{R}}_i, \hat{\mathbf{b}}_i$ according to eq. (4)) may be built. This stamp is described by a structure which contains three KLU matrices: two for $\hat{\mathbf{R}}_i, \hat{\mathbf{b}}_i$ and one more in order to return voltage solutions at the connection nodes of the subcircuit. Its calculation requires a solution of the eq. (5) and the sparse multiplications indicated in equation (4). After the construction of all stamps of the subcircuits, we proceed to the main matrix building process in the same manner as it was for matrices of the subcircuits: first we add all elements belonging to the main circuit and then all the stamps (4) of the subcircuits.

At the next step we extract positions where information on the nonlinear elements appears and so the stamps will be added to the submatrices and the main matrix. This considerably speeds up the execution process. The prepared information is stored in linked lists which contain pointers to entries in the matrices where values are iteratively updated and convergence tests are performed. These lists are of different structure for different types of elements and have the length equal to a number of elements of a given type. This extraction process is performed on two levels: extraction of positions of nonlinear elements in the subcircuits and extraction of positions of stamps to the main matrix of distinct nonlinear elements and subcircuits.

In the next step, an iterative solution is performed by means of the NR algorithm with the BBD decomposition. In this process, a mechanism for bypassing the formulation of linearized equivalents for selected nonlinear elements has been included as well. As a first stage in this iterative process, the main circuit should be solved first. This solution should be passed to the subcircuits in order to solve them for their internal variables. After completing this task, convergence tests for all nonlinear elements of all subcircuits must be carried out. A convergence test for a given nonlinear element is performed according to the following expression

$$|x_1 - x_0| \leq \varepsilon \max(|x_1|, |x_0|) + \delta \tag{8}$$

where δ and ε are absolute and relative accuracies while x_1, x_0 are the values of the controlling and controlled variables of the nonlinear element in consideration (voltages and/or currents) during the current and preceding NR iteration respectively. If a nonlinear element meets condition (8), its next iterative calculation is bypassed and the element is treated as a linearized one. Moreover, if all the nonlinear elements belonging to one subcircuit satisfy the convergence condition, then the iterative calculation of the stamp for this subcircuit is bypassed and the subcircuit is treated as a linear one. This two-level bypassing process considerably speeds up calculations. If an element does not satisfy the stopping condition, corresponding entries in the matrix Ax are updated by recalculation of the

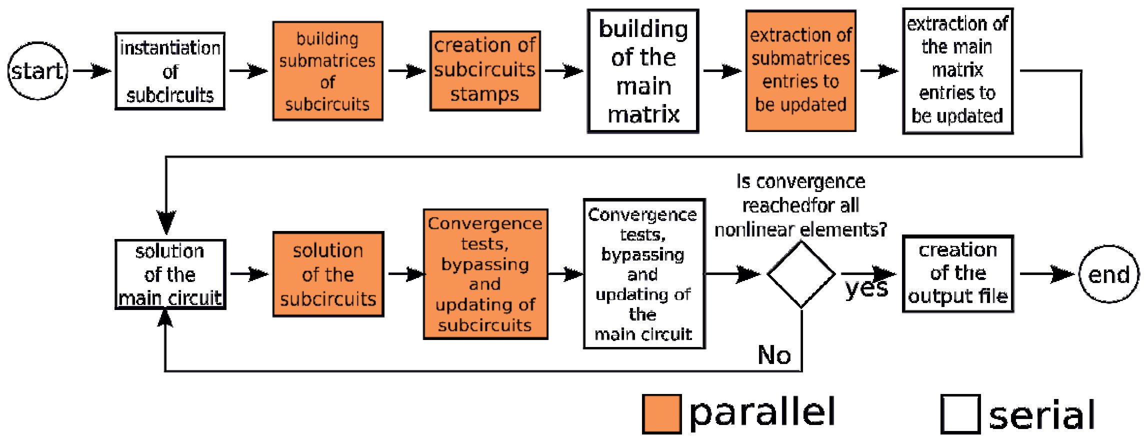
start
instantiation of subcircuits
building submatrices of subcircuits
creation of subcircuits stamps
building of the main matrix
extraction of submatrices entries to be updated
extraction of the main matrix entries to be updated
solution of the main circuit
solution of the subcircuits
Convergence tests, bypassing and updating of subcircuits
Convergence tests, bypassing and updating of the main circuit
Is convergence reachedfor all nonlinear elements?
yes
No
creation of the output file
end
parallel
serial

Fig. 9. General structure of the software.

stamp and NR iterations are continued until convergence is reached.

During the final stage, the DC variables for all the elements (voltages, currents, powers) are calculated and printed to an output file. These results are arranged according to the order established by the subcircuits.

An overall structure of the program is summarized in Fig. 9. In this schematic, those tasks which can be executed either in a serial or in a parallel fashion are clearly marked. The code belonging to the tasks which can be concurrently executed was parallelized using the Intel Parallel Studio design environment [31]. The use of this software tool made possible in a relatively short time the implementation of the proposed algorithm on multi-core, multithread processors.

It must be noted that the setup of the data structures required to handle each of the subcircuit instances is executed in a serial way. The setup of the aforementioned data structures for each of the subcircuits may involve multiple accesses to the same memory block. This kind of operations should be discouraged in a parallel environment. Once the auxiliary data structures are ready in sparse form, we can perform in parallel a building process for all the structures of the subcircuits and execute, also in parallel, the process of extraction of the updated entries in the subcircuit matrices. As for the main circuit, its matrix manipulation must be done in a serial way as well as the extraction of its updated entries to the matrix. Afterwards, the solving process may be performed in a mixed manner: the solution of the main circuit is done in series, while the solutions of all internal variables of the subcircuits are performed in parallel. Regarding the convergence tests for the nonlinear elements and the update process of all the matrices involved, they must be done in a mixed fashion. Calculation of the convergence criteria for nonlinear elements of the main circuit (distinct elements and those belonging to the subcircuits) have to be implemented in series, while the convergence criteria for elements of the subcircuits may be calculated in parallel. At the very end the output file is created in a serial process.

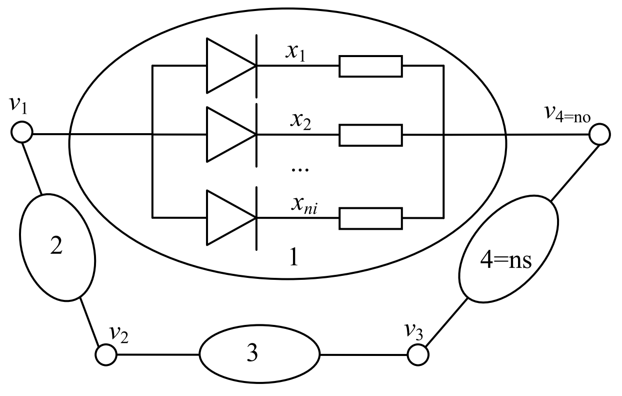

Fig. 10. Principle of construction of benchmark circuits for the performance assessment of the parallel simulator.

V. Verification and Conclusions

A. Circuit Benchmarks Used for Verification

In this section, some results obtained from the performance testing of the BBD algorithm on parallel processors will be presented. For this aim adequate benchmarks have been generated which have a fixed total number of variables and a regular structure composed of the same subcircuits. Each subcircuit is a one-port (two-terminal) network built of n_i parallel branches, each one composed of a diode and a conductance in series. We place n_s subcircuits like this among n_0 block nodes as it is shown in Fig. 10. All the benchmarks differ in n_i, n_s, n_0 but always hold the relation $n_s n_i + n_0 = N$.

For each subcircuit, the matrix given on the LHS of (2) is of the following sparse structure:

$$\begin{bmatrix} \mathbf{Y}_i & \mathbf{P}_i \\ \mathbf{Q}_i & \mathbf{R}_i \end{bmatrix} = \begin{bmatrix} x & & & & x & x \\ & x & & & x & x \\ & & \dots & & \dots & \dots \\ & & & x & x & x \\ x & x & \dots & x & x & \\ x & x & \dots & x & & x \end{bmatrix} \tag{9}$$

where structural nonzero entries are denoted by x. The last two rows and columns of the matrix stand for the terminal nodes while the first n_i rows and columns correspond to n_i internal nodes x_i. The mean sparsity coefficient of this matrix is

$$\gamma = n_s(5N - 5n_0 + 2n_s)/(N + 2n_s - n_0)^2. \tag{10}$$

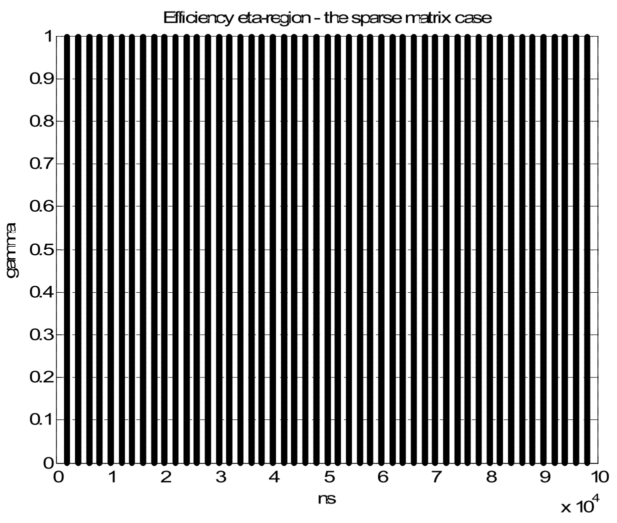

Fig. 11. The η-region for fixed $n_0 = 110$.

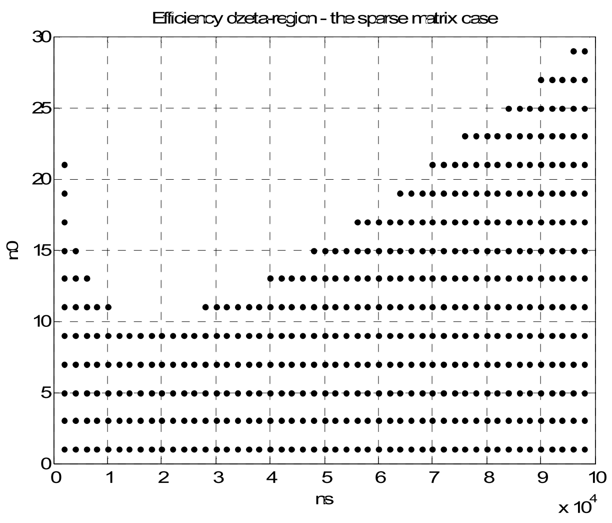

Fig. 13. The ς-region for the introduced benchmark circuits.

If $N = 100000$, $n_0 = 1000$, $n_s = 2000$, then for instance γ is equal to 0.094.

B. Efficiency Regions

Efficiency regions, η-region and ς-region, provide a convenient description of the algorithm performance, namely:

$$\eta - \text{region} = \{(n_s, n_0, \gamma) : \eta(n_s, n_0, \gamma) > 1\}, \quad (11)$$

$$\varsigma - \text{region} = \{(n_s, n_0, \gamma) : \varsigma(n_s, n_0, \gamma) > 1\}. \quad (12)$$

where $\eta(n_s, n_0, \gamma)$ and $\varsigma(n_s, n_0, \gamma)$ have been introduced in Subsection II.A. 2D cross-sections of these regions for a given n_0 and complexity models from Subsection III.B can be easily generated. The η-region is rather extensive, as we see in Fig. 11 for $n_0 = 110$, and $N = 100000$. It demonstrates that the BBD-parallel algorithm is more efficient than the serial one for all reasonable combinations of the circuit parameters.

As for the ς-region, its properties are quite different. For the same N and n_0, as in the η-region case, we obtain a plot given in Fig. 12. From this plot we see that the BBD-parallel algorithm is more efficient than the decomposition-free one if $\gamma < 3e - 7n_s$ (matrices are of strong sparsity).

Now we come back to the aforementioned benchmarks, whose sparsity coefficients satisfy the relation (10) and we explore their ς-region in the (n_s, n_0)-plane. From this region, plotted in Fig. 13, it can be found out that the BBD-parallel algorithm may be more efficient than the decomposition-free one if $n_0 < 30$. This condition may be satisfied for $n_s < 10000$ or for $n_s > 30000$ and $n_0 < 3e - 4n_s$. The algorithm will be efficient for instance if $n_0 = 13$ and $n_s = 5000$ or $n_s = 50000$ and $n_0 < 15$.

C. How to Test Practical Efficiency?

Practical verification of the algorithm will require two or better three versions of the simulator code.

In the case of two versions generated by means of the Intel Parallel Studio [31] in the Linux environment: a serial one with the BBD-decomposition and a parallel one with

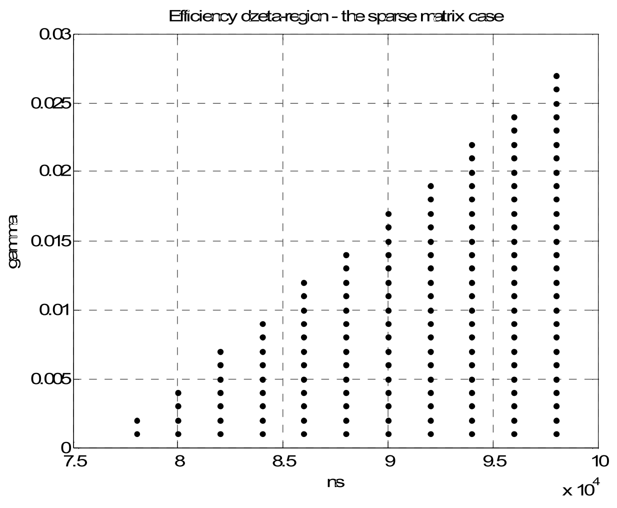

Fig. 12. The ς-region for fixed $n_0 = 110$.

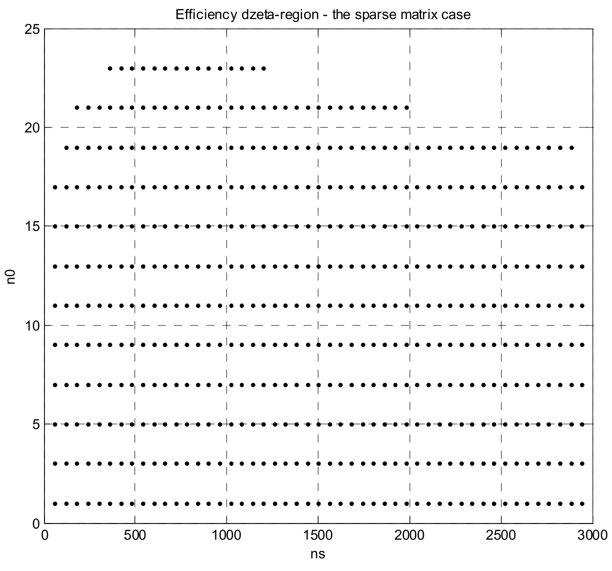

Fig. 14. The zoomed ς-region for the introduced benchmark circuits.

the BBD-decomposition, the latter according to the schematic in Fig. 9. Both versions are executed on the same available processor, simulation times, t_{serial} and $t_{parallel}$, are measured for several combinations of n_s and n_0, and a practical figure of merit $\eta_p = t_{serial}/t_{parallel}$ is calculated. As for the figure of merit $\varsigma_p = t_{one}/t_{parallel}$, we have a problem with the lack of the decomposition-free code and its simulation time t_{one}. That is why only an approximation ς_{app} of the ς_p may be obtained in virtue of an assumption that execution times of the serial and the decomposition-free codes are both proportional to the theoretical figures of merit T_{serial}, and T_{one}, known from Subsections III.A and III.B, respectively. Hence, we obtain the approximation $\varsigma_{app} = \eta_p T_{one}/T_{serial}$. This enables a rough verification of the BBD-parallel code compared with the decomposition-free code.

After implementation of the decomposition-free algorithm by means of the KLU package used in the BBD-parallel simulator, the three codes will be available and a better evaluation of the BBD-parallel algorithm will be possible in a straightforward way. So far we have only a semi-empirical comparison based on measurements of the sparse matrix operations in Section III.B.

D. Conclusions from the Semi-empirical Verification

As we see from analysis of the semi-empirical figure of merit η, the parallel computation of the subcircuits provides an advantage theoretically proportional to a number of subcircuits n_s. In practice this optimistic result is limited by an efficiency of a linear equations solver. Hence, we have to use the figure of merit ς instead of η. The higher efficiency of the linear solver, relatively the lower is efficiency ς of the BBD-parallel algorithm.

Theoretically, for the full matrix techniques this figure of merit might be even $O(n_s^3)$. For the sparse matrix techniques, the BBD-parallel algorithm is less advantageous, though still efficient. So far used sparse matrix packages, operating in time $O(n^{1.1-1.5})$, provided an efficient BBD-parallel analysis for a not to big number of the block-nodes. However, the KLU package implemented in this project is yet more efficient and operates in the linear time. This limits efficiency of the BBD-parallel algorithm to rather small main circuits ($n_0 < 30$) and rather low sparsity coefficients, though a number of subcircuits may vary in a wide range, as it is seen in Fig. 13. Fig. 14 demonstrates the same ς-region zoomed to $n_s < 3000$. We see that a biggest n_0 (about 25) occurs for $500 < n_s < 1000$. In this range the algorithm is the most useful.

The efficiency conditions found in this paper limit practical applications of the BBD-parallel algorithm to a rather small number of block nodes. This requires a smart method of large-scale circuit decomposition. In this paper we recommend its implementation as an alternative for a decomposition-free algorithm, automatically switched-on in the case when the efficiency conditions discussed in this paper and also decomposition conditions are satisfied.

References

[1] K. A. Gallivan, M.-C. Chang, I. N. Hajj, D. Smart, and T. N. Trick, "Parallel circuit simulation on supercomputers," *Proceedings of the IEEE*, vol. 77, no. 12, pp. 1915–1931, 1989.

[2] M. Günther, U. Feldmann, and J. ter Maten, "Modelling and discretization of circuit problems," in *Handbook of Numerical Analysis: Numerical Methods in Electromagnetics*, W. H. A. Schilders and E. J. W. ter Maten, Eds. Amsterdam, The Netherlands: Elsevier Science, 2005, pp. 523–659.

[3] X. Ye, W. Dong, P. Li, and S. Nassif, "Hierarchical multialgorithm parallel circuit simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 45–58, 2011.

[4] E. R. Keiter, H. K. Thornquist, R. J. Hoiekstra, T. V. Russo, R. L. Schiek, and E. L. Rankin, "Parallel transistor-level circuit simulation," in *Simulation and Verification of Electronic and Biological Systems*, P. Li, L. M. Silveira, and P. Feldmann, Eds. Heidelberg, Germany: Springer, 2011, pp. 1–21.

[5] H. K. Thornquist and E. R. Keiter, "Advances in parallel transistor-level circuit simulation," in *Scientific Computing in Electrical Engineering SCEE 2010*, B. Michielsen and J.-R. Poirier, Eds. Heidelberg, Germany: Springer, 2012, pp. 257–265.

[6] C. Baker, E. Boman, M. Heroux, E. Keiter, S. Rajamanickam, R. Schiek, and H. Thornquist, "Enabling next-generation parallel circuit simulation with Trilinos,," in *Euro-Par 2011: Parallel Processing Workshops*, M. Alexander, P. D'Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. D. Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. Scott, J. L. Traff, G. Vallé, and J. Weidendorfer, Eds. Heidelberg, Germany: Springer, 2012, pp. 315–323.

[7] H. Quian, Y. Deng, B. Wang, and S. Mu, "Towards accelerating irregular EDA applications with GPUs,," *Integration, the VLSI Journal*, vol. 45, no. 1, pp. 46–60, 2012.

[8] T.-H. Weng, R.-K. Perng, and K.-C. Li, "On parallelization of circuit simulation SPICE3 using multithreaded programming techniques," *Journal of the Chinese Institute of Engineers*, vol. 35, no. 2, pp. 259–267, 2012.

[9] W. Ho Chung, D. A. Zein, A. E. Ruehli, and P. A. Brennan, "An algorithm for DC solution in an experimental general purpose interactive circuit design program," *IEEE Transactions on Circuits and Systems*, vol. 24, no. 8, pp. 416–421, 1977.

[10] D. A. Zein, "Solution of a set of nonlinear algebraic equations for general purpose CAD programs," in *Circuit analysis simulation and design. General aspects of circuit analysis and design*, A. E. Ruehli, Ed. Amsterdam-New York-Oxford-Tokyo: North Holland, 1986.

[11] J. Ogrodzki, *Circuit simulation methods and algorithms*. Boca Raton-New York-Tokyo: CRC Press, 1995.

[12] J. M. Orthega and W. C. Rheinboldt, *Iterative solution of Nonlinear Equations in several variables*. New York: Academic Press, 1970.

[13] J. I. Aliaga, M. Bollhöffer, A. F. Martn, and E. S. Quintana-Ortí, "Exploiting thread-level parallelism in the iterative solution of sparse linear systems," *Parallel Computing*, vol. 37, no. 3, pp. 183–202, 2011.

[14] H. Huang, L. Wang, E. J. Lee, and P. Chen, "An MPI-CUDA implementation and optimization for parallel sparse equations and least squares (LSQR)," *Procedia Computer Science*, vol. 9, pp. 76–85, 2012.

[15] Y. Wang and H. Yang, "An adaptive LU factorization algorithm for parallel circuit simulation," in *Proceedings of 17Th Asia and South Pacific Design Automation Conference*, 2012, pp. 359–364.

[16] N. Rabbat, A. Sangiovanni-Vincentelli, and H. Hsieh, "A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain," *IEEE Transactions on Circuits and Systems*, vol. 26, no. 9, pp. 733–741, 1979.

[17] N. Frohlich, B. M. Riess, U. A. Wever, and Q. Zheng, "A new approach for parallel simulation of VLSI circuits on a transistor level," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 45, no. 6, pp. 601–613, 1998.

[18] M. Honkala, J. Roos, and M. Valtonen, "New multilevel Newton-Raphson method for parallel circuit simulation," *Proceedings of European Conference on Circuit Theory and Design*, vol. 1, pp. 113–116, 2001.

[19] J. G. Fijnvandraat, S. H. M. J. Houben, E. J. W. ter Maten, and J. M. F. Peters, "Time domain analog circuit simulation," *Journal of Computational and Applied Mathematics*, vol. 185, no. 2, pp. 441–459, 2006.

[20] F. F. Wu, "Solution of large scale networks by tearing," *IEEE Transactions on Circuits and Systems*, vol. 23, no. 12, pp. 706–713, 1976.

[21] M. Vlach, "LU decomposition and forward-backward substitution of recursive bordered block diagonal matrix," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1983, pp. 701–703.

[22] D. Bukat, G. Centkowski, and J. Ogrodzki, "OPTIMA-1.1 - A hierarchical decomposition based analyser including user defined models," in *Proceedings of the European Conference on Circuit Theory and Design*, 1991.

[23] D. E. C. Udave, J. Ogrodzki, and M. A. G. de Anda, “A study of the parallel algorithm for DC large-scale simulation of nonlinear systems,” in *Photonics Application in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2012*, R. S. Romaniuk and K. S. Kulpa, Eds., 2012, Proceedings of SPIE, vol. 7503 (SPIE, Bellingham, WA).

[24] ——, “DC simulator of large-scale nonlinear systems for parallel processor,” in *Photonics Application in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2012*, R. S. Romaniuk and K. S. Kulpa, Eds., 2012, Proceedings of SPIE, vol. 7503 (SPIE, Bellingham, WA).

[25] T. A. Davis and E. P. Natarajan, “Algorithm 907: KLU, a direct sparse solver for circuit simulation problems,” *ACM Transactions on Mathematical Software*, vol. 37, no. 3, 2010.

[26] T. A. Davis, “Direct methods for sparse linear systems,” in *SIAM Book Series on the Fundamentals of Algorithms*. Philadelphia: SIAM, 2006.

[27] K. Stanley, “KLU: a “Clark Kent” sparse LU factorization algorithm for circuit matrices,” in *SIAM Conference on Parallel Processing for Scientific Computing (PP04)*, 2004.

[28] SYNOPSYS ®, [HSPICE ®Reference Manual: Commands and Control Options], ver. A-2007.09, September (2007).

[29] J. Levine, *Flex & Bison*. California: O’Reilly Media, 2009.

[30] C. Van Reeuwijk, “Tm: a code generator for recursive data structures,” *Software - Practice and Experience*, vol. 22, no. 10, pp. 899–908, 1992.

[31] “Intel ®Parallel Studio XE 2011 for Linux* - Documentation,” 20 June 2012, http://software.intel.com/en-us/articles/intel-parallel-studio-xe-for-linux-documentation/#inspector.