2015-06-16

System formulation for parallel circuit analysis

Savalia, Tapankumar Kishorbhai

System Formulation for Parallel Circuit Analysis

by

Tapankumar Kishorbhai Savalia

A Thesis
Presented to Lakehead University
in Partial Fulfilment of the Requirement for the Degree of
Master of Science
in
Electrical and Computer Engineering

Thunder Bay, Ontario, Canada

May 2014

Abstract

Advances in communication systems and VLSI circuits increase the performance requirements and complexity of circuits. During the design process, there is a need to perform computationally demanding numerical simulations to verify the functionality of circuits under design. One way to reduce computing time is to use parallel processing. This thesis discusses different techniques for parallel circuit analysis with emphasis in the formulation of equations for a circuit decomposed in subcircuit blocks.

For manually decomposed circuit, this thesis introduces two approaches to formulate circuit equations. The two formulations allow to independently analyze each subcircuit block by periodically exchanging information with a master process. A node-tearing process is used to divide the system Jacobian in blocks. The first formulation is based on the nodal voltages and currents at the interface nodes. The second formulation is presented in this thesis for the first time and uses scattering waves to exchange information between subcircuits. The two formulations are described in detail and implemented in a general circuit simulator.

Simulation results comparing the performance of the proposed formulations for different circuits are presented. These results indicate that there is no advantage in using waves to exchange information between subcircuits. Moreover, at least with the current software implementation the formulation based on nodal variables is significantly more efficient. This thesis concludes with a road map for future work.

Acknowledgments

There are no enough words to convey my deep gratitude and respect for my thesis and research advisor, Dr. Carlos Christoffersen, for his valuable guidance, encouragement and support throughout my graduate studies. Also, I am grateful to Dr. Ehsan Atoofian and Dr. Xiaoping Liu for reviewing my thesis and providing me with valuable comments.

I would like to thank all my friends who have provided me with joy and warmth through these years. I would also like to thank my parents and my younger brother. They were always supporting me and encouraging me with their best wishes. Finally, I would like to thank my wife, Purvi Bhatt. She was always there cheering me up and stood by me through the good times and bad.

Tapankumar Kishorbhai Savalia

tsavalia@lakeheadu.ca

Contents

List of Figures

List of Tables

List of Symbols

$\boldsymbol{x}$ - Unknown nodal vector

$\boldsymbol{b}$ - Right hand side vector of linear matrix equation

x - Scalar variable

Z_0 - Reference impedance

$\boldsymbol{s}$ - Source vector

v^+, v^- - Voltage waves

$\boldsymbol{i_k}$ - Nonlinear current vector of subcircuit k

$\boldsymbol{i_I}$ - Interconnect current vector

$\boldsymbol{n}$ - Newton iteration index

$\boldsymbol{A_j}$ - Subcircuit matrix block $\boldsymbol{j}$

$\boldsymbol{J_j}$ - Jacobian matrix of subcircuit $\boldsymbol{j}$

$\boldsymbol{k}$ - Number of subcircuits

$\boldsymbol{j}$ - Subcircuit index

$\Delta\boldsymbol{x}$ - Newton-Raphson update

List of Abbreviations

CPU	- Central Processing Unit.
RF	- Radio Frequency
DC	- Direct Current
KCL	- Kirchhoff's Current Law
KVL	- Kirchhoff's Voltage Law
CAD	- Computer Aided Design
NR	- Newton-Raphson
MLNA	- Multilevel Newton Analysis
MLNR	- Multilevel Newton-Raphson
DD	- Domain Decomposition
PANEM	- Parallel Newton Method
PANEM_MUL	- Parallel Multilevel Newton Method
PANEM_MUL_L	- Parallel Multilevel Newton Method with Latency
OP	- Operating Point Analysis
EOP	- Operating Point Analysis with nodal variables
WAVEOP	- Operating Point Analysis with waves
HMAPS	- Hierarchical Multialgorithm Parallel Simulation

Chapter 1

Introduction

1.1 Motivation and Objective of This Study

As one of the most critical forms of pre-silicon simulation and verification, transistor-level circuit simulation (e.g., SPICE) is essential for the design of a very broad range of integrated circuits and systems such as custom digital integrated circuits (ICs), memories, analog, mixed signal, and radio-frequency (RF) designs [1]. Circuit simulation predicts circuit performance and makes it possible to disqualify a failing design for expensive chip fabrication. Equally, the ability of predicting circuit performance through simulation is at the core of any design process; it makes the implementation of complex integrated circuits technically feasible and economically viable while relaxing any heavy need for prototyping.

Performing expensive transistor- level circuit analysis consumes lots of CPU time. The simulation bottleneck significantly limits pre- silicon verification and design space exploration, contributing to long design turnaround time, suboptimal designs and even chip failures. With the advent of more complex device models and increased design complexity, high-capacity circuit simulation is strongly desirable in order to boost design productivity. One of the most effective ways to reduce the computing time is to use parallel processing. The necessary requirement for parallel processing is parallel hardware. Traditionally, the parallel processing was performed in supercomputers with multiple processors, but these computers were usually very expensive [2]. In networked parallel processing, each serial (or parallel) computer is used as a processing unit and data is transferred via a local area network, like Ethernet. In the meantime, the industry's shift to the multi-core processor technology and emergence of new types of accelerators has introduced new challenges and opportunities for addressing today's CAD problems, including circuit simulation. Because the slowdown in single-core clock frequency scaling, there are limits in the performance of single-threaded CAD applications, new parallel algorithms and tools, which are able to utilize parallel hardware, have attracted great renewed interest. Parallel circuit

simulation naturally comes into the picture under this context. To this end, the main challenge is to develop highly scalable parallel simulation techniques so as to tackle computationally challenging simulation tasks while maintaining high accuracy and robustness across a wide range of circuit applications.

Simulation of large circuits suffers from excessive computational cost. In general, simulation cost is proportional to S^a, where S represents the original matrix size and a depends on the sparsity of the circuit matrix. For typical circuits, a varies from 1.1 to 2.4 [3]. The computational cost would be high if the S is large. For modern electronic circuits, S can be very large, in the range of several millions. Hence, it is desirable to approach the circuit simulation problem by dividing the original circuit into several smaller subcircuits, and solving each smaller subcircuit independently and in parallel. Merging the subcircuit results will get the solution of the original circuit.

Hence, what is needed is a method for accurate and fast analysis of large circuits and formulations that effectively partition the given problem while providing a mechanism requiring minimum computational cost to synchronize the solution among different partitions/processors. There have been earlier attempts to develop parallel simulation capabilities on multiprocessors, and supercomputers, either custom built or commercially available [4], [5], [6], [7], [8], [9]. On the other hand, the recent industry's shift to multi and many core processor technology has literally made every modern-day desktop, server, and laptop a parallel computer [10], [11], [12], [13]. This shift toward chip multiprocessors (CMPs) reflects the fundamental performance and power tradeoffs in lieu of VLSI technology scaling. The main contribution of this thesis is to investigate the performance of some circuit decomposition techniques for efficient parallel circuit simulation. Effective parallel circuit simulation requires minimal communication between processors. This thesis presents two main approaches : Circuit decomposition based on nodal variables and based on scattering waves. In the former approach partitioned subcircuits exchange nodal variables i.e. voltage and current, while in the latter approach subcircuits exchange scattering waves.

A node-tearing process is used to divide the system Jacobian in blocks. The first formulation is based on the nodal voltages and currents at the interface nodes. Although this formulation is not new, until recently branch-tearing was preferred because it requires less number of variables. It

will be shown in this thesis that the node-tearing approach results in a simpler matrix structure that is more convenient for parallel analysis. In addition, the node-tearing formulation can be modified to use scattering wave variables at the subcircuit interfaces instead of voltages and currents. This approach is also explored in this work. A formulation based on wave variables is attractive because they can handle open- and short-circuit conditions without the numerical problems that may arise when using voltage and currents. For example, if a non-zero voltage is assumed across a short-circuit, the corresponding current is infinite. The use of waves also enables the use of a simpler convergent relaxation approach [3] to exchange information between subcircuits. Both reference algorithms have been implemented in the Cardoon circuit simulator [14].

1.2 Thesis Overview

The thesis is composed of four chapters. The basic concepts and literature review is presented in Chapter 2. Chapter 3 shows design procedure and simulation results of circuit decomposition based on nodal variables and scattering waves. In the last chapter conclusion and proposed direction of future work is discussed.

Chapter 2

Literature Review

2.1 Introduction

As circuit sizes increase, it is essential to improve the performance of simulations without sacrificing the accuracy of the results. The larger the circuit, the larger the computational cost. For modern electronics, circuits can be very large, in the range of several millions nodes. Hence it is desirable to approach the circuit simulation problem by dividing the original circuit into several small subcircuits by decomposition. The decomposition can be performed using specific partitioning algorithms. Here, we do not consider how the partition is performed but, as a guideline, the optimal partition has only a few connections when compared with the size of the subcircuit and subcircuit should be of about the same size for load balancing. This thesis focuses on how system of equations can be solved effectively in parallel assuming circuit is partitioned into several subsystems and hence this chapter presents different approaches to solve nodal equations efficiently in blocks rather than the partitioning approach. In all approaches it is assumed that circuit is readily partitioned into subcircuits.

In the following section, basic concepts like Newton method, nodal analysis and Diakoptics are explained, formulations based on domain decomposition are summarized in Section 2.3 and other formulations are discussed in Section 2.4.

2.2 Basic Concepts

2.2.1 Newton Method

Newton's method often called Newton-Raphson method, particularly in the engineering literature is the most successful method for the numerical solution of nonlinear problems provided with some differentiability. Because its idea of successive linearization is so fundamental, there are many possible applications.

Suppose that a solution of a nonlinear equation

$$f(x) = 0 \tag{2.1}$$

is to be found, where *f* is a differentiable function for which a root is sought. Newton's Method solves this nonlinear equation iteratively. Let, *f'(x)* be derivative of function *f*, and iteration index is *n*. At *n+1* iteration, by taking first order Taylor's expansion, we approximate nonlinear function *f(x)* into linear function:

$$f(x^{n+1}) = f(x^n) + f'(x^n)(x^{n+1} - x^n). \tag{2.2}$$

This is linearization of *f(x)* around *n+1*. Newton's method in one dimension is obtained by making x^{n+1} equal to the root of the linear approximation at *n+1* iteration. The correction (Δx^{n+1}) at iteration *n+1* is given by:

$$\Delta x^{n+1} = x^{n+1} - x^n = -\frac{f(x^n)}{f'(x^n)}. \tag{2.3}$$

Figure 2.1 explains Newton's method. Suppose that solution of the function *f(x)* is to be found then Newton method iterates with initial guess x^0. x^1 and x^2 are the approximations to the solution of *f(x)* at iteration 2 and 3, respectively. x^* is the solution of *f(x)*.

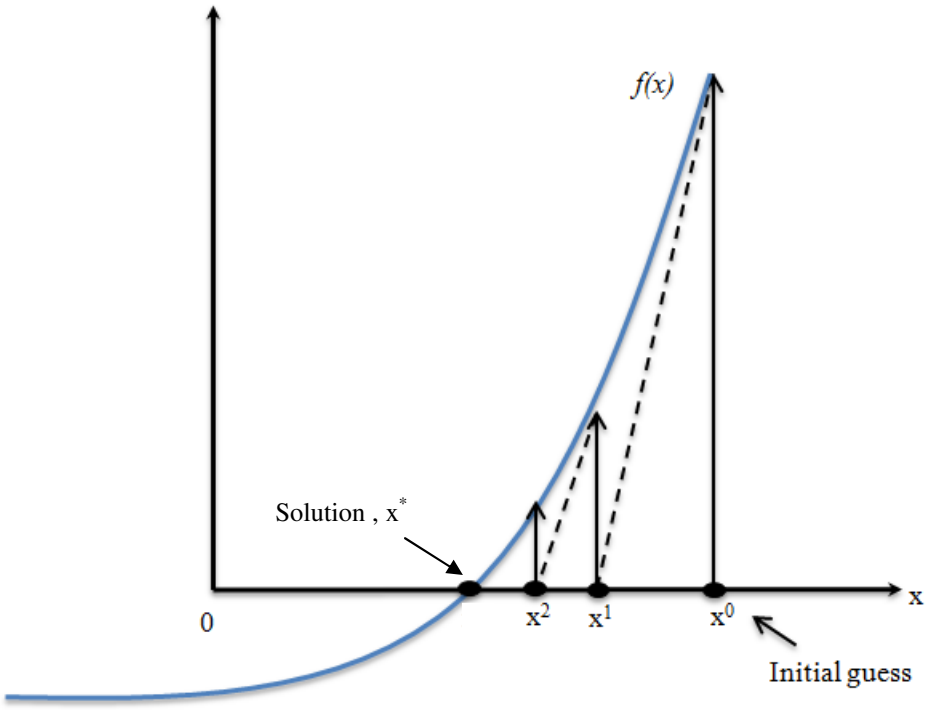


Figure 2.1 Newton's Method

Now for multidimensional Newton method, consider nonlinear system of equation,

$$\boldsymbol{F(x) = 0} \tag{2.4}$$

where $\boldsymbol{x}$ is nodal voltage vector, $\boldsymbol{F(x)}$ is a differentiable vector function. The correction at each iteration is the following:

$$\boldsymbol{\Delta x}^{n+1} = \boldsymbol{x}^{n+1} - \boldsymbol{x}^{n} = -\boldsymbol{J_F}(\boldsymbol{x}^{n})^{-1}\boldsymbol{F}(\boldsymbol{x}^{n}) \tag{2.5}$$

where $\boldsymbol{J_F}$ is Jacobian matrix of $\boldsymbol{F}$ and it is defined as,

$$\boldsymbol{J_F} = \begin{bmatrix} \dfrac{\partial F_1(x)}{\partial x_1} & \cdots & \dfrac{\partial F_1(x)}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial F_k(x)}{\partial x_1} & \cdots & \dfrac{\partial F_k(x)}{\partial x_k} \end{bmatrix}.$$

Equation (2.5) is solved iteratively until convergence. Convergence check is done by checking condition $\|\Delta x\| \leq tolerance$, where $\|\Delta x\| = x^{n+1} - x^n$. When iterations get close to the solution *i.e.* $\|\Delta x\| \approx tolerance$ then convergence rate is quadratic.

2.2.2 Nodal Analysis

The circuit equations can be created using nodal analysis [15]. Nodal equations are created by formulating Kirchhoff current law (KCL) for all nodes, except for the reference node. These nodal voltages are assigned with respect to a reference node. This reference node is denoted as ground. A simple circuit example for nodal analysis is shown in Figure 2.2.

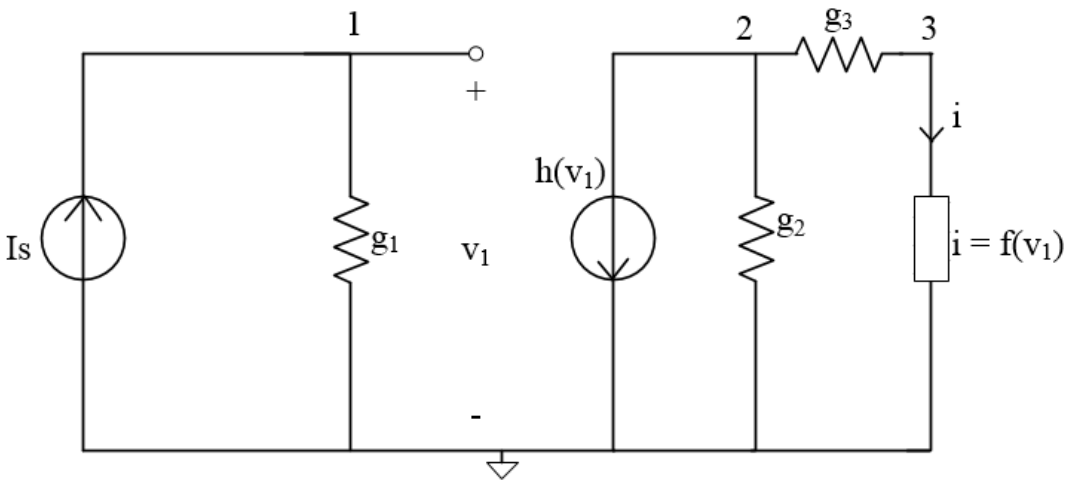


Figure 2.2 Example circuit to explain Nodal analysis

To write a nodal analysis for a circuit shown in Figure 2.2 first identify nodes and assign one node as the reference node. After that write KCL at each node. All nodal equations can be expressed as:

$$i = f(v). \tag{2.6}$$

Writing KCL for circuit shown in Figure 2.2 for each node except reference node gives:

KCL at node 1: $g_1 v_1 - I_s = 0$

KCL at node 2: $h(v_1) + g_2 v_2 + g_3(v_2 - v_3) = 0$

KCL at node 3: $-g_3(v_2 - v_3) + f(v_1) = 0.$ (2.7)

A set of equations (2.8) can be written in matrix form as:

$$\begin{pmatrix} g_1 & 0 & 0 \\ 0 & g_2 + g_3 & -g_3 \\ 0 & -g_3 & g_3 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} 0 \\ h(v_1) \\ f(v_1) \end{pmatrix} = \begin{pmatrix} I_s \\ 0 \\ 0 \end{pmatrix} \tag{2.8}$$

Now let the nonlinear system of equations be

$$\boldsymbol{F(x) = s}$$

$$\boldsymbol{F(x) - s = 0} \tag{2.9}$$

where $\boldsymbol{x}$ is nodal voltage vector, $\boldsymbol{F}$ is a differentiable function and $\boldsymbol{s}$ is source vector. Now comparing Equation (2.8) and (2.9) yields :

$$\boldsymbol{F(x)} = \underbrace{\begin{pmatrix} g_1 & 0 & 0 \\ 0 & g_2 + g_3 & -g_3 \\ 0 & -g_3 & g_3 \end{pmatrix}}_{\boldsymbol{G}} \underbrace{\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{pmatrix} 0 \\ h(v_1) \\ f(v_1) \end{pmatrix}}_{\boldsymbol{i(x)}}, \quad \boldsymbol{s} = \begin{pmatrix} I_s \\ 0 \\ 0 \end{pmatrix}$$

Hence from Equation (2.9)

$$\boldsymbol{F(x) - s = G\,x + i(x) - s = 0} \tag{2.10}$$

where, $\boldsymbol{Gx}$ is linear contribution of the function $\boldsymbol{F}$ and $\boldsymbol{i(x)}$ is nonlinear contribution of function $\boldsymbol{F}$. Applying Newton's method to Equation (2.10) will get :

$$\boldsymbol{x^{n+1} = x^n - J_F^{-1}\left(F(x^n) - s\right)}$$

$$\boldsymbol{J_F\, \Delta x^{n+1} = -\left(F(x^n) - s\right) = s - Gx^n - i(x^n)} \tag{2.11}$$

But $\boldsymbol{J_F = G + J_i}$. where $\boldsymbol{J_F}$ is Jacobian of function $\boldsymbol{F}$ and $\boldsymbol{J_i}$ is Jacobian of current vector $\boldsymbol{i(x)}$. Using this relation Equation (2.11) can be written as:

$$\boldsymbol{[G + J_i]\, \Delta x^{n+1} = s - Gx^n - i(x^n)}. \tag{2.12}$$

2.2.4 Diakoptics

Diakoptics [16] is tearing down an electric network into sub-systems, in other words the circuit is partitioned into subcircuits. Figure 2.3 shows a circuit network $\prod$ partitioned into three subcircuits. Components outside of subcircuits are part of the interconnect block.

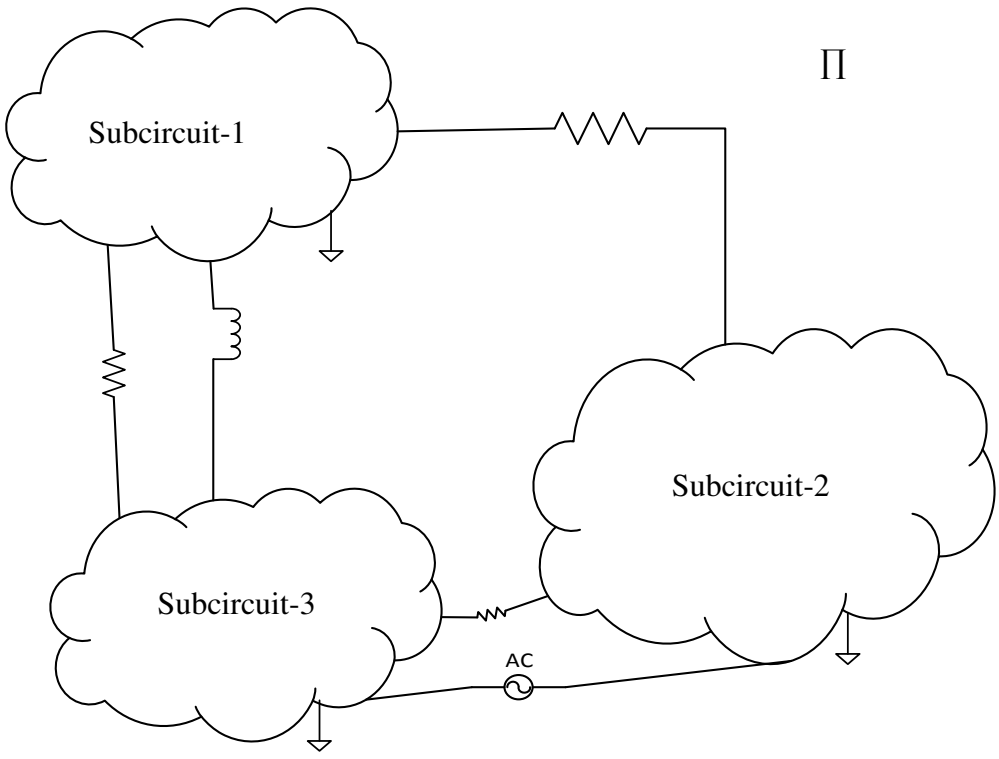


Figure 2.3 General Circuit Diagram to explain Diakoptics

Kron [17], [18] derived the equations resulting after partitioning a network into *k* subcircuits. Now suppose linear circuit network $\prod$ is defined by:

$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b} \tag{2.12}$$

where $\boldsymbol{A}$ is nodal matrix, $\boldsymbol{x}$ is unknown voltage vector and $\boldsymbol{b}$ is source vector. Now Equation (2.12) could be partitioned into *k* subsystems :

$$\begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ k \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & \cdots & k \end{array} \\ \begin{pmatrix} \boldsymbol{A}_1 & & & \boldsymbol{N}_1 \\ & \boldsymbol{A}_2 & & \boldsymbol{N}_2 \\ & & \ddots & \vdots \\ & & \boldsymbol{A}_k & \boldsymbol{N}_k \\ \boldsymbol{M}_1 & \boldsymbol{M}_2 & \cdots \; \boldsymbol{M}_k & \boldsymbol{C} \end{pmatrix} \end{array} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_k \\ \boldsymbol{x}_{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_k \\ \boldsymbol{b}_{k+1} \end{pmatrix} \tag{2.13}$$

where, $\boldsymbol{A_j}$ with $j=1, 2, ..., k$, is a matrix representing subcircuit j, depending on circuit partitioning approach the interconnect network spreads in $\boldsymbol{A}_j$, $\boldsymbol{N}_j$, $\boldsymbol{M}_j$ and $\boldsymbol{C}$, $\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x}_k$ are unknown internal nodal voltage vectors of subcircuits, $\boldsymbol{x_{k+1}}$ is interconnect nodal voltage vector, $\boldsymbol{b_1}, \boldsymbol{b_2}, ..., \boldsymbol{b_k}$ are source vectors of subcircuits and $\boldsymbol{b}_{k+1}$ is the source vector for interconnect block. There are two types of circuit decomposition is possible: branch tearing and node tearing. If circuit decomposition is done using branch tearing, links connecting to subcircuits are distributed to $\boldsymbol{A}_j$, $\boldsymbol{N}_j$, $\boldsymbol{M}_j$ and $\boldsymbol{C}$ blocks and create dependency between subcircuits. Node tearing will be discussed in detail in Chapter 3.

To solve Equation (2.13), unknown voltage vector for j^{th} subcircuit is found :

$$\boldsymbol{x}_j = \boldsymbol{A}_j^{-1} (\boldsymbol{b}_j - \boldsymbol{N}_j \boldsymbol{x}_{k+1}) \tag{2.14}$$

Interconnect nodal voltages can be found as follows:

$$\sum_{j=1}^{k} \boldsymbol{M}_j \boldsymbol{A}_j^{-1} (\boldsymbol{b}_j - \boldsymbol{N}_j \boldsymbol{x}_{k+1}) + \boldsymbol{C} \; \boldsymbol{x}_{k+1} = \boldsymbol{b}_{k+1}$$

$$\left[\boldsymbol{C} - \sum_{j=1}^{k} \boldsymbol{M}_j \boldsymbol{A}_j^{-1} \boldsymbol{N}_j \right] \boldsymbol{x}_{k+1} = \boldsymbol{b}_{k+1} - \sum_{j=1}^{k} \boldsymbol{M}_j \boldsymbol{A}_j^{-1} \boldsymbol{b}_j \tag{2.15}$$

Now to evaluate system of equations (2.13), interconnect nodal voltage $\boldsymbol{x}_{k+1}$ is calculated first from Equation (2.15) and then subcircuit nodal voltage $\boldsymbol{x}_j$ can be found from Equation (2.14).

For an example of Diakoptics consider a linear circuit network shown in Figure 2.4. This circuit is divided in three subsystems: two subcircuits and one interconnect block.

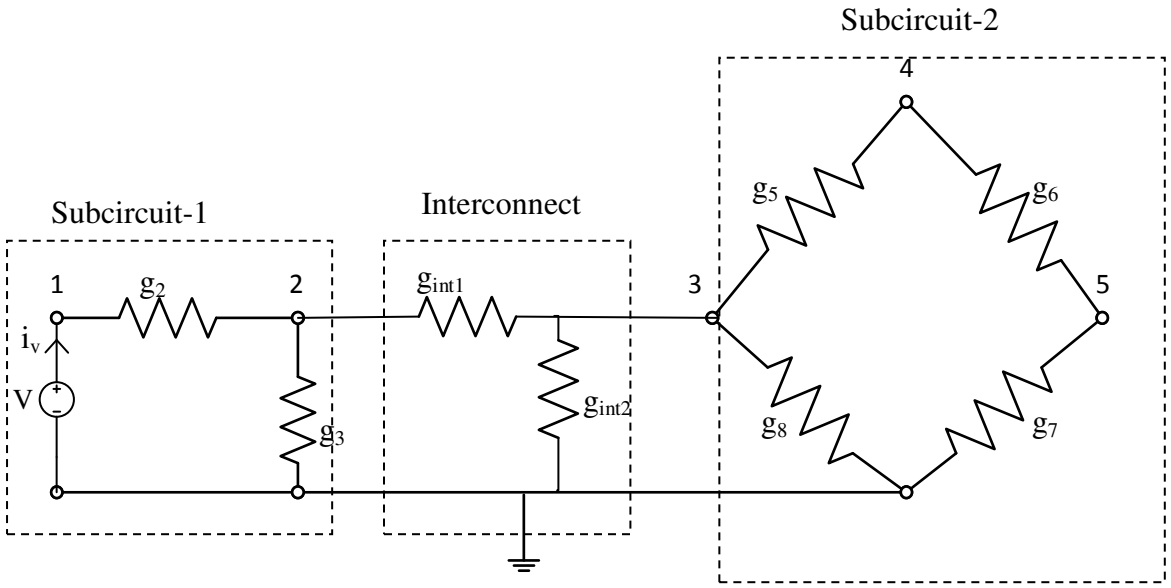
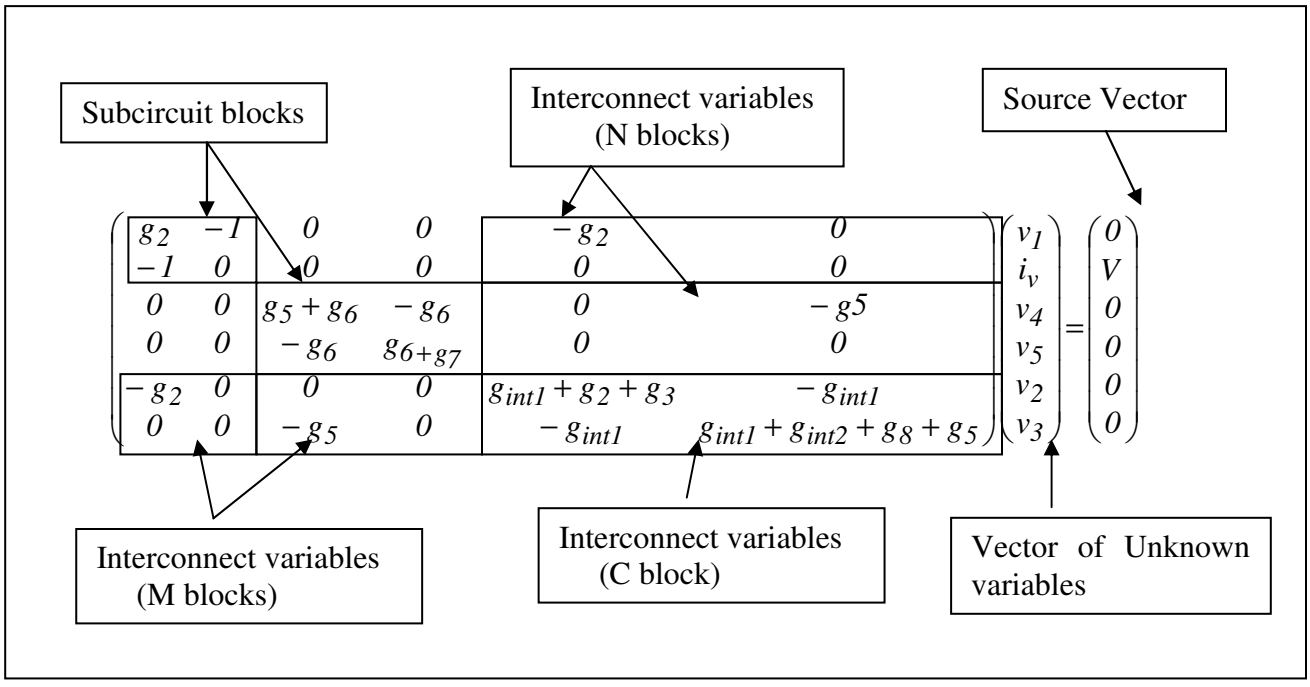


Figure 2.4 Example Circuit for Diakoptics

Figure 2.5 shows system of equations corresponding to the circuit shown in Figure 2.4.



$$\begin{pmatrix} g_2 & -1 & 0 & 0 & -g_2 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_5+g_6 & -g_6 & 0 & -g5 \\ 0 & 0 & -g_6 & g_{6+g_7} & 0 & 0 \\ -g_2 & 0 & 0 & 0 & g_{int1}+g_2+g_3 & -g_{int1} \\ 0 & 0 & -g_5 & 0 & -g_{int1} & g_{int1}+g_{int2}+g_8+g_5 \end{pmatrix} \begin{pmatrix} v_1 \\ i_v \\ v_4 \\ v_5 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ V \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 2.5 System of Equation

Likewise, for a nonlinear circuit, Diakoptics is applied to solve each Newton iteration.

2.3 Formulations Based on Domain Decomposition

Domain decomposition method refer to a collection of divide-and-conquer techniques which have been primarily developed for solving Partial Differential Equations [19], [20]. Domain decomposition refers to a class of methods for solving linear or nonlinear systems of equations, primarily arising from the discretization of partial differential equations (PDEs). In a way that is mostly relevant to the focus of this thesis, a domain decomposition method finds the solution to a large system by subdividing it into smaller sub-domains and solving these sub-domains separately. This section discusses different domain decomposition approaches.

2.3.1 Multilevel Newton Method [21]

Multilevel Newton Method

To speed-up simulation, one possible improvement that can be achieved from NR method is parallelization. In order to further improve the speed, other iteration methods than NR iteration may be utilized. Digital circuits are usually modular, latent, and unidirectional *i.e.* loosely coupled. Because block, waveform, and nonlinear relaxation methods utilize these properties, they have been found suitable for this kind of circuits. These methods cannot apply effectively to the analog circuits, which usually are tightly coupled. Multilevel Newton method is one of the methods that can be effectively applied in parallel processing [21].

Two characteristics of many electronic circuits are discussed for a more efficient analysis :

1. Many electronic circuits consist of identical repetitive sub-networks. This characteristic is utilized by macromodeling.
2. Many electronic networks contain sub-networks which are inactive *i.e.* their electrical variables are constant most of the simulation time.

Macromodels

A macromodel of a network is defined as a set of nonlinear and/or time varying elements simulating external behaviour of the sub-network. It consists of a set of nonlinear and/or time

varying elements. Some papers e.g. [22] use macromodels represented by circuit elements or equations which approximate the external behaviour of subnetwork whereas reference [21] define macromodel such that the external behaviour of the circuit is exactly represented by the macromodel.

Consider an example for macromodel: Let $\prod$ be the large-scale network composed of interconnected sub-networks $S_i = S_1, S_2, S_3$ and S_4 (Figure 2.6).

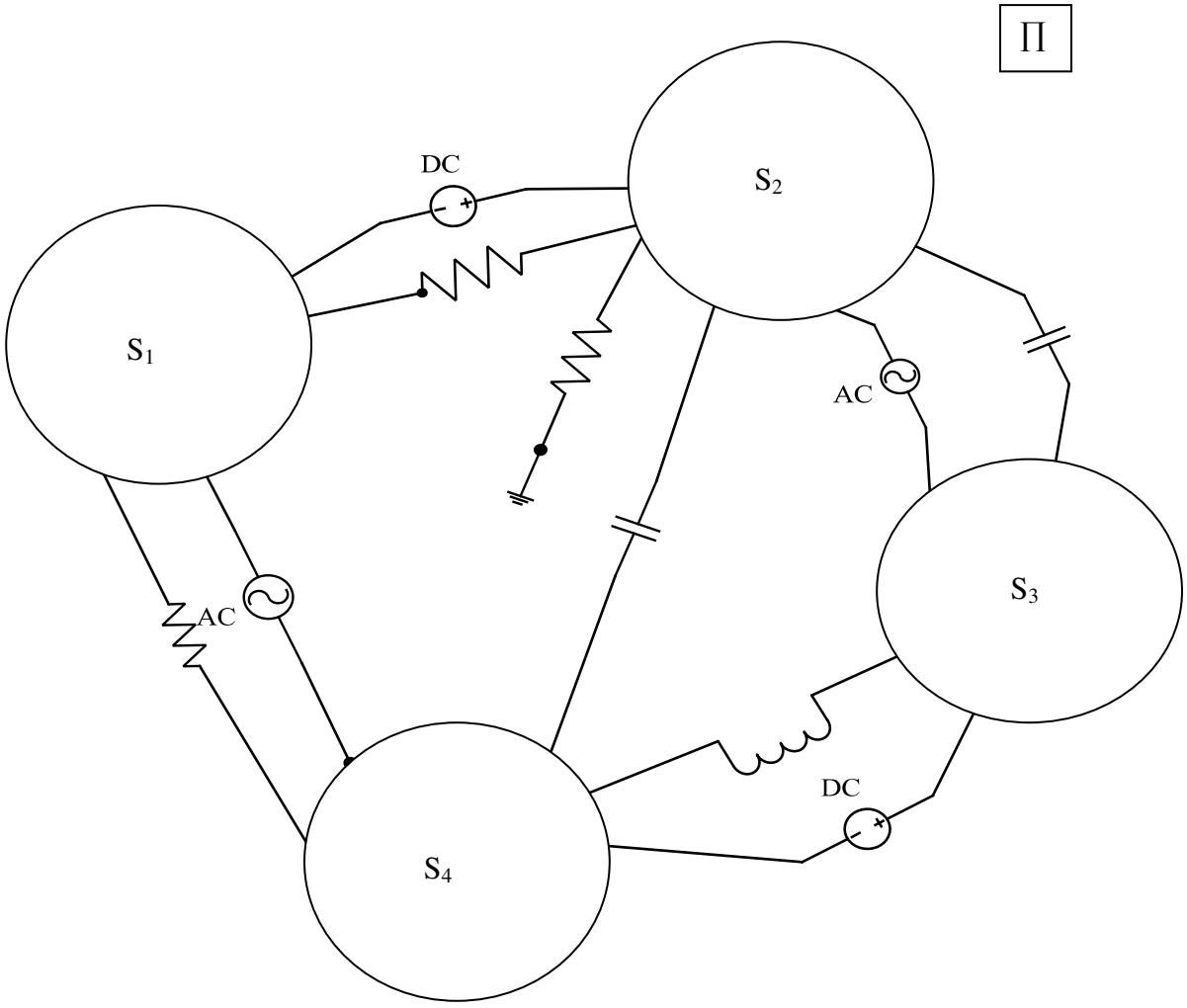
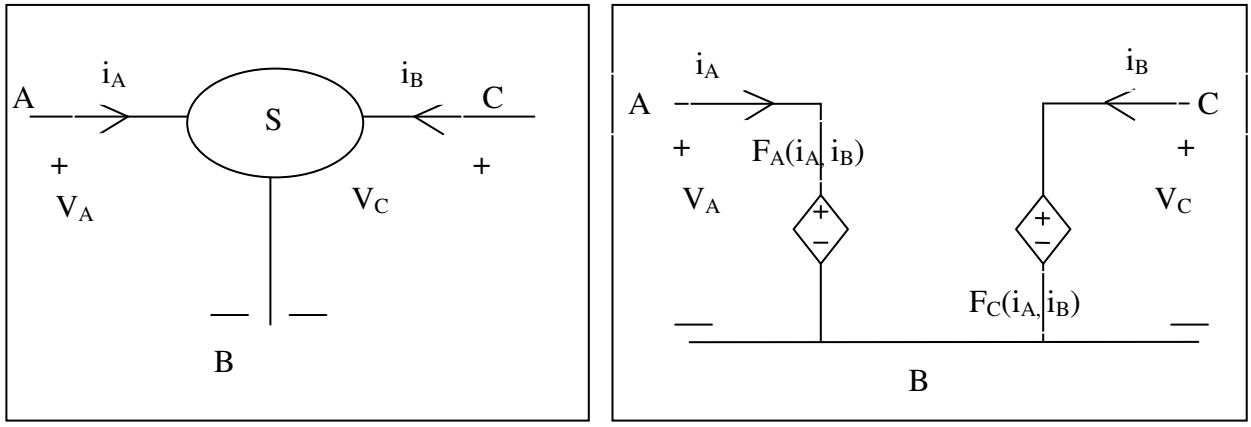


Figure 2.6 Example circuit network ($\prod$) to explain macromodel

Let S be a subnetwork (Figure 2.7(a)) of whole circuit network ∏ (Figure 2.6) to be represented by macromodel. Figure 2.7(b) is macromodel of subnetwork S.



(a) **(b)**

Figure 2.7 Example of macromodel (a) subnetwork S of circuit network ∏ (b) Macromodel of subcircuit S which represents its external behaviour.

The algorithm of multilevel newton method using Macromodel is as follows:

Let the equations describing behaviour of circuit network ∏ having only one subcircuit *S*,

$$\boldsymbol{F(u,\ G(\,x_{int}\,),\ x_{ext}\,) = 0} \tag{2.16}$$

where $\boldsymbol{u}$ is the vector of inputs, $\boldsymbol{x_{int}}$ is internal variables of subcircuit *S*, $\boldsymbol{G(\,x_{int}\,)}$ is a macromodel of subnetwork *S*, $\boldsymbol{x_{ext}}$ is the vector of interconnect (external) variables in ∏ not communicating with *S*.

Applying Newton's algorithm to Equation (2.16) by updating only outer variables and keeping internal variables of macromodel $G(\,\boldsymbol{x_{int}}\,)$ constant will get :

$$\boldsymbol{x_{ext}^{n+1} = x_{ext}^{n} - (J_F(x_{ext}^{n}))^{-1}\ F(x_{ext}^{n})} \tag{2.17}$$

where $\boldsymbol{J_F}$ is Jacobian of function $\boldsymbol{F}$ (Equation 2.16) with respect to variable $\boldsymbol{x_{ext}}$ ($\boldsymbol{x_{int}}$ is constant), $\boldsymbol{x_{ext}^{n+1}}$ is value at next Newton iteration step, $\boldsymbol{x_{ext}^{n}}$ is previous value of Newton iteration and *n* is iteration index.

Now, to evaluate macromodel $G(x_{int})$ another Newton method is applied to equations representing macromodel. Equation (2.18) represents macromodel :

$$H(u, x_{int}, y) = 0 \tag{2.18}$$

where y is output vector. In this Newton method only update inner variables x_{int} and keep outer variables u constant :

$$x_{int}^{n+1} = x_{int}^{n} - (J_H(x_{int}^{n}))^{-1} H(x_{int}^{n}) \tag{2.19}$$

J_H is Jacobian of macromodel with respect to x_{int} and y. Keep iterating until it converges. Convergence condition is given by :

$$\left\| \Delta x_{int} \right\| < \left\| \Delta x_{ext} \right\|$$

where $\Delta x_{int} = x_{int}^{n+1} - x_{int}^{n}$ is error in subcircuit nodal voltages and $\Delta x_{ext} = x_{ext}^{n+1} - x_{ext}^{n}$ is error in interconnect nodal voltages. Here, Newton's algorithm applied twice on Equations (2.16) and (2.18), that's why this method is called multilevel Newton's algorithm.

Latency

The second characteristic of the network is latency. Suppose that in the electrical network to be analyzed, at any particular time t_1, most of the subcircuits are latent *i.e.* the value of their electrical variables remain constant. Latency is used to speed up the analysis in logic simulation when only the active part of the circuit is analysed which is called event driven simulation. Basically, when any subcircuit is found to be latent at a certain instant of time e.g. t_2, then obviously no function or Jacobian evaluations are needed to find the value of the subcircuit variables at all the subsequent time steps until a change in the input variables of the subnetwork occurs. In other words, the corresponding element in Jacobian of the circuit equation are not evaluated at t_2 and the value of the subcircuit variables is set to the one taken at time t_1 [21]. Latency approach is more effective for digital circuits as they are usually modular and latent. Use of latency can achieve significant savings in computer time. An additional advantage is that it can easily deal with asynchronous designs. Latency can be used for timing analysis including the usage of multi-delay model. This aspect is not implemented in this thesis.

2.3.2 New Approach for Parallel Circuit Simulation [23]

A new approach for parallel simulation of very large scale integration (VLSI) circuit on a transistor level is presented [23]. Authors proposed circuit partitioning algorithms along with formulation for parallel circuit simulation. Three algorithms are presented:

I. Parallel Newton Method
II. Parallel Multilevel Newton Method
III. Parallel Multilevel Newton Method with Latency

Supose that linear system of equations to be solved at each Newton iteration is presented by :

$$\boldsymbol{Ax = b} \quad . \tag{2.20}$$

Diakoptics [17] is used to implement parallel simulation. These three algorithms are implemented on Equations (2.14) and (2.15) mentioned in Section 2.2.4 Diakoptics. For quick reference equations are rewritten below :

$$\boldsymbol{x}_j = \boldsymbol{A}_j^{-1} (\boldsymbol{b}_j - \boldsymbol{N}_j \boldsymbol{x}_{k+1})$$

$$\left[\boldsymbol{C} - \sum_{j=1}^{k} (\boldsymbol{M}_j \boldsymbol{A}_j^{-1} \boldsymbol{N}_j) \right] \boldsymbol{x}_{k+1} = \boldsymbol{b}_{k+1} - \sum_{j=1}^{k} \boldsymbol{M}_j \boldsymbol{A}_j^{-1} \boldsymbol{b}_j \, .$$

Decomposition of Equation (2.20) leads to the decoupled nonlinear system:

$$\boldsymbol{F}(\boldsymbol{x}_j, \boldsymbol{x}_{k+1}) = \boldsymbol{0} \tag{2.21}$$

$$\boldsymbol{G}(\boldsymbol{x}_j, \boldsymbol{x}_{k+1}) = \boldsymbol{0} \tag{2.22}$$

where $j = 1, \ldots, k$, $\boldsymbol{F}$ is subcircuits ($\boldsymbol{A}$ blocks) and $\boldsymbol{G}$ is interconnect (coupling) system whose elements are spread in $\boldsymbol{M}$, $\boldsymbol{N}$ and $\boldsymbol{C}$ blocks (Equation 2.13). Every subcircuit j is represented by its own nonlinear system (2.21) and dependencies between each subcircuits are given by Equation (2.22). Figure 2.8 shows Newton's method for the decoupled nonlinear equation systems (Equations (2.21) and (2.22)) [23]. Computation is divided between master and slave processors.

Start : $(x_1^0,.....,x_j^0,x_{k+1}^0)$; $n = 0$

repeat

Slaves:

Do parallel j=1,....,k

$J = L_j\ U_j$ (LU decomposition)

$$S_j^n = M_j * (A_j)^{-1} * N_j$$
$$w_j^n = M_j * (A_j)^{-1} * b_j$$

(Forward and backward substitution)

send S_j^n and w_j^n to master

end_j

Master:

$$S^n = C - \sum_{j=1}^{k} S_j^n$$

$$w^n = G(x_1^n,...., x_k^n, x_{k+1}^n) - \sum_{j=1}^{k} w_j^n$$

$$S^n * \Delta x_{k+1}^n = w^n$$ (Serial computation of linear system)

send Δx_{k+1}^n to slaves to calculate internal nodal voltages.

Slaves:

Do parallel k=1,...., j

Do forward and backward substitution

$$\Delta x_j^n = (J)^{-1} * (F_j(x_k^n, x_{k+1}^n) - N_j * \Delta x_{k+1})$$

$$\Delta x_j^{n+1} = x_j^{n+1} - x_j^n$$ (Update)

end_j

Figure 2.8 Parallel Newton's algorithm

The algorithms shown in Figure 2.8 can be classified in three main steps:(i) Slave processors calculate S_j and w_j for each subcircuit and sent these variables to master processor. (ii) Master processor calculates S^n, w^n, interconnect nodal voltage vector x_{k+1} and send x_{k+1} to slave processors. (iii) Slave processors calculates subcircuit nodal voltages for each subcircuit using x_{k+1}. However, this simple approach has the following potential limitation. Since several Newton iterations may be needed before the solution to the nonlinear system converges, there may be a considerable amount of inter-processor communication, which limits the efficiency of the parallel simulation.

A trade-off can be made between communication and computation by introducing the multilevel Newton Method. In this case, one Newton iteration consists of an inner iteration loop and an outer Newton update step. In the inner iteration loop, each local nonlinear equation is iteratively solved to converge under a fixed outer (interface) variable vector $\Delta \boldsymbol{x}_{k+1}$ to update all local variables $\boldsymbol{x}_1, \boldsymbol{x}_2 ...., \boldsymbol{x}_k$. Then an outer Newton step is taken to update outer variable vector $\Delta \boldsymbol{x}_{k+1}$ based on the solutions received from the slaves. Finally, to complete one Newton iteration for the entire system, a Newton step is taken to correct all local variables $\boldsymbol{x}_1, \boldsymbol{x}_2 ...., \boldsymbol{x}_k$ using the updated $\Delta \boldsymbol{x}_{k+1}$. Since more work is done at the slave level in the above multilevel Newton method, the number of top-level Newton iterations may be reduced, leading to less communication between the slaves and the master. The conditions under which the multilevel Newton method maintains local quadratic convergence was provided in [21].

Latency can be also exploited under this multilevel framework [21], [23]. PNAM_MUL method is efficient if there is a sufficient decrease of the interconnect variables in the outer iteration, in that case slave processors do not need to evaluate the outer derivative. Sending the matrices S_j to the master processor causes the main part of the communication. In case of latency, these matrices do not have to be sent to the master process. Only the inner variables have to be transmitted to the master. One condition latency = *true* is added in PANEM_MUL algorithm. If this condition satisfies then slave neither have to evaluate S_j matrices nor have to send it to the master process.

Simulation Results

The parallel multilevel Newton method was demonstrated as part of the TITAN simulator, running on both computer clusters and shared-memory multiprocessors [23]. Good parallel speedups were demonstrated up to eight processors. Table 2.1 shows the comparison of three algorithms PANEM, PANEM_MUL AND PANEM_MUL_L with required number of iterations to simulation CPU time to simulate circuit industry 1. Table 2.2 shows the CPU time for simulation, speedup and number of interconnect for circuit industry 5.

Table 2.1 Simulation results of PANEM, PANEM_MUL and PANEM_MUL_L for circuit industry 1

Method	#iterations(DC/TR)	Real simulation CPU-time (min:sec)
PANEM	59/222	34:35
PANEM_MUL	11/158	27:28
PANEM_MUL_L	11/158	21:59

Table 2.2 Simulation results for circuit Industry 5(large industry circuit having approx. 50k MOSFETs)

	1 processor	4 processors	8 processors
Real simulation CPU-time(hour:min:sec)	5:07:12	1:17:49	0:39:26
Speedup	-	3.95	7.79
#interconnect	-	7	8

Modified Multilevel Newton Method [24]

The New Multilevel Newton-Raphson Method is modified from the multilevel Newton method [21]. Good global convergence can be achieved by adjusting inner iterations and local quadratic convergence is achieved [24].

Authors used Diakoptics [17] method to implement parallel circuit simulation as discussed in Section 2.2.4. This approach is slightly modified from Reference [21]. It is the exact same process from Equations (2.10) to (2.12) without defining a macromodel. In New Multilevel

Newton Method, instead of taking global NR steps [24], the iterations are taken at multiple levels. Between each outer NR step only fixed number of inner iterations (q) are taken to synchronise local and global convergence *i.e.* for load balancing. At each outer iteration instead of updating only outer variables, update all variables. In this way overall convergence would be faster by achieving local quadratic convergence.

Simulation Results of modified Multilevel Newton Method

Simulation Results of modified Multilevel Newton Method for a circuit with 1440 BJTs and 7746 nodes is shown in Table 2.3, where *p* is number of outer iterations and *q* is the number of inner iterations. This circuit is partitioned into 3 subcircuits and simulated with three processors. Processing time is the time to decompose the whole circuit network into subcircuits, symbolic recording of sparse matrix etc.

Table 2.3 Simulation results of modified multilevel Newton method

q	p	Preprocessing time	Iteration time	total
0	19	9.0 s	2.6 s	11.6
1	13	9.0 s	2.6 s	11.6
2	6	9.0 s	1.9 s	10.9
3	5	9.0 s	1.7 s	10.7

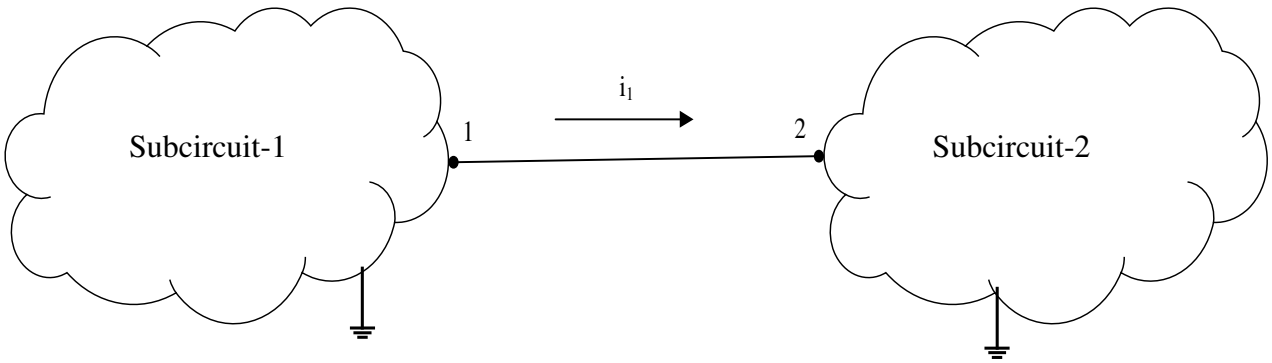
Even though load balancing can be achieved by fixed number of iterations, total number of iterations for small subcircuits will be different than big subcircuit blocks *i.e.* small subcircuit may converge faster than big subcircuit block having more internal nodes compare to small subcircuits.

2.3.3 Formulation with Binary Link [25]

This algorithm formulates the interface vectors between partitions, through binary vectors, leading to enhanced parallelism, scalability and reduced CPU costs while synchronizing the solutions between various partitions. The CPU cost per iteration as a function of the number of links *L* between subcircuits is in the order of L^2. That leads to poor scalability as its complexity increases. This reference [25] proposed an algorithm that exhibits superior scalability as its complexity increases only in order of *L*.

Node tearing (decomposition) technique [16], [26] is used to partition circuit into several subcircuits. Consider a circuit divided in two subcircuits using node tearing as shown in Figure 2.9. Figure 2.9 shows procedure to find out external current along this link (current i_1) by finding Thévenin equivalent. Open circuit port voltages V_{th1} and V_{th2} can be found with independent sources of subcircuits enabled as show in Figure 2.9 (b). Thévenin equivalent impedance for each subcircuit can be calculated by connecting a unit current source to each port and deactivate the independent sources of circuits. Now from open circuit voltages V_{th1} and V_{th2} and Thévenin equivalent impedance (Z_{th1} and Z_{th2}) external current i_1 flowing from subcircuit-1 to subcircuit-2 can be found by

$$i_1 = \frac{V_{th1} - V_{th2}}{Z_{th1} + Z_{th2}} \ . \tag{2.23}$$



(a)

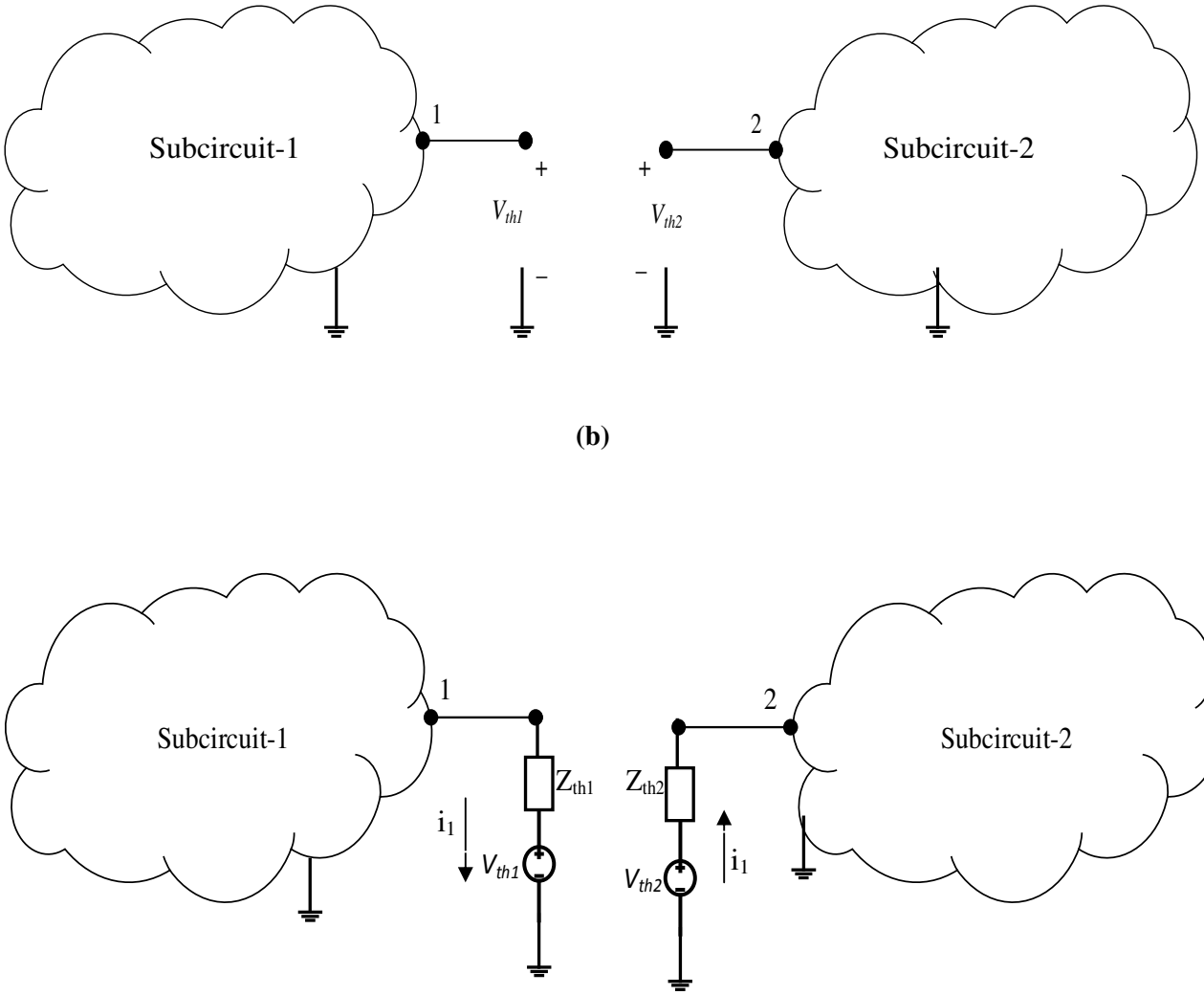


Figure 2.9 Thévenin equivalent measurement to find external current (a) Example of node tearing (b) Measurement of V_{th} (c) Link current (i_1) measurement (Equation 2.23)

A binary selector column vector N_1 is constructed where the row of *node 1* is +1 and rest are zero. Number of rows are equal to subcircuit block A_1, which is MNA matrix for subcircuit 1. Likewise binary selector column vector N_2 is constructed for subcircuit 2 where row of *node 2* is -1 and rest are zero.

The system of equations for the circuit shown in Figure 2.9 is :

$$\begin{pmatrix} \boldsymbol{A_1} & \boldsymbol{0} & \boldsymbol{N_1} \\ \boldsymbol{0} & \boldsymbol{A_2} & \boldsymbol{N_2} \\ \boldsymbol{N_1^T} & \boldsymbol{N_2^T} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{v_1} \\ \boldsymbol{v_2} \\ \boldsymbol{i_1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{pmatrix}. \tag{2.24}$$

Now the overall matrix for *k* subcircuits will have the following form :

$$\boldsymbol{F(x)} = \begin{pmatrix} \boldsymbol{A_1} & & & & \boldsymbol{N_1} \\ & \boldsymbol{A_2} & & & \boldsymbol{N_2} \\ & & \ddots & & \vdots \\ & & & \boldsymbol{A_k} & \boldsymbol{N_k} \\ \boldsymbol{N_1^T} & \boldsymbol{N_2^T} & \cdots & \boldsymbol{N_k^T} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \vdots \\ \boldsymbol{x_k} \\ \boldsymbol{i_E} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \vdots \\ \boldsymbol{b_k} \\ \boldsymbol{0} \end{pmatrix} \tag{2.25}$$

where, $\boldsymbol{i_E}$ is external current vector flowing from one subcircuit to another. Now rewriting Equations (2.14) and (2.15) using notation used in Equation (2.25)

$$\boldsymbol{x}_j = \sum_{j=1}^{k} \boldsymbol{A}_j^{-1} (\boldsymbol{b}_j - \boldsymbol{N}_j \boldsymbol{i}_E) \tag{2.26}$$

$$(-\sum_{j=1}^{k} (\boldsymbol{N}_j^{\boldsymbol{T}} (\boldsymbol{A}_j)^{-1} \boldsymbol{N}_j) \boldsymbol{i_E} = -\sum_{j=1}^{k} \boldsymbol{N}_j^{\boldsymbol{T}} (\boldsymbol{A}_j)^{-1} \boldsymbol{b}_j \tag{2.27}$$

Now for more than two subcircuits (e.g. k subcircuits) the Thévenin equivalent impedance matrix defined is:

$$\boldsymbol{Z} = -\sum_{j=1}^{k} (\boldsymbol{N}_j^{\boldsymbol{T}} (\boldsymbol{A}_j)^{-1} \boldsymbol{N}_j \tag{2.28}$$

and the external currents flowing from one subcircuit to another is given by

$$\boldsymbol{i_E} = -\boldsymbol{Z}^{-1} \sum_{j=1}^{k} \boldsymbol{N}_j^{\boldsymbol{T}} (\boldsymbol{A}_j)^{-1} \boldsymbol{b}_j \,. \tag{2.29}$$

using relation given in (2.28) in Equation (2.27). Now the entire system can be solved by two Equations (2.26) and (2.29) iteratively. Matrices $\boldsymbol{M}_j$, $\boldsymbol{N}_j$ and $\boldsymbol{C}$ in Equation (2.13) for [21], [23] and [24] approaches contain original nodal variables and are non-binary matrices. Consequently, solution cost of interconnecting equations (solved on a master processor) as well as the

communication cost among slaves and the master processor grow rapidly with the increasing number of partitions. This causes poor scalability with the increasing number of processors and partitions. Reference [25] minimizes the computations required for interfacing various parallel blocks as well as minimizes the communication overhead between the processors involved. This is accomplished by efficient form of node splitting, during Newton Raphson iterations, at any time point. At each NR iteration, since the resulting circuit is linear, the technique of node tearing can be applied at the identified partitioning nodes, leading to coupling vectors (linking various resulting subcircuits) that are purely binary in nature, and an impedance matrix whose dimension depends on the number of links between various partitions.

This approach is similar to the formulations presented in Section 3.1 of this thesis. That formulation was developed independently of this reference.

Simulation Results

Here each circuit network is simulated with two implementations : one using node tearing and other using branch tearing. The circuit partitioning is performed by hMETIS [27], [28]. Each subcircuit was simulated on a single processor. In reference [25] simulation is done using up to 16 CPUs but here to summarize simulation result with only 8 processors are shown in Table 2.4 with analysis time and speed up. The speedup is measured relative to a standard simulation using a traditional LU solver, with no parallelism or partitioning. Table 2.4 shows simulation result summary of DSP example, SRAM, dual SRAM and Array Multiplier Example.

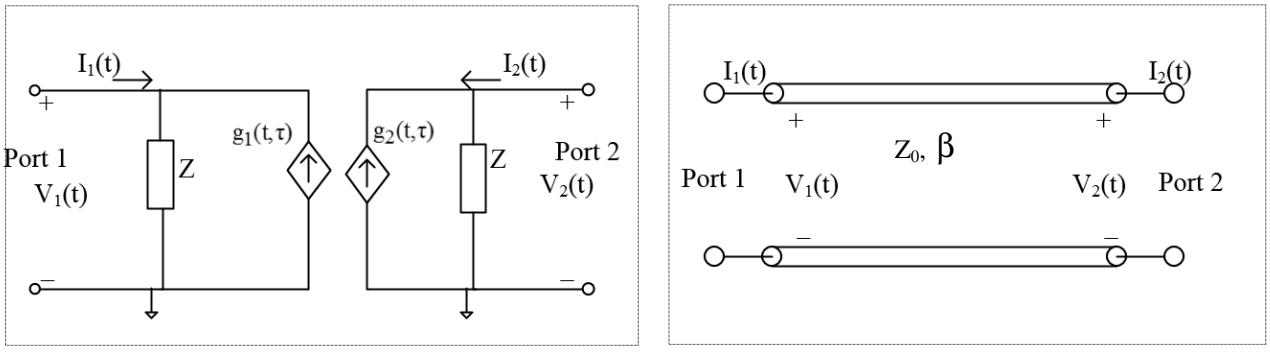
Table 2.4 Performance results with 8 processors

Examples	Branch Tearing		Node Tearing	
	Analysis Time	**Speed up**	**Analysis time**	**Speed up**
DSP	480.4 s	2.0	187.2s	5.1
SRAM	399.0 s	1.9	148.2s	4.6
Dual SRAM	378.0s	2.2	139.0s	6.0
Array Multiplier	682.1s	1.9	261.3s	4.8

2.4 Other Formulations

The technique proposed in [29] exploits the inherent delay present within some circuits. This delay element is used to partition a circuit network into several subcircuits and these subcircuits can be simulated on different cores of a shared-memory CPU. A delay element interfacing subcircuits is used to formulate the whole domain simulation.

Figure 2.10 shows delay elements. Figure 2.10(a) shows state-variable model of an ideal delay element and Figure 2.10(b) shows electrical circuit equivalent of the ideal lossless transmission line. The state variable model replicates a bidirectional delay so that circuit behavior at either port of the element does not affect the circuit at the opposite port until after a delay, τ.



(a) **(b)**

Figure 2.10 Delay Elements. (a) Ideal state variable based delay element. (b) Ideal lossless transmission line.

The simulated behavior of the delay element is given by the behavior of the state variables g_1 and g_2, which depend upon the past voltage and current at ports 2 and 1, respectively. Figure 2.11 (a) shows two subcircuits connected with a delay element. This delay element partitioned into sub-delay elements as shown in figure 2.11 (b) and those partitions can be iterated independently.

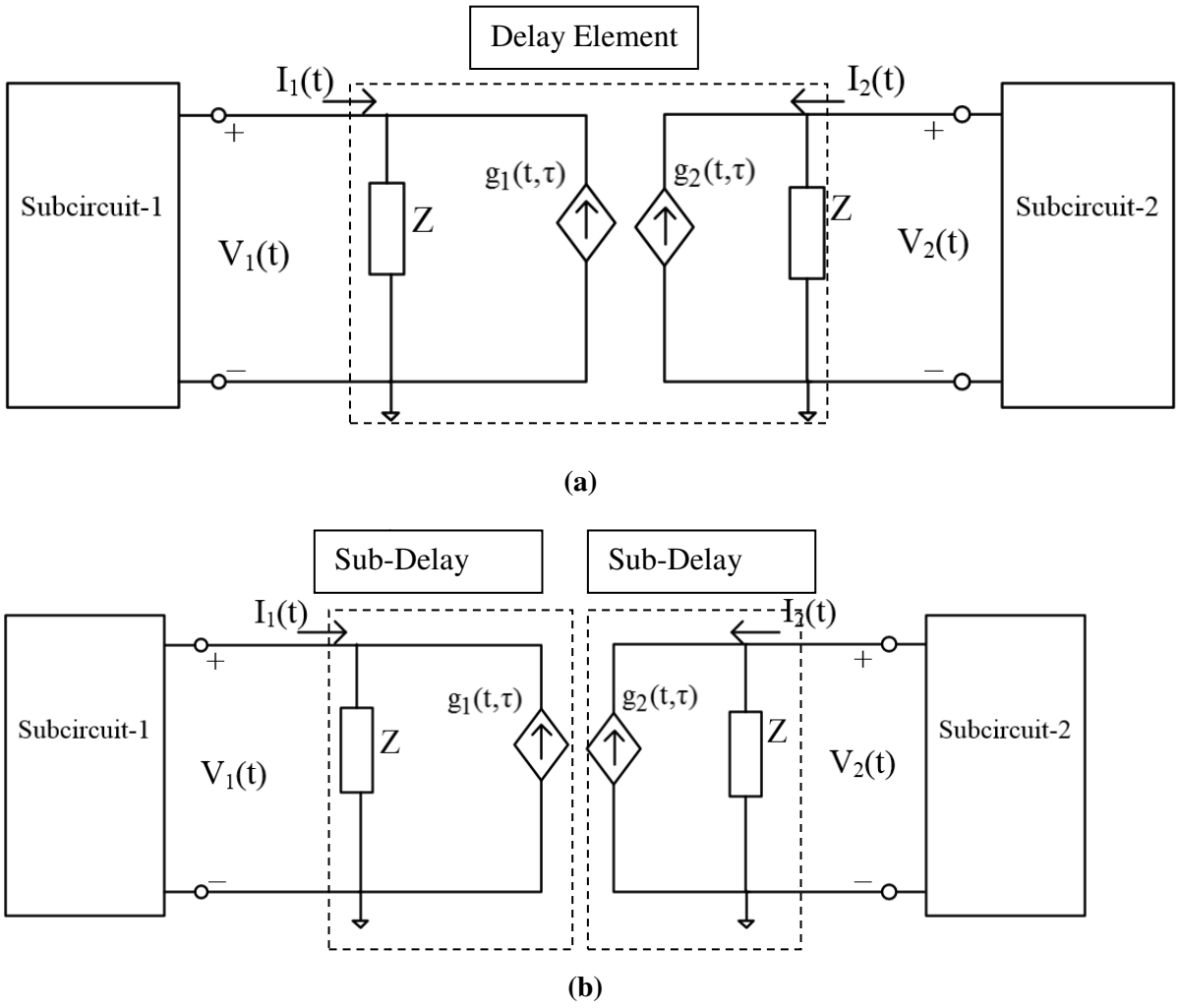


Figure 2.11 Partition of two subcircuits with delay element

To interface the NR-based iterations of each of the circuit partitions, a delay-based iterative scheme is used. In the case of finite delay, the top-level simulator iterates at the delay elements until voltages and currents at the delay element ports become consistent with the subcircuits connected to them. In the case of an instantaneous connection, *i.e.*, zero delay, equations are solved by means of waveform relaxation [30]. All subcircuits are solved independently and relaxation iterations at the delay elements match the voltages at the ports of the delay element. Figure 2.12 shows flowchart of parallel simulation of delay based partitioning [29].

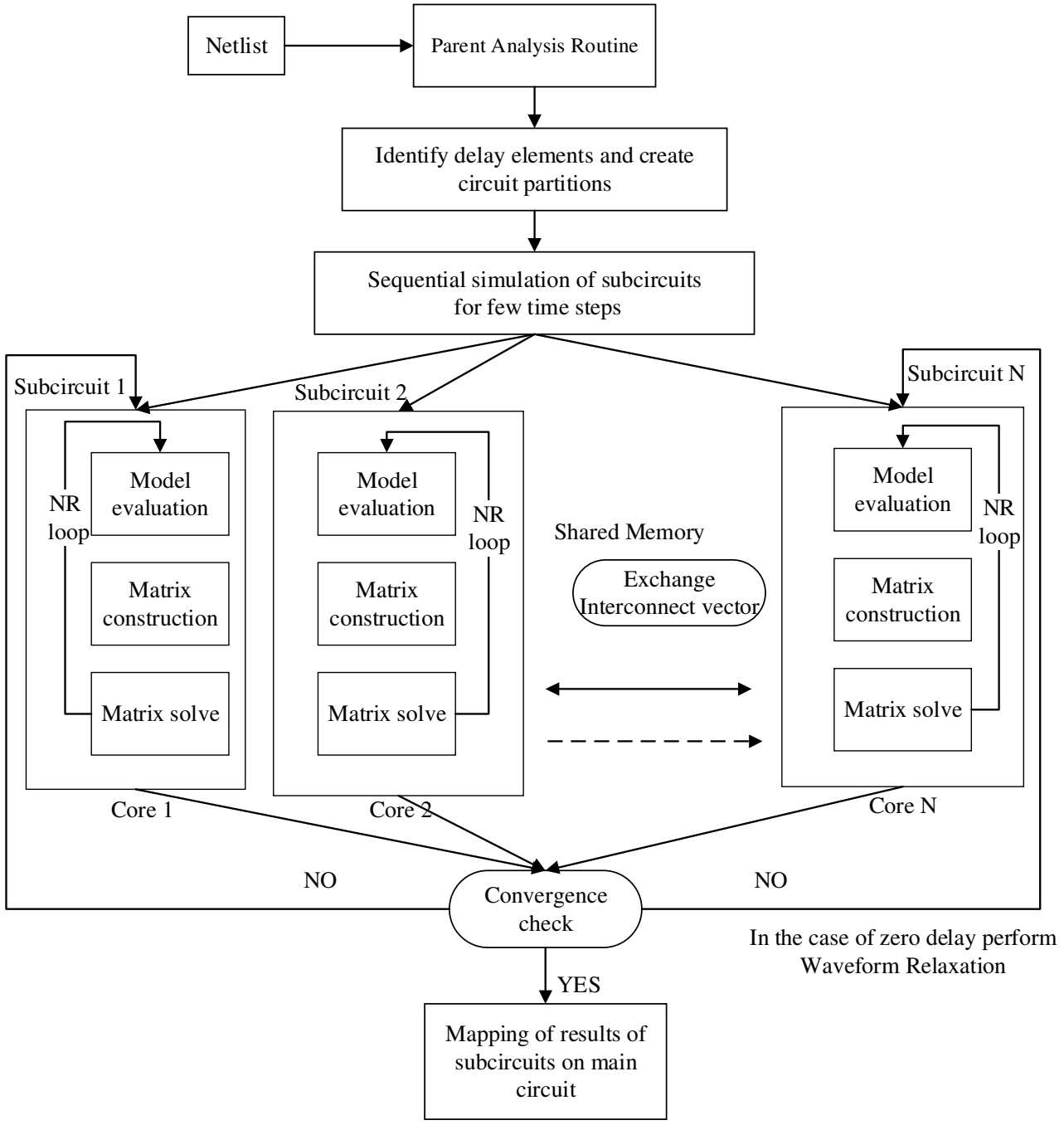


Figure 2.12 Flowchart of parallel simulation of delay based partitioning

First of all, the netlist is analyzed by the parent analysis routine. Then, the delay elements whose two ports belong to different LRGs are identified. These delay elements represent the temporal isolation between the subcircuits connected to the corresponding delay element. The delay elements are then divided into two sub-delay elements which is also called partner sub-delay

elements, as shown in Figure 2.11, resulting in two circuit partitions belonging to different LRGs. The circuit partitions thus formed are simulated sequentially in their circuit topological order initially for a few time steps (ΔT) within the parent routine. This builds history, which enables efficient automated parallel simulation of circuit partitions for the rest of the simulation time points. After initial sequential simulation, multiple child threads are allocated from the parent routine with the number of threads. The number of allocated threads depends on the number of circuit partitions and the number of available cores of the shared-memory multicore processor. Each child thread is assigned one circuit partition and directed to one of the available cores. Likewise, if N cores are available then N circuit subcircuits can be run in parallel [29]. Each circuit partition is simulated for $D_F\Delta T$ time period, where D_F is delay factor described by relationship show in Equation (2.30) and ΔT is fixed time step.

$$D_F = \frac{[min\ (\tau_1, \tau_2,.., \tau_N)]}{\Delta T} \tag{2.30}$$

The two partner sub-delay elements exchange their current and past state-variable-based current source values (called the interconnect vector) after each ΔT time duration. The interconnect vectors are stored in a shared memory data structure. The individual circuit partitions are solved using the direct method that comprises three steps:

1) Model evaluation (linearization of nonlinear device characteristics and Jacobian matrix calculation)
2) Matrix build (construction of a sparse matrix equation)
3) Matrix solve (The solution of matrix equation coupled in an NR loop)

An error criterion is formulated in the parent routine to check convergence. In the case of zero delay element perform waveform relaxation. Each subcircuit iterates for $D_F\Delta T$ time then sends results to the other subcircuits. Each subcircuit then will check whether it is consistent with previous solution. If the previous solution is not same as the current solution then iterate again till find consistent solution. At the end, subcircuit voltages and currents are mapped to the parent circuit voltages and currents after convergence is achieved.

Simulation Results

Simulation results for 8 different circuits are presented in [29] and 4 of them are listed in Table 2.5 with percentage of total simulation time taken by model evaluation, matrix build and matrix solve. Simulation results for unpartitioned chain of 12 frequency divider circuit, chain of 8 frequency multipliers, soliton line and 20-bit ripple carry adder are shown. Table 2.6 shows percentage of total simulation time reduced by partitioning circuit and simulated on different number of processors 2 and 8.

Table 2.5 Percentage of Total Simulation Time Taken by Various Steps During Simulation On a Single Core

Circuits	Model Evaluation	Matrix build	Matrix Solve
Chain of 12 frequency dividers	64	31.53	3.96
Chain of 8 frequency multiplier	32.20	56.93	10.34
Soliton Line	2.24	85.26	11.90
20 Bit Ripple Counter	54.9	40.53	4.21

Table 2.6 Percentage reduction in the various steps of simulation in delayed partitioned parallel simulation on multiple cores w.r.t. unpartitioned simulation on a single core

	Model Evaluation		Matrix Build		Matrix Solve	
Cores	**2**	**8**	**2**	**8**	**2**	**8**
Chain of 12 frequency dividers	50.81	90.10	74.83	98.6	45.93	90.83
Chain of 8 frequency multiplier	47.10	86.36	76.81	97.93	48.51	86.23
Soliton Line	46.73	87.85	73.73	98.02	44.93	85.51
20 Bit Ripple Counter	56.11	89.33	71.65	96.54	51.61	83.87

There are two main parallelization overheads in proposed method [29]. The first one is sequential simulation. In order to create history, sequential simulation has to be performed at the beginning of the simulation that enables parallel simulation at subsequent time points. The second overhead is locks and barriers implemented at each $D_F\Delta T$. After every $D_F\Delta T$ time frame the circuit is synchronised, which reinitializes the parallel simulation to a sequential simulation.

This overhead increases as the number of subcircuits increases, because now more number of subcircuits will access the shared memory data, resulting in longer waiting times due to the lock on the data structure. Efficiency of this approach depends on partition and it depends on specific type of circuits having delay elements. If there is no delay element in circuit then waveform relaxation is performed. In waveform relaxation each subcircuit has to iterate for several time for the same time interval until it converges and hence it is not efficient.

HMAPS [31]

All literature discussed till now suffers two main disadvantages. First one, all parallel approaches are intra-algorithm, *i.e.* parallel computing is only applied to expedite intermediate computational steps within a single algorithm. This choice often leads to fine grained parallel algorithm which requires a significant amount of data dependency analysis and programming efforts.

Second common disadvantage is load balancing. Circuit may not be partitioned evenly *i.e.* each partitioned subcircuit blocks may have different number of nodes and elements. For example, one circuit network is divided in three subcircuits and Subcircuit 1 is smaller than other two. These subcircuits are assigned to three different processors. Now, processor with small subcircuit will complete its calculations faster than other two processors with big subcircuit blocks. Processor with smaller subcircuit has to wait for information from other processors. Hence parallel simulation of such circuit is not efficient for all approaches discussed till now. However this depends on the circuit partition, which is not the focus of this thesis.

Circuit behaviours to be simulated are complex functions of circuit types, structures and input excitations. Furthermore, for a fixed circuit, the circuit behaviour may vary significantly over the time, exhibiting varying amount of switching activities and because of nonlinearities. It is not difficult to predict that the characteristics of circuit behavior have a definite influence on how such characteristics may be simulated by different families of simulation algorithms. However, in practice, it is difficult to select a single best algorithm that fits all type of circuits or even one complete simulation run for a given circuit.

This observation of variations in the performance of a single simulation algorithm over the time and different types of circuit suggests that it is beneficial to run multiple algorithms in parallel

[31]. There are four different computing models possible for circuit simulation: (i) single algorithm on single core processor (Figure 2.13(a)) (ii) single algorithm on multi-core CPU (Figure 2.13(b)) (iii) multialgorithm on multicore CPU and (iv) hierarchical multialgorithm on multicore CPU.

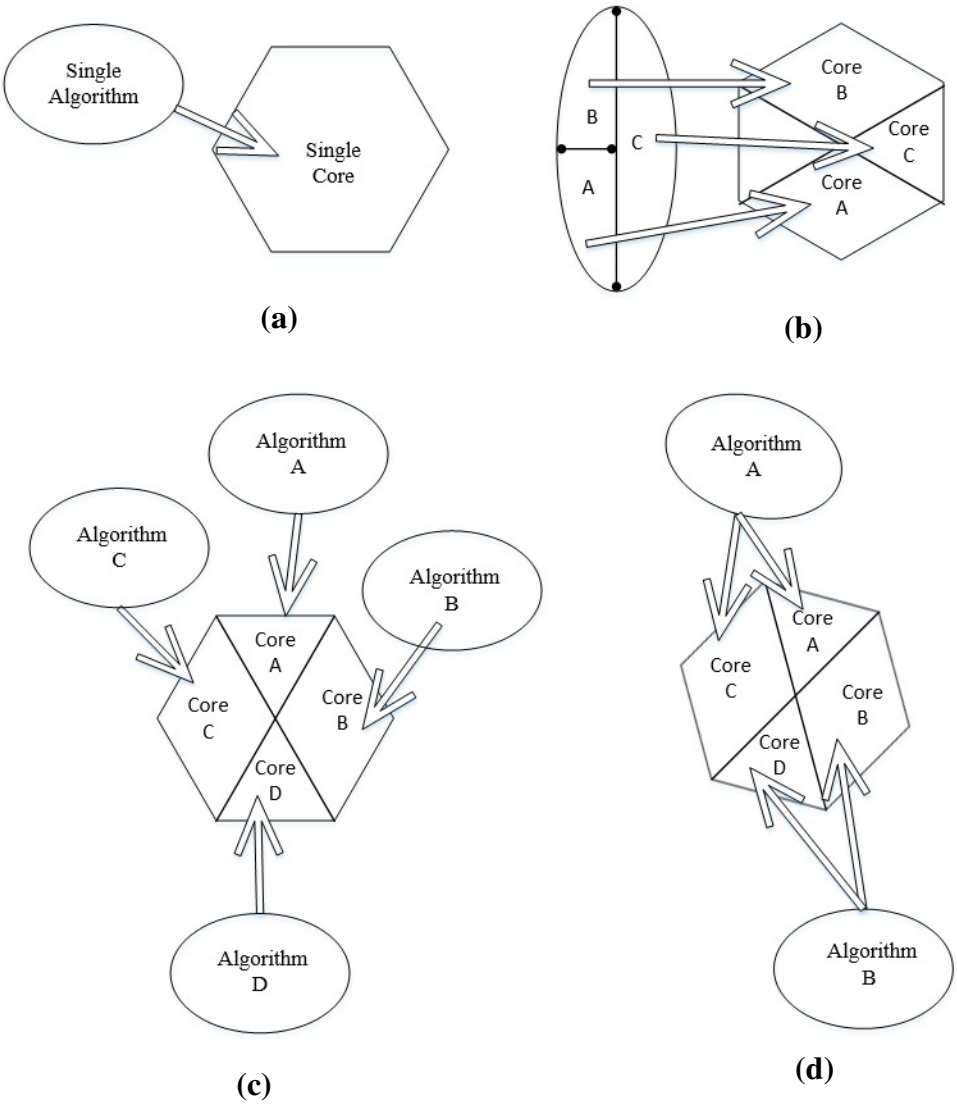


Figure 2.13 Possible computing models of circuit simulation approaches (a) Single algorithm on single core processor (b) Single algorithm on multi-core CPU (c) Multialgorithm on multicore CPU and (d) Hierarchical multialgorithm on multicore CPU

In this work, the researchers propose a hierarchical multi-algorithm (MA) parallel circuit simulation framework for parallel time-domain transistor level circuit simulation. Their

framework exploits the advantageous characteristics of the recent multi-core processor computing platforms such as small inter-processor communication cost, flexible shared memory programming environment to achieve good runtime performance.

Unlike conventional approaches where a single (parallel) algorithm is employed for a given application, in HMAPS, multiple algorithms with varying characteristics are launched to process the same simulation task. In their framework, they implemented two levels of parallelism. For a simulation task, multiple different simulation algorithms begin in parallel. Parallel speedups are obtained by having these algorithms interact with each other in a cooperative manner on the fly. This opens up a somewhat unorthodox angle to approach parallel circuit simulation as it allows one to explore a combination of intra- and interalgorithm parallelism. This combination of different levels of parallelism not only opens up new opportunities, but also explore advantages that are simply not possible when working within one fixed algorithm. Each algorithm in the multi-algorithm framework uses multiple CPU cores to do its own computing tasks. By using this hierarchical multi-algorithm parallel circuit simulation framework, super-linear speedup is achieved for some test circuits [31].

Other Contribution in Parallel Circuit Simulation

An adaptive sparse matrix solver called NICSLU is proposed by paper [33]. They proposed matrix solver called NICSLU, which uses multithread parallel LU factorization algorithm on shared memory computers with multicore CPUs to accelerate circuit simulators. A simple method is proposed to predict whether a matrix is suitable for parallel factorization.

Another reference [32] proposes a new method for transient analysis of nonlinear circuits based on power waves instead of voltages and currents. The circuit is partitioned into two parts : linear and nonlinear. This method uses relaxation approach to decouple the calculation in each part. The advantage using power waves is that iterations can never diverge to infinity. The use of waves results in guaranteed convergence for any linear passive circuit and some types of nonlinear circuits. Another advantage using power waves is, this method does not require large matrix decomposition if time step is constant. This method was implemented in the *fREEDA* [29] circuit simulator. Because of the concurrent calculations of this approach, the method can be adopted to solve in parallel.

Chapter 3

System Formulation for Parallel Circuit Analysis

This chapter discusses the proposed two circuit decomposition methods for efficient parallel analysis, one is based on nodal variables and another based on scattering waves. Both approaches have been implemented in two analysis types in a circuit simulator: EOP and WAVEOP.

The formulation for each approach is presented first, followed by details about the software implementation. The performance of both formulations is then evaluated with simulation examples.

3.1 Formulation Based on Nodal Voltages and Currents

If circuit partition is performed using branch tearing then elements shared by subcircuit block and interconnect block create dependency between each other. Branch tearing causes additional process interdependencies and thus increases simulation time. Let's consider an example circuit diagram Figure 2.5 and Equation (2.9) used in Section 2.2.4 which is shown again in Figure 3.1 and Equation (3.1) for quick reference.

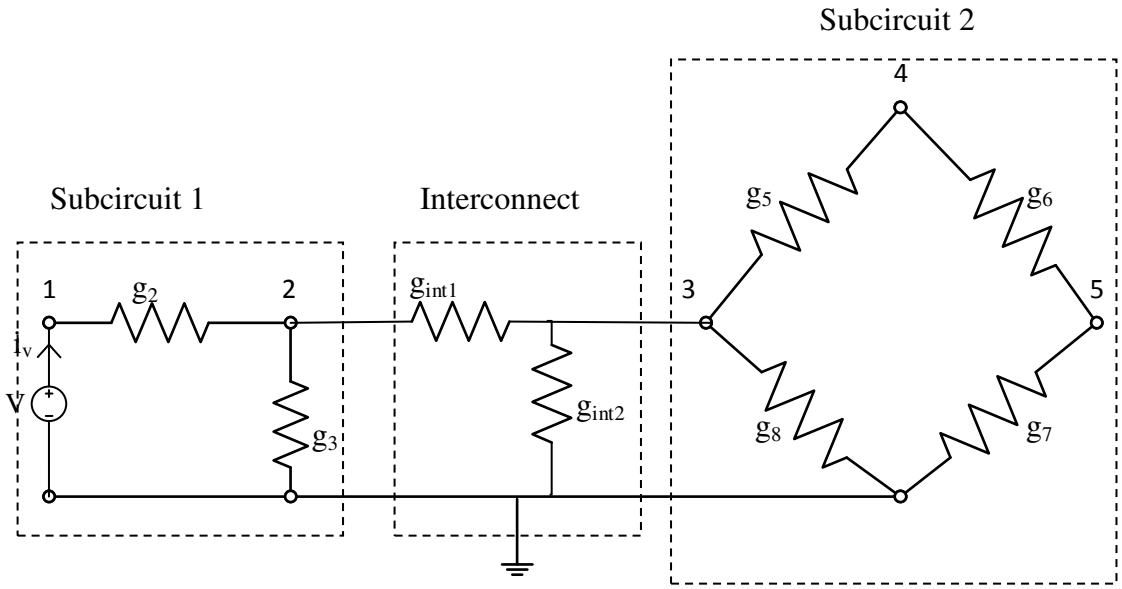


Figure 3.1 Example Circuit Network

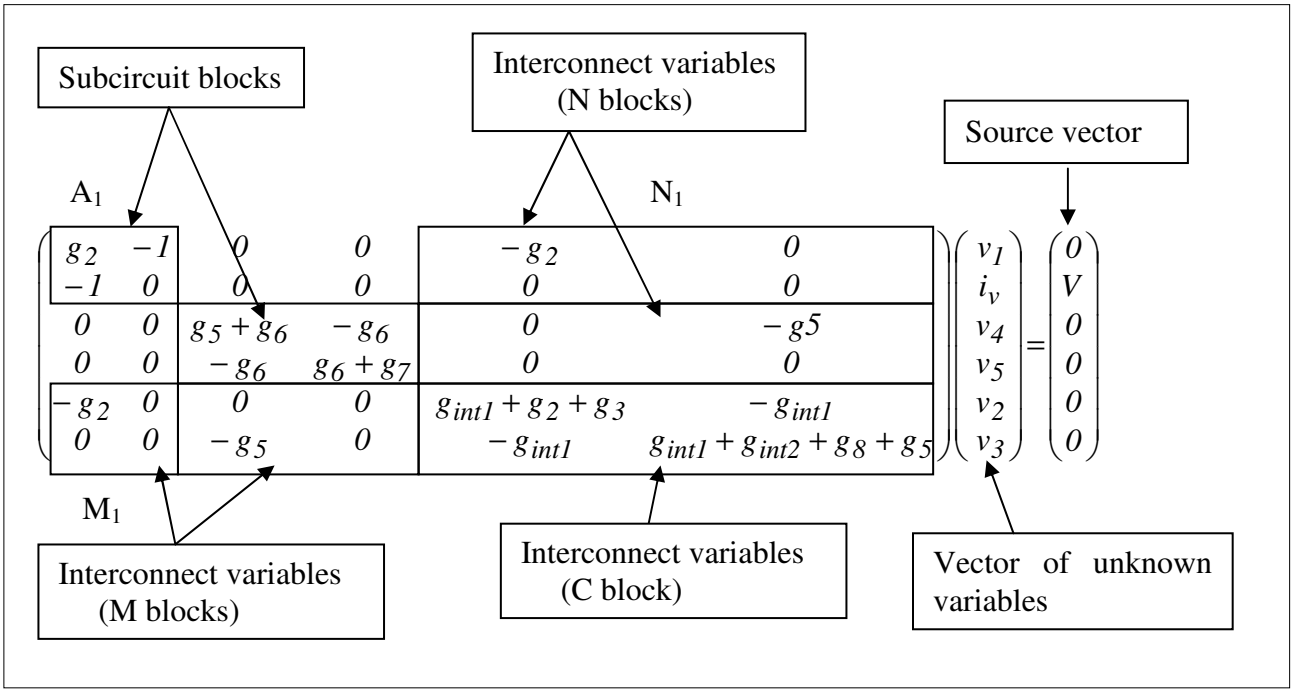


Figure 3.2 System of equation

From Figure 3.2 it is clear that conductance g_2 is shared by Blocks $\boldsymbol{A_1}$, $\boldsymbol{M_1}$, $\boldsymbol{N_1}$ and $\boldsymbol{C}$. Hence change in that element in $\boldsymbol{A_1}$ will affect all other blocks. Linear components will not change after each Newton iteration but nonlinear elements change at every Newton iteration which increases information exchange between processors sharing same nonlinear element. Furthermore, if interconnect network is large it increases amount of information exchange between processors and also increases the complexity of the interconnect system. In the proposed method, the interconnect block is successfully removed from Jacobian matrix.

The node tearing approach is described next. The following derivation assumes that:

1. Circuit network is manually partitioned in blocks.
2. Each subcircuit has a ground connection.

Figure 3.3 shows the general circuit diagram using node tearing [17]. Each interface node is separated into two nodes by means of ideal voltage source.

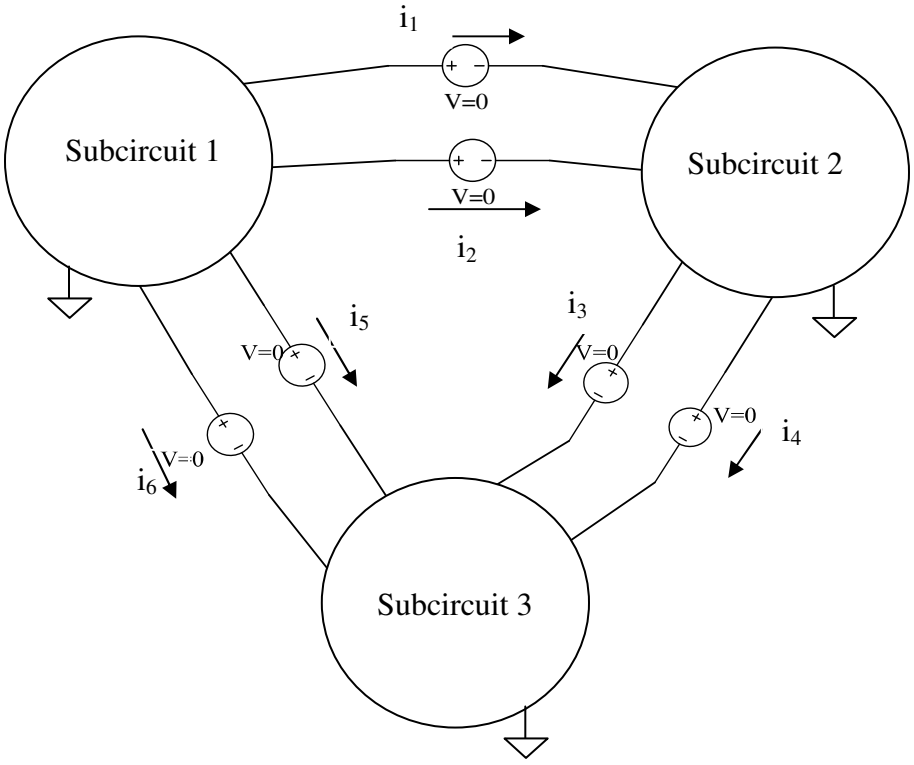


Figure 3.3 General circuit diagram to explain partition approach

Connections between subcircuits appear as single node in nodal equations. So, we split these nodes by adding ideal voltage sources having 0V. According to MNA, adding an ideal voltage

source in circuit needs an extra variable to solve nodal admittance matrix [34] and hence we added external current as an extra variable. This addition does not change the overall response of the circuit. Adding external subcircuit currents duplicates the amount of common nodal voltages between subcircuits and hence dependency between blocks is reduced.

Now from Figure 3.3 the nodal admittance matrix for circuit partitioned into three subcircuits has the form shown in Figure 3.4. A_1, A_2, and A_3 represent individual subcircuits, N_1, N_2, and N_3 are incidence matrices. Most entries of incidence matrices are '0', except that there is a '±1' in each row corresponding to an external connection (shaded part in N_j and N_j^T blocks) .

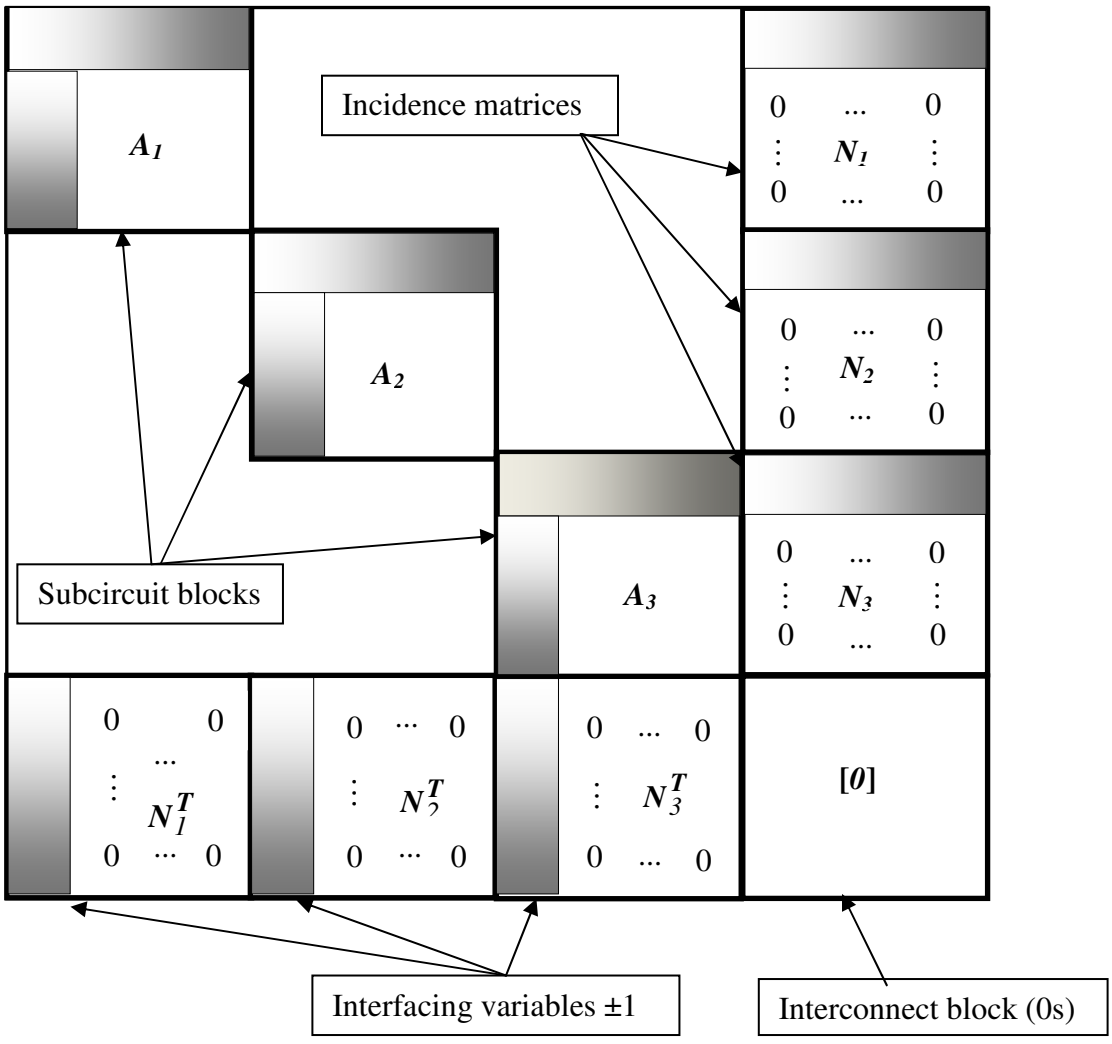


Figure 3.4 Jacobian matrix block (shaded parts are matrix entries that correspond to subcircuit interconnections)

In Figure 3.4, $\boldsymbol{N_j}$ and $\boldsymbol{N_j^T}$ blocks are constant and interconnect block $\boldsymbol{(C)}$ is zero. This not only results in less information exchange between processors but also in simpler matrix structure. Another advantage of this partitioning approach is that a pair of voltage and current at external port of each subcircuit is available. This is useful for a later objective of exchanging information between subcircuits using scattering waves and for waves both voltage and current are required.

This approach is also compatible with the connection where more than two subcircuits are connected to one node. This is handled by inserting ideal voltage sources having 0 volts to the external ports of any *k-1* subcircuits if there are *k* subcircuits connected to one node. Figure 3.5 shows such an example where three subcircuits are sharing one node *a*. In this case two ideal voltage sources and two external currents are assigned to Subcircuit 2 and Subcircuit 3.

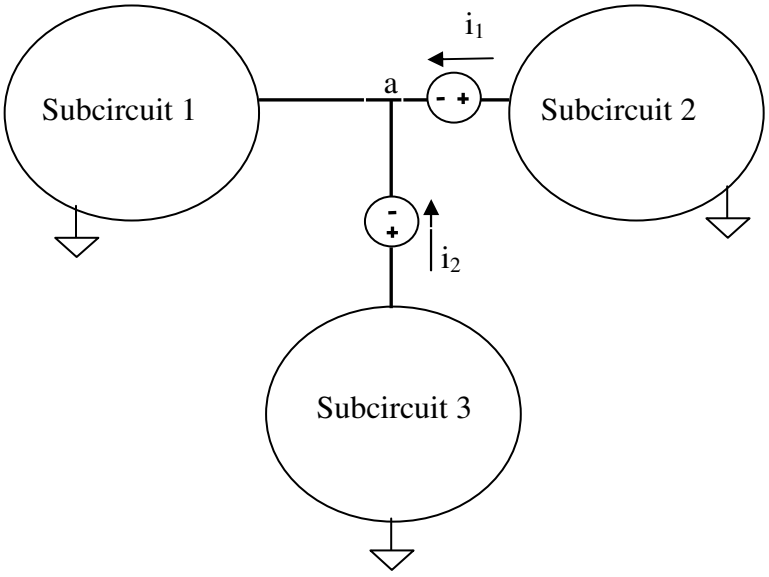


Figure 3.5 General circuit diagram of three subcircuits sharing same node *a*

Node *a* would appear in each of three subcircuit blocks $\boldsymbol{A_1}$, $\boldsymbol{A_2}$, $\boldsymbol{A_3}$ (Figure 3.4) and $\boldsymbol{N_j}$ blocks would have two columns as shown in Figure 3.6.

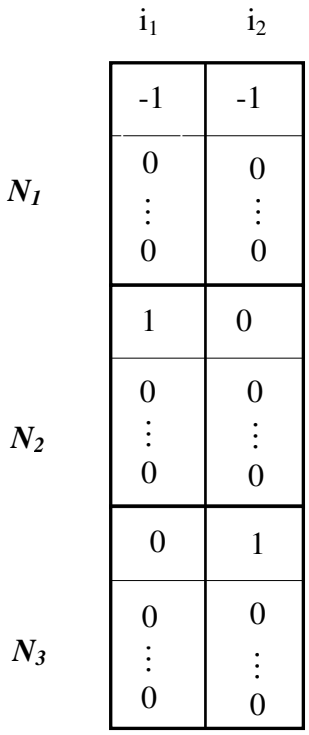


Figure 3.6 External currents arrangement in N blocks

3.1.1 Diakoptics Applied to Node Tearing

Assume that a circuit is partitioned into *k* subcircuits separated by zero-volt ideal voltage sources as shown in Figure 3.7.

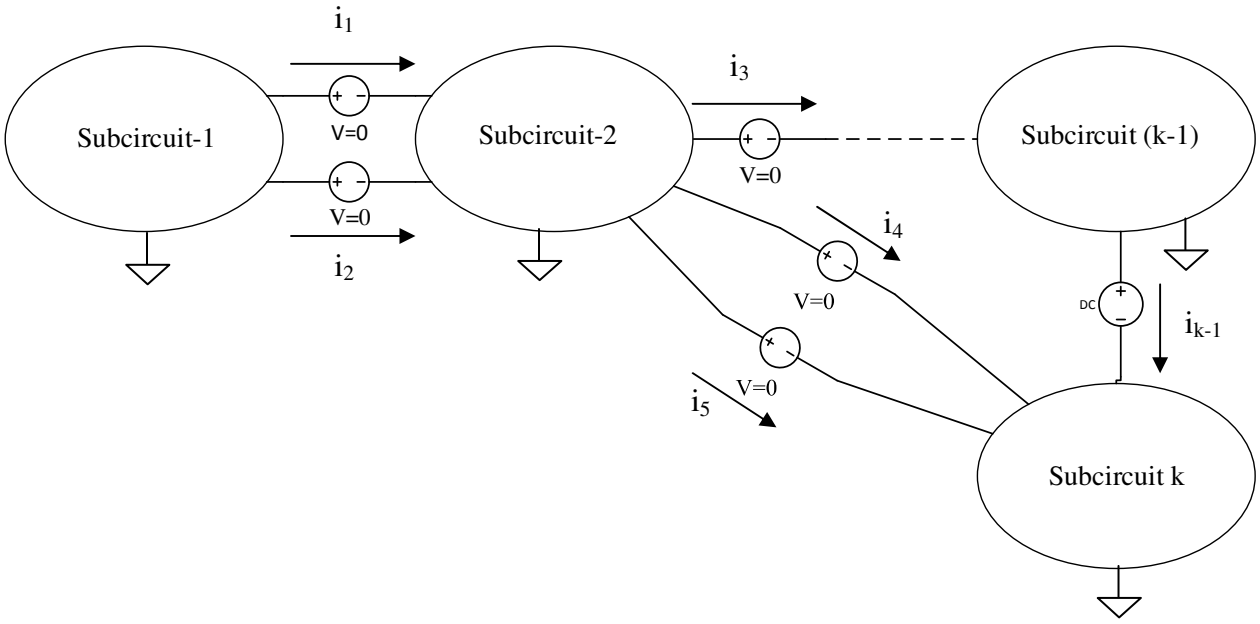


Figure 3.7 General circuit diagram of partitioned circuit for derivation

From Figure 3.7 a generalized system of equation can be written as:

$$\begin{pmatrix} \boldsymbol{G}_1 & & & & \boldsymbol{N}_1 \\ & \boldsymbol{G}_2 & & & \boldsymbol{N}_2 \\ & & \ddots & & \vdots \\ & & & \boldsymbol{G}_k & \boldsymbol{N}_k \\ \boldsymbol{N}_1^{\boldsymbol{T}} & \boldsymbol{N}_2^{\boldsymbol{T}} & \cdots & \boldsymbol{N}_k^{\boldsymbol{T}} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_k \\ \boldsymbol{i}_{\boldsymbol{I}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{i}_1(\boldsymbol{x}_1) \\ \boldsymbol{i}_2(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{i}_k(\boldsymbol{x}_k) \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \vdots \\ \boldsymbol{s}_k \\ \boldsymbol{0} \end{pmatrix} \tag{3.1}$$

where $\boldsymbol{G}_j$ is subcircuit block, $\boldsymbol{N}_j$ is incidence matrix block, $\boldsymbol{i}_1(\boldsymbol{x}_1), \boldsymbol{i}_2(\boldsymbol{x}_2), \ldots, \boldsymbol{i}_k(\boldsymbol{x}_k)$ are internal currents of nonlinear components inside subcircuits, $\boldsymbol{i}_{\boldsymbol{I}}$ is external current vector and $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k$ are subcircuit source vectors. Notice that there is neither interconnect matrix block nor interconnect source vector in Jacobian matrix written in Equation (3.1).

Now applying Newton's method on Equation (3.1) and using $\boldsymbol{F(x)=G\,x+i(x)}$ and hence $\boldsymbol{J_F=G+J_i}$, the following is obtained:

$$\underbrace{\begin{pmatrix} \boldsymbol{G_1+J_1} & & & & \boldsymbol{N_1} \\ & \boldsymbol{G_2+J_2} & & & \boldsymbol{N_2} \\ & & \ddots & & \vdots \\ & & & \boldsymbol{G_k+J_k} & \boldsymbol{N_k} \\ \boldsymbol{N_1^T} & \boldsymbol{N_2^T} & \cdots & \boldsymbol{N_k^T} & \boldsymbol{0} \end{pmatrix}}_{\boldsymbol{G+J_i}} \begin{pmatrix} \boldsymbol{\Delta x_1^{n+1}} \\ \boldsymbol{\Delta x_2^{n+1}} \\ \vdots \\ \boldsymbol{\Delta x_k^{n+1}} \\ \boldsymbol{\Delta i_I^{n+1}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{s_1} \\ \boldsymbol{s_2} \\ \vdots \\ \boldsymbol{s_k} \\ \boldsymbol{0} \end{pmatrix} - \underbrace{\begin{pmatrix} \boldsymbol{G_1 x_1^n + N_1 i_I^n + i_1(x_1^n)} \\ \boldsymbol{G_2 x_2^n + N_2 i_I^n + i_2(x_2^n)} \\ \vdots \\ \boldsymbol{G_k x_k^n + N_k i_I^n + i_k(x_k^n)} \\ \boldsymbol{\sum_{j=1}^{k} N_j^T x_j^n} \end{pmatrix}}_{\boldsymbol{-(G\,x+i(x))}} \tag{3.2}$$

where $\boldsymbol{J_i}$ is Jacobian matrix of current vector $\boldsymbol{i(x)}$, $\boldsymbol{n}$ is iteration index and $\boldsymbol{\Delta i_I^{n+1}}$ is external (interconnect) current vector .

$$\text{For each subcircuit let } \begin{cases} \boldsymbol{f_k(x_k^n)=G_k x_k^n + i_k(x_k^n)} \\ \boldsymbol{A_k=G_k+J_k} \ . \end{cases} \tag{3.3}$$

Using relations shown in Equations (3.3) in an Equation (3.2) results in:

$$\begin{pmatrix} \boldsymbol{A_1} & & & & \boldsymbol{N_1} \\ & \boldsymbol{A_2} & & & \boldsymbol{N_2} \\ & & \ddots & & \vdots \\ & & & \boldsymbol{A_k} & \boldsymbol{N_k} \\ \boldsymbol{N_1^T} & \boldsymbol{N_2^T} & \cdots & \boldsymbol{N_k^T} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta x_1^{n+1}} \\ \boldsymbol{\Delta x_2^{n+1}} \\ \vdots \\ \boldsymbol{\Delta x_k^{n+1}} \\ \boldsymbol{\Delta i_I^{n+1}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{s_1} \\ \boldsymbol{s_2} \\ \vdots \\ \boldsymbol{s_k} \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{f_1(x_1^n)} \\ \boldsymbol{f_2(x_2^n)} \\ \vdots \\ \boldsymbol{f_k(x_k^n)} \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{N_1 i_I^n} \\ \boldsymbol{N_2 i_I^n} \\ \vdots \\ \boldsymbol{N_k i_I^n} \\ \boldsymbol{\sum_{j=1}^{k} N_j^T x_j^n} \end{pmatrix} . \tag{3.4}$$

Now consider the system of equation for j^{th} subcircuit. First

$$\boldsymbol{\Delta x_j^* = A_j^{-1}(s_j - f_j(x_j^n) - N_j\, i_I^n)} \tag{3.5}$$

is found. where, *j=1, 2, ..., k.* Unknown nodal voltages of subcircuit *j* can be found using Equation (3.5) from:

$$\Delta \boldsymbol{x}_j^{n+1} = \Delta \boldsymbol{x}_j^{*} - \boldsymbol{A}_j^{-1} \boldsymbol{N}_j \Delta \boldsymbol{i}_I^{n+1} \tag{3.6}$$

where, $\Delta \boldsymbol{x}_j^{n+1}$ is $\boldsymbol{j}^{th}$ subcircuit nodal voltage. The equation related to interconnect blocks is :

$$\sum_{j=1}^{k} \boldsymbol{N}_j^T \Delta \boldsymbol{x}_j^{n+1} = -\sum_{j=1}^{k} \boldsymbol{N}_j^T \boldsymbol{x}_j^n \tag{3.7}$$

Replace $\Delta \boldsymbol{x}_j^{n+1}$ from Equation (3.6) into Equation (3.7) and rearrange equation to get :

$$\sum_{j=1}^{k} \boldsymbol{N}_j^T \Delta \boldsymbol{x}_j^{*} - \sum_{j=1}^{k} (\boldsymbol{N}_j^T \boldsymbol{A}_j^{-1} \boldsymbol{N}_j) \Delta \boldsymbol{i}_I^{n+1} = -\sum_{j=1}^{k} \boldsymbol{N}_j^T \boldsymbol{x}_j^n$$

$$\sum_{j=1}^{k} \left(\boldsymbol{N}_j^T \boldsymbol{A}_j^{-1} \boldsymbol{N}_j \right) \Delta \boldsymbol{i}_I^{n+1} = \sum_{j=1}^{k} \boldsymbol{N}_j^T \Delta \boldsymbol{x}_j^{*} + \sum_{j=1}^{k} \boldsymbol{N}_j^T \boldsymbol{x}_j^n$$

$$\left(\sum_{j=1}^{k} \boldsymbol{N}_j^T \boldsymbol{A}_j^{-1} \boldsymbol{N}_j \right) \Delta \boldsymbol{i}_I^{n+1} = \sum_{j=1}^{k} \boldsymbol{N}_j^T (\boldsymbol{x}_j^n + \Delta \boldsymbol{x}_j^{*}) \tag{3.8}$$

The practical implementation of the above process has been developed to solve system of Equation (3.1). First Equation (3.5) is solved, followed by interconnect current vector which can be obtained from Equation (3.8) and then subcircuit nodal voltages can be found from Equation (3.6) using Equations (3.5) and (3.8).

3.1.2 Algorithm Flowchart

The flowchart of the analysis based on nodal voltages and currents is shown in Figure 3.8.

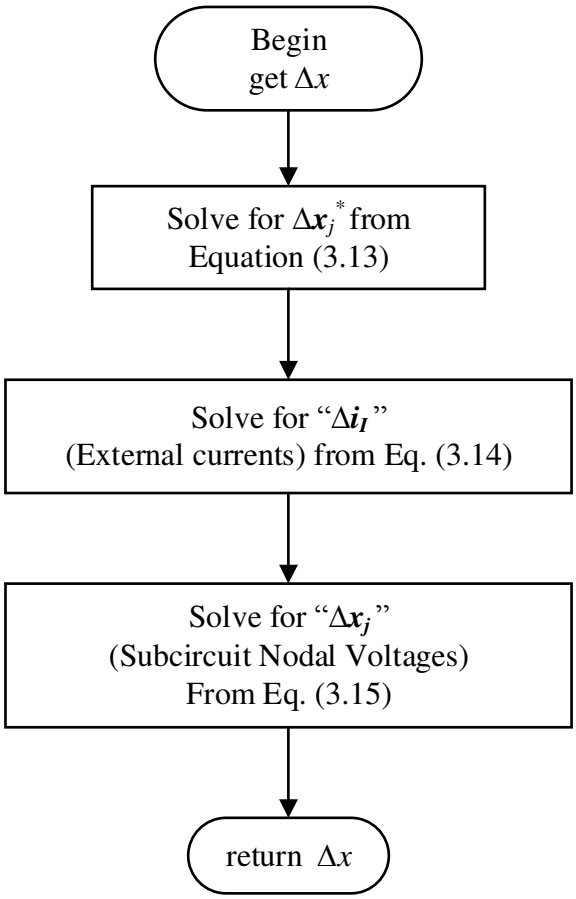


Figure 3.8 Algorithm of analysis based on nodal variables

A flow diagram of Newton method is shown in Figure 3.9. It uses ***Δx*** variable from analysis based on nodal variables and perform Newton method to check convergence.

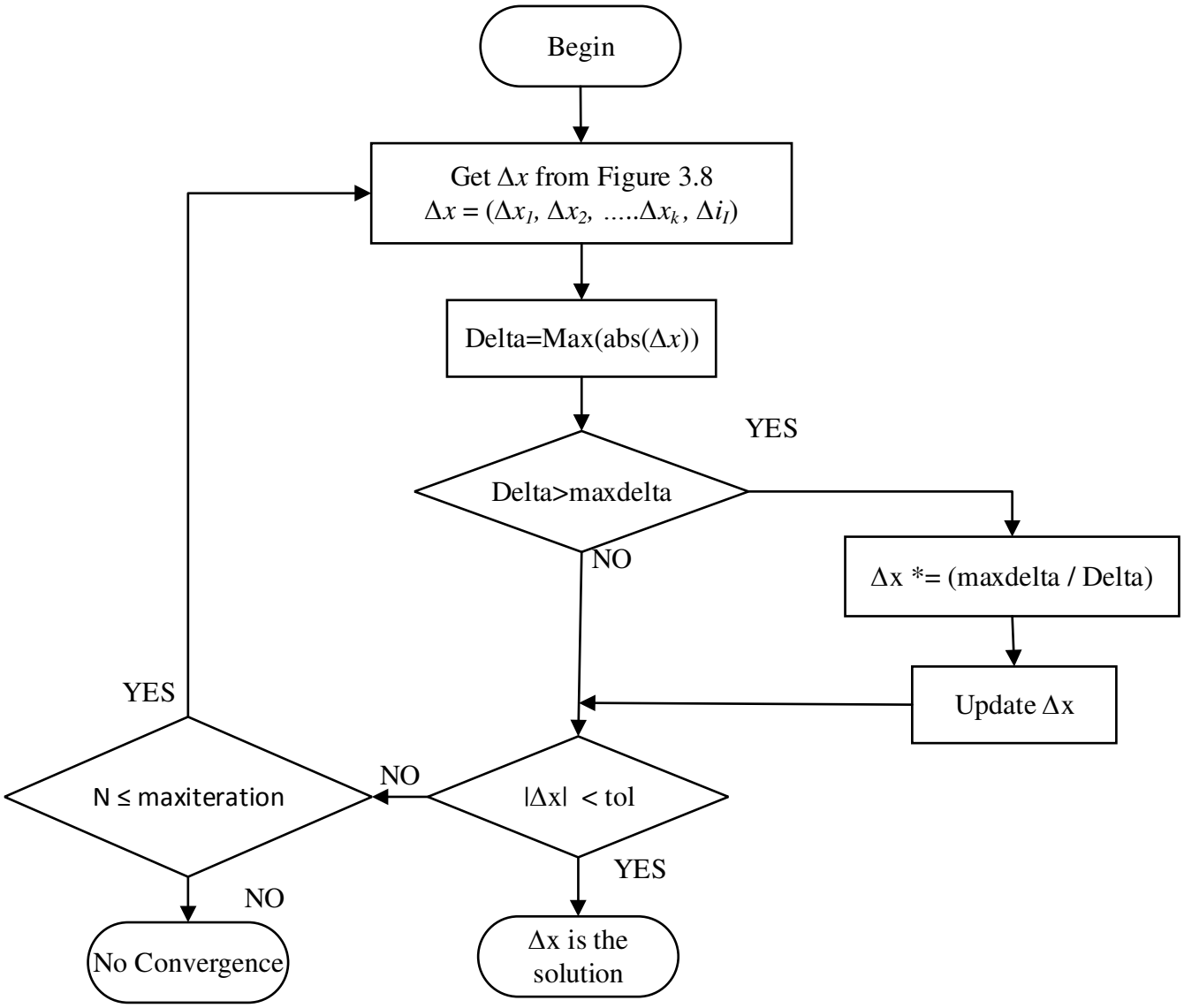


Figure 3.9 Algorithm flowchart of Newton method

Inter-processor Communication Analysis

Since the parallel algorithm is not implemented in this thesis, as a guideline, possible way to implement parallel algorithm is discussed briefly. Parallel algorithm for the proposed approach is the same as shown in Figure 2.8. The key difference between algorithm shown in Figure 2.8 and analysis with nodal variables is that, in proposed approach interconnect block ($\boldsymbol{C}$) and interconnect source vector ($\boldsymbol{b}_{k+1}$) are zero. In the case of algorithm presented in Figure 2.8, a nonlinear element connected to an external node in Subcircuit *j*, will produce entries in $\boldsymbol{N}_j$, $\boldsymbol{M}_j$ and $\boldsymbol{C}$ blocks. If this element in Subcircuit *j* changes then slave processor has to communicate this change with master processor to update $\boldsymbol{C}$ block, as interconnect nodal voltage vector is solved by master processor (Equation 2.15). But in proposed analysis, $\boldsymbol{N}_j$ blocks are constant and $\boldsymbol{C}$ block is zero. So there is less communication between master and slave processors compared to algorithm shown in Figure 2.8. And also because of the simpler structure of $\boldsymbol{N}_j$ blocks, there is less work for each slave processor to perform. In proposed analysis, $\boldsymbol{M}_j = \boldsymbol{N}_j^T$ and $\Delta \boldsymbol{x}_{k+1}^{n+1} = \Delta \boldsymbol{i}_{\boldsymbol{I}}^{n+1}$. Slave processors calculate $\boldsymbol{x}_j^*$, $\boldsymbol{S}_j^n$, $\boldsymbol{w}_j^n$ and send that information to master processor which calculates $\boldsymbol{S}^n$, $\boldsymbol{w}^n$ and $\Delta \boldsymbol{i}_{\boldsymbol{I}}^{n+1}$ (Equation 3.8). where,

$$\boldsymbol{S}^n = -\sum_{j=1}^{k} \boldsymbol{S}_j^n \text{ and } \boldsymbol{w}^n = -\sum_{j=1}^{k} \boldsymbol{w}_j^n .$$

And at last, slave processors retrieve $\boldsymbol{\Delta i}_{\boldsymbol{I}}^{n+1}$ from master processor and calculate $\boldsymbol{\Delta x}_j^{n+1}$ (Equation 3.6).

3.1.3 Complete Example

A linear circuit example is discussed here with the whole process from partitioning circuit to writing a netlist file for simulation. Figure 3.10 shows linear circuit with conductances, one ideal voltage source and 8 nodes. Subcircuits 1 and 2 have 3 nodes each, whereas Subcircuit 3 has 4 nodes. This circuit is partitioned into three subcircuits separated using ideal voltage sources with 0 volts as shown in Figure 3.10. i_1 and i_2 are the external currents.

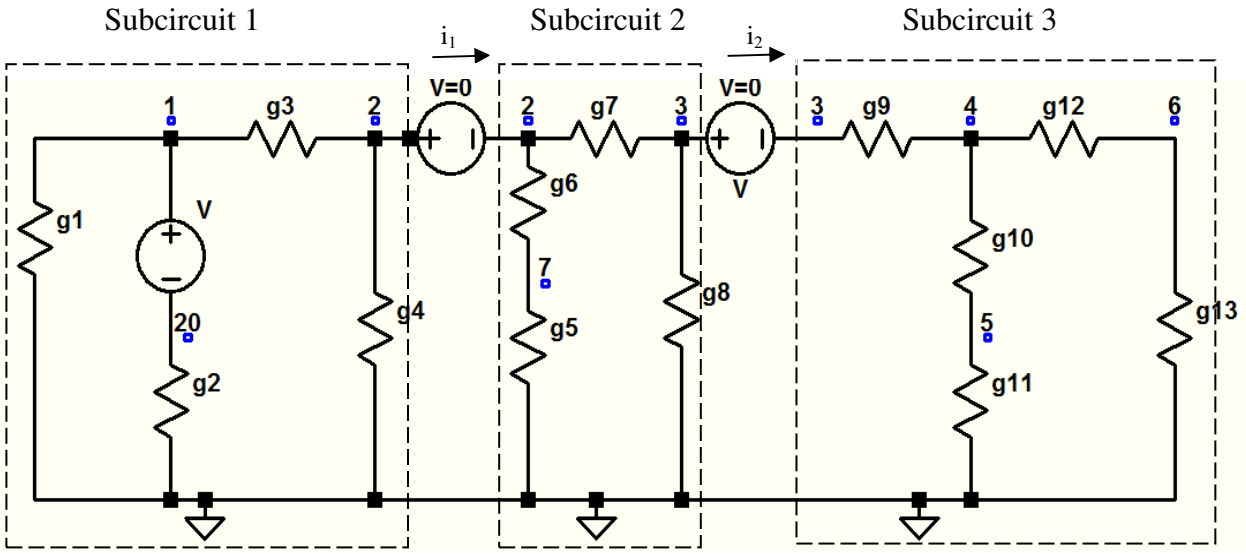
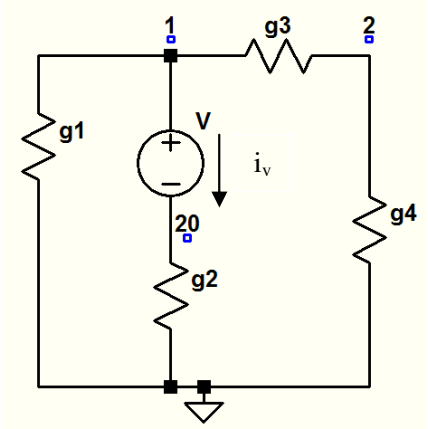


Figure 3.10 Partitioned linear circuit with nodal variables (Numbers shown above square dots are node numbers)

To simulate this circuit, it should be described in netlist format. Full netlist file is given in Appendix A. For reference Jacobian matrix block of Subcircuit 1 of Figure 3.10 is shown in Figure 3.11. Figure 3.11(b) shows Jacobian matrix of Subcircuit 1 shown in Figure 3.11 (a) [34].



(a)

$$\begin{array}{cccc} 2 & 1 & 20 & i_v \\ \hline \left|\begin{array}{cccc} g3+g4 & -g3 & 0 & 0 \\ -g3 & g1+g3 & 0 & 1 \\ 0 & 0 & g2 & -1 \\ 0 & 1 & -1 & 0 \end{array}\right| & & & \end{array} \qquad \left|\begin{array}{cc} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{array}\right| \qquad \left|\begin{array}{c} 0 \\ 0 \\ 1 \end{array}\right|$$

Subcircuit block A_1 | External currents N_1 | Source vector S_1

(b)

Figure 3.11 Example (a) Subcircuit block (b) Nodal matrix blocks

Now the Jacobian matrix for the whole circuit (Figure 3.11) with system of equation can be written as shown in Figure 3.12.

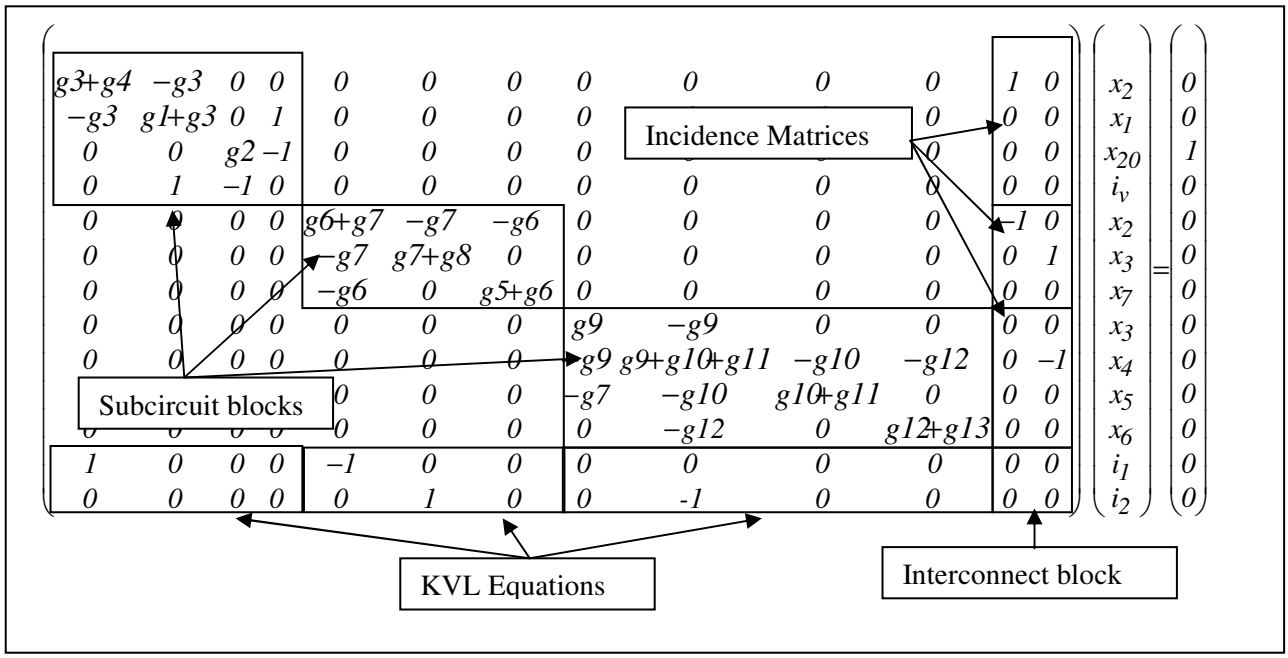


Figure 3.12 System of equations

There are two important main differences between equations shown in Figure 3.2 and Figure 3.12: (i) Interconnect block is zero in Figure 3.12 and (ii) Incidence matrices are independent of subcircuit components . Hence proposed partitioning approach reduces dependencies between subcircuit blocks.

3.2 Formulation Based on Scattering Waves

Basic Concepts

This section presents an original formulation for parallel circuit simulation using a combination of voltages, currents and scattering waves. Same partitioning approach as analysis with nodal variables is used in this implementation with zero volt voltage source and external current. Scattering waves are defined in transmission line theory. The voltage and current variables in external port are replaced by incident and reflected voltage waves. Figure 3.13 shows the inter connection of two subcircuit ports.

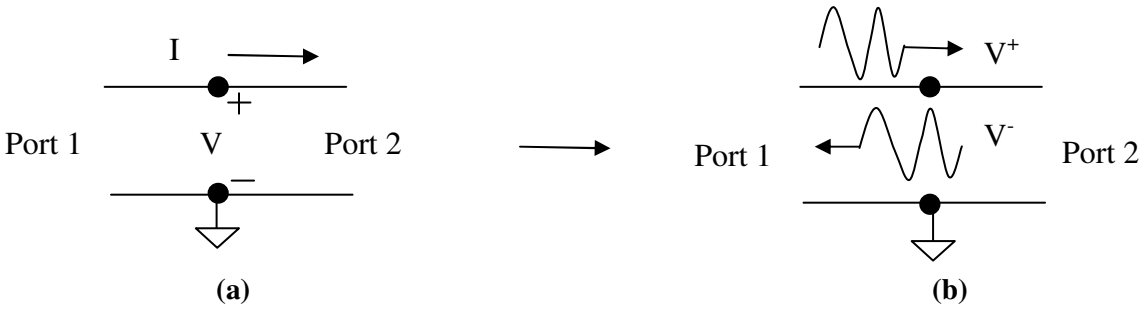


Figure 3.13 Wave transformation from voltage and current variables (a) voltage and current at external port (b) voltage waves at external port

As shown in Figure 3.13(b), for Port 1 V^+ is the reflected wave and V^- is incident wave and for Port 2 V^+ is incident wave and V^- is reflected wave. If external current (I) direction is same as reflected wave V^+ from Port 1 as shown in Figure 3.13(b) then voltage and current at external port one is defined as :

$$V = V^+ + V^- \tag{3.9}$$

$$I = {}^{1}\!/_{Z_0}(V^+ - V^-) \tag{3.10}$$

where Z_0 is the reference impedance.

This transformation has some advantages with respect to voltage-current pairs if relaxation is used to exchange results between subcircuits [32]. For relaxation, waves guaranteed convergence for any linear passive circuits and some nonlinear circuits. A formulation based on wave variables is attractive because they can handle open- and short-circuit conditions without the numerical problems that may arise when using voltage and currents. For example, if a non-zero voltage is assumed across a short-circuit, the corresponding current is infinite. The use of waves also enables the use of a simpler convergent relaxation approach to exchange information between subcircuits. One of the objectives of this thesis was to investigate Newton's method convergence properties using waves.

Now to understand role of reference impedance (Z_0), add and subtract Equations (3.9) and (3.10) to obtain:

$$V + Z_0 I = 2V^+ \Rightarrow V^+ = \frac{V + Z_0 I}{2} \tag{3.11}$$

$$V - Z_0 I = 2V^- \Rightarrow V^- = \frac{V - Z_0 I}{2}. \tag{3.12}$$

Suppose that current flowing in one subcircuit is very small compared to the voltage of that subcircuit, then from Equations (3.11) and (3.12), Z_0 should be a large number otherwise, numerical problem arises by adding small number to a large value. Here reference impedance keep $Z_0 I$ product in order of voltage. Hence, reference impedance should be in order of 100 Ω to 1 kΩ, because currents are usually in order of milliamperes.

A limitation of the current implementation based on waves is that sharing a same node by more than two subcircuits is not supported. Figure 3.14 shows general circuit diagram with three subcircuit sharing same node *a*. With this type of connection, formation of Jacobian becomes more complex and consequently it makes the code harder to implement.

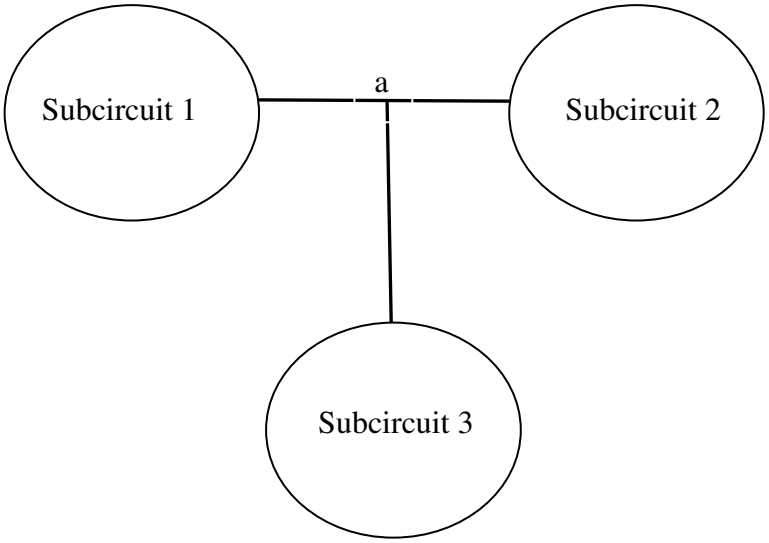


Figure 3.14 General circuit diagram for limitation of analysis based on waves

3.2.1 Formulation Details

In the decomposition discussed in Section 3.1, there is a pair of voltage and current at external port of each subcircuit interconnection, such voltage and current pair is replaced by a pair of incident and reflected waves. Each subcircuit exchanges incident and reflected waves with neighbour subcircuits. Assumptions for analysis based on waves are same as mentioned in Section 3.1 for analysis based on nodal variables, with the additional condition that only two subcircuits can share an external node.

Consider the circuit in Figure 3.15 partitioned into k subcircuits. Each pair of voltage and current of external node in subcircuit is divided into two variables: incident and reflected waves. Each node connecting to another subcircuit is combined with reference node form one port of particular subcircuit as shown in Figure 3.15 and waves are defined for such a port.

General Jacobian matrix block for the circuit partitioned into k subcircuits is same as shown in Figure 3.4 except structure of $\boldsymbol{N}_j$ block.

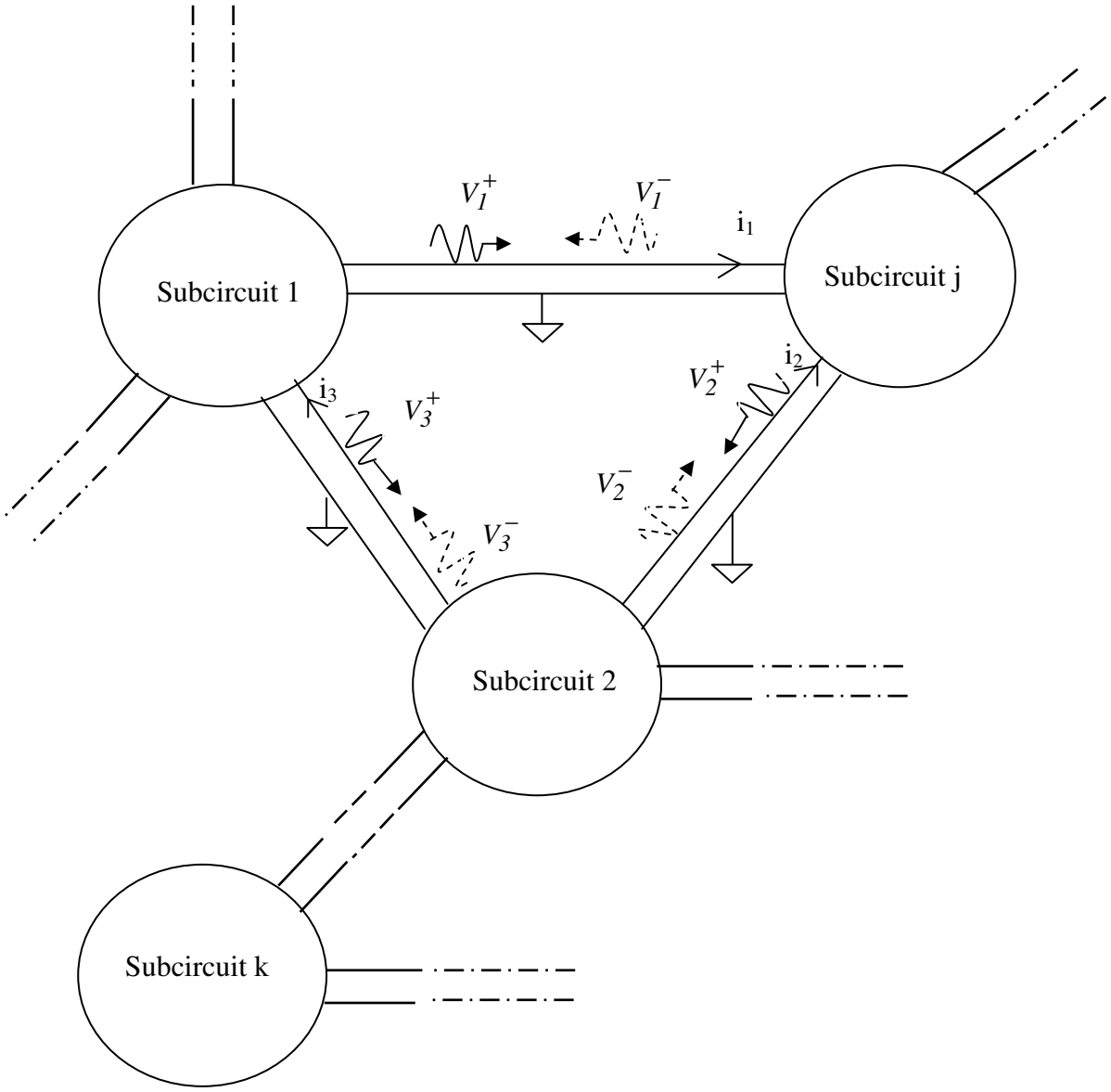


Figure 3.15 General circuit diagram of partitioned circuit with waves

Consider Equation (3.4) for k subcircuits to be used for analysis with waves. Equation (3.4) is rewritten as Equation (3.13) for quick reference:

$$\begin{pmatrix} \boldsymbol{A}_1 & & & & \boldsymbol{N}_1 \\ & \boldsymbol{A}_2 & & & \boldsymbol{N}_2 \\ & & \ddots & & \vdots \\ & & & \boldsymbol{A}_k & \boldsymbol{N}_k \\ \boldsymbol{N}_1^T & \boldsymbol{N}_2^T & \cdots & \boldsymbol{N}_k^T & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta x}_1^{n+1} \\ \boldsymbol{\Delta x}_2^{n+1} \\ \vdots \\ \boldsymbol{\Delta x}_k^{n+1} \\ \boldsymbol{\Delta i}_I^{n+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \vdots \\ \boldsymbol{s}_k \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{f}_1(\boldsymbol{x}_1^n) \\ \boldsymbol{f}_2(\boldsymbol{x}_2^n) \\ \vdots \\ \boldsymbol{f}_k(\boldsymbol{x}_k^n) \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{N}_1 \boldsymbol{i}_I^n \\ \boldsymbol{N}_2 \boldsymbol{i}_I^n \\ \vdots \\ \boldsymbol{N}_k \boldsymbol{i}_I^n \\ \sum_{j=1}^{k} \boldsymbol{N}_j^T \boldsymbol{x}_j^n \end{pmatrix} . \tag{3.13}$$

Let's take one subcircuit $\boldsymbol{A}_j = \begin{pmatrix} (\boldsymbol{A_{EE}})_j & (\boldsymbol{A_{IE}})_j \\ (\boldsymbol{A_{EI}})_j & (\boldsymbol{A_{II}})_j \end{pmatrix}$, $\boldsymbol{\Delta x}_j = \begin{pmatrix} (\boldsymbol{\Delta v_E})_j \\ (\boldsymbol{\Delta x_i})_j \end{pmatrix}$ and

$$\boldsymbol{s}_j - \boldsymbol{f}_j(\boldsymbol{x}_j^n) - \boldsymbol{N}_j \boldsymbol{i}_I^n = \boldsymbol{b}_j = \begin{pmatrix} (\boldsymbol{b_E})_j \\ (\boldsymbol{b_i})_j \end{pmatrix}$$

where $\boldsymbol{A}_j$ is the matrix block of subcircuit j, $(\boldsymbol{A_{EE}})_j$ is the corresponding sub-matrix for external nodes, $(\boldsymbol{A_{EI}})_j$ and $(\boldsymbol{A_{IE}})_j$ are sub-matrices corresponding to internal and external nodes, $(\boldsymbol{A_{II}})_j$ is sub-matrix corresponding to internal nodes, $(\boldsymbol{\Delta v_E})_j$ is external nodal voltages of subcircuit, $(\boldsymbol{\Delta x_i})_j$ is the internal nodal voltages of subcircuit, $(\boldsymbol{b_E})_j$ and $(\boldsymbol{b_i})_j$ are external and internal variable vectors, respectively. Now writing system of equation for subcircuit j from Equation (3.13) with variable transformation shown above gives:

$$\begin{pmatrix} (\boldsymbol{A_{EE}})_j & (\boldsymbol{A_{IE}})_j \\ (\boldsymbol{A_{EI}})_j & (\boldsymbol{A_{II}})_j \end{pmatrix} \begin{pmatrix} (\boldsymbol{\Delta v_E})_j \\ (\boldsymbol{\Delta x_i})_j \end{pmatrix} + \boldsymbol{N}_j \begin{pmatrix} \boldsymbol{\Delta i_I} \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} (\boldsymbol{b_E})_j \\ (\boldsymbol{b_i})_j \end{pmatrix} . \tag{3.14}$$

Using relation described in Equations (3.9) and (3.10) in Equation (3.14) to replace external voltages and currents for subcircuit j will get:

$$\begin{pmatrix} (\boldsymbol{A_{EE}})_j & (\boldsymbol{A_{IE}})_j \\ (\boldsymbol{A_{EI}})_j & (\boldsymbol{A_{II}})_j \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta v}_j^+ + \boldsymbol{\Delta v}_j^- \\ (\boldsymbol{\Delta x_i})_j \end{pmatrix} + \boldsymbol{N}_j \frac{\boldsymbol{1}}{\boldsymbol{Z_0}} \begin{pmatrix} \boldsymbol{\Delta v}_j^+ - \boldsymbol{\Delta v}_j^- \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} (\boldsymbol{b_E})_j \\ (\boldsymbol{b_i})_j \end{pmatrix} . \tag{3.15}$$

Now rearrange equation :

$$\begin{pmatrix} \left((A_{EE})_j - (1/Z_0) I_j^*\right) & (A_{IE})_j \\ (A_{EI})_j & (A_{II})_j \end{pmatrix} \begin{pmatrix} \Delta \bar{v}_j \\ (\Delta x_i)_j \end{pmatrix} = \begin{pmatrix} (b_E)_j \\ (b_i)_j \end{pmatrix} - \left(\begin{pmatrix} (A_{EE}^*)_j \\ (A_{EI}^*)_j \end{pmatrix} + (1/Z_0) N_j \right) \Delta v_j^+ \tag{3.16}$$

where I_j^* is diagonal matrix with diagonal elements equal to 1 or -1, depending on the sign of the elements in corresponding row of N_j, $(A_{EE}^*)_j$ and $(A_{EI}^*)_j$ are matrices obtained by column permutation of $(A_{EE})_j$ and $(A_{EI})_j$, respectively.

To understand column permutation of $(A_{EE})_j$ and $(A_{EI})_j$, let's take

$$\begin{pmatrix} (A_{EE}^*)_j \\ (A_{EI}^*)_j \end{pmatrix} = N_j^* \quad \text{and} \quad \begin{pmatrix} (A_{EE})_j \\ (A_{EI})_j \end{pmatrix} = A_{Ext}.$$

To build N_j^*, first a zero matrix with same dimensions as N_j is created. Then each column of A_{Ext} is copied in the column of N_j^* corresponding to the respective external currents.

Equation involving last row ($N_1^T, N_2^T, ..., N_k^T$) of Equation (3.13):

$$\sum_{j=1}^{k} N_j^T \Delta x_j^{n+1} = 0 \tag{3.17}$$

which represents that incident waves for each subcircuit should be equal to reflected wave of another subcircuit, in other words sum of waves from connected subcircuits should be equal to zero. It is similar to Kirchhoff's voltage law (KVL) that in any close network, sum of voltage is zero. Equation (3.17) remain same as analysis based on nodal variables but now sum of waves equal to zero instead of voltages.

Now to write whole system of equation for general partitioned circuit shown in Figure 3.15, let's take

$$\boldsymbol{A}_j^w = \begin{pmatrix} \left(\boldsymbol{A_{EE}}\right)_j - \left(\boldsymbol{1}/\boldsymbol{Z_0}\right)\boldsymbol{I}_j^* & \left(\boldsymbol{A_{IE}}\right)_j \\ \left(\boldsymbol{A_{EI}}\right)_j & \left(\boldsymbol{A_{II}}\right)_j \end{pmatrix}, \quad \boldsymbol{\Delta x}_j^w = \begin{pmatrix} \boldsymbol{\Delta v}_j^- \\ \left(\boldsymbol{\Delta x}_i\right)_j \end{pmatrix} \text{ and}$$

$$\boldsymbol{N}_j^w = \begin{pmatrix} \begin{pmatrix} \boldsymbol{A}_{\boldsymbol{EE}}^* \end{pmatrix}_j \\ \left(\boldsymbol{A}_{\boldsymbol{EI}}^*\right)_j \end{pmatrix} + \left(\boldsymbol{1}/\boldsymbol{Z_0}\right)\boldsymbol{N}_j$$

The system of equations can be written as:

$$\begin{pmatrix} \boldsymbol{A}_1^w & & & & \boldsymbol{N}_1^w \\ & \boldsymbol{A}_2^w & & & \boldsymbol{N}_1^w \\ & & \ddots & & \vdots \\ & & & \boldsymbol{A}_k^w & \boldsymbol{N}_k^w \\ \boldsymbol{N}_1^T & \boldsymbol{N}_2^T & \cdots & \boldsymbol{N}_k^T & [\boldsymbol{0}] \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta x}_1^w \\ \boldsymbol{\Delta x}_2^w \\ \vdots \\ \boldsymbol{\Delta x}_k^w \\ \boldsymbol{\Delta v}_I^+ \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_k \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \vdots \\ \boldsymbol{s}_k \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{f}_1(\boldsymbol{x}_1^n) \\ \boldsymbol{f}_2(\boldsymbol{x}_2^n) \\ \vdots \\ \boldsymbol{f}_k(\boldsymbol{x}_k^n) \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{N}_1 \boldsymbol{i}_{\boldsymbol{I}}^n \\ \boldsymbol{N}_2 \boldsymbol{i}_{\boldsymbol{I}}^n \\ \vdots \\ \boldsymbol{N}_k \boldsymbol{i}_{\boldsymbol{I}}^n \\ \sum_{j=1}^{k} \boldsymbol{N}_j^{\boldsymbol{T}} \boldsymbol{x}_j^n \end{pmatrix} \tag{3.18}$$

Now apply Diakoptics to Equation (3.18). Equations are the same as Equations (3.5), (3.6) and (3.8) of analysis with nodal variables discussed in Section 3.1. First

$$\left(\boldsymbol{\Delta x}_j^w\right)^* = \boldsymbol{A}_j^{-1}\left(\boldsymbol{s}_j - \boldsymbol{f}_j(\boldsymbol{x}_j^n) - \boldsymbol{N}_j \boldsymbol{i}_{\boldsymbol{I}}^n\right) \tag{3.19}$$

can be found. Then wave vector $\boldsymbol{\Delta v}_I^+$ is obtained using $\left(\boldsymbol{\Delta x}_j^w\right)^*$ from :

$$\left(\sum_{j=1}^{k} \boldsymbol{N}_j^{\boldsymbol{T}} \left(\boldsymbol{A}_j^w\right)^{-1} \boldsymbol{N}_j^w\right) \boldsymbol{\Delta v}_{\boldsymbol{I}}^+ = \sum_{j=1}^{k} \boldsymbol{N}_j^{\boldsymbol{T}} \left(\boldsymbol{x}_j^w + \left(\boldsymbol{\Delta x}_j^w\right)^*\right) \tag{3.20}$$

Then unknown internal voltages ($(\boldsymbol{\Delta x}_i)_j$) of subcircuit j and unknown waves $\Delta \boldsymbol{v}_j^-$ can be found from :

$$\boldsymbol{\Delta x}_j^w = \begin{pmatrix} \boldsymbol{\Delta v}_j^- \\ (\boldsymbol{\Delta x}_i)_j \end{pmatrix} = \left(\boldsymbol{\Delta x}_j^w\right)^* - \left(\left(\boldsymbol{A}_j^w\right)^{-1} \boldsymbol{N}_j^w \boldsymbol{\Delta v}_{\boldsymbol{I}}^+\right) \tag{3.21}$$

Example

Now for simplicity, consider a circuit network with two subcircuits shown in Figure 3.16.

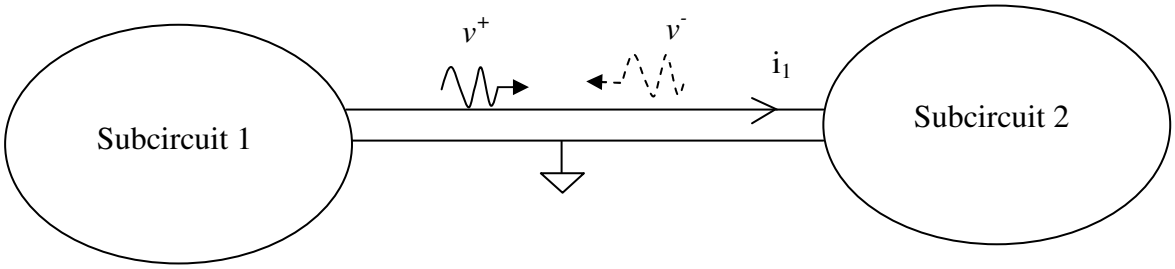


Figure 3.16 General circuit network partitioned into two subcircuits

Now system of equation for this circuit network can be written as:

$$\begin{pmatrix} \begin{pmatrix} \begin{pmatrix} A_{EE_1} - 1/Z_0 \\ A_{EI_1} \end{pmatrix} & \begin{matrix} A_{IE_1} \\ A_{II_1} \end{matrix} \end{pmatrix} & [0] & \begin{pmatrix} A^*_{EE_1} \\ A^*_{EI_1} \end{pmatrix} + \begin{pmatrix} 1/Z_0 \\ 0 \end{pmatrix} \\ [0] & \begin{pmatrix} \begin{pmatrix} A_{EE_2} + 1/Z_0 \\ A_{EI_2} \end{pmatrix} & \begin{matrix} A_{IE_2} \\ A_{II_2} \end{matrix} \end{pmatrix} & \begin{pmatrix} A^*_{EE_1} \\ A^*_{EI_1} \end{pmatrix} - \begin{pmatrix} 1/Z_0 \\ 0 \end{pmatrix} \\ 1 \quad 0 & -1 \quad 0 & [0] \end{pmatrix} \begin{pmatrix} \begin{pmatrix} v^- \\ x_{i_1} \end{pmatrix} \\ \begin{pmatrix} v^- \\ x_{i_2} \end{pmatrix} \\ (v^+) \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ 0 \end{pmatrix}. \tag{3.22}$$

3.2.2 Algorithm Flowchart

Using Equations (3.19), (3.20) and (3.21) the algorithm with waves is shown in Figure 3.17. Newton method is used to solve the nonlinear system of equations. The same flowchart from Figure 3.9 is used, but Δx is calculated using the flowchart of Figure 3.17 instead of Figure 3.8.

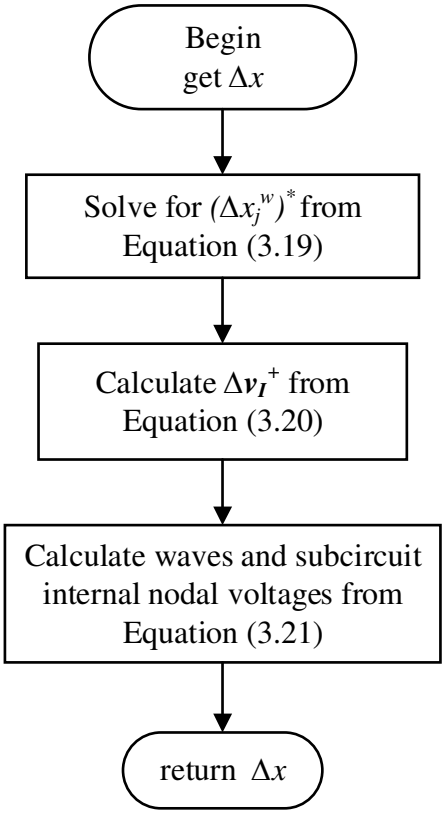


Figure 3.17 Reference algorithm flowchart of analysis based on scattering waves

Inter-processor Communication Analysis

Parallel implementation of analysis with scattering waves is same as analysis with nodal variables as discussed in Section 3.1.2. The key difference between analysis with nodal variables and analysis with waves variables is structure of $\boldsymbol{A}_j^w$ and $\boldsymbol{N}_j^w$ blocks. In proposed analysis, the structure of $\boldsymbol{N}_j^w$ block is not as simple as $\boldsymbol{N}_j$ block of analysis with nodal variables. Hence slave processors have to perform more work compared to work needed for analysis with nodal variables. Same as analysis with nodal variables, in analysis with waves interconnect block (C) and interconnect source vector ($\boldsymbol{b}_{k+1}$) are zero. Slave processors calculate $(\boldsymbol{x}_j^w)^*$, $\boldsymbol{S}_j^n$, $\boldsymbol{w}_j^n$ and send that information to master processor which calculates $\boldsymbol{S}^n$, $\boldsymbol{w}^n$ and $\Delta\boldsymbol{v}_I^+$ (Equation 3.20). where,

$$\boldsymbol{S}^n = -\sum_{j=1}^{k} \boldsymbol{S}_j^n \text{ and } \boldsymbol{w}^n = -\sum_{j=1}^{k} \boldsymbol{w}_j^n .$$

And at last slave processors calculate $\Delta\boldsymbol{x}_j^w$ (Equation 3.21) from $\Delta\boldsymbol{v}_I^+$.

3.2.3 Complete Example

Now a linear circuit example is presented here again for analysis based on waves. Figure 3.18 shows linear circuit with wave variables. i_1 and i_2 are external currents.

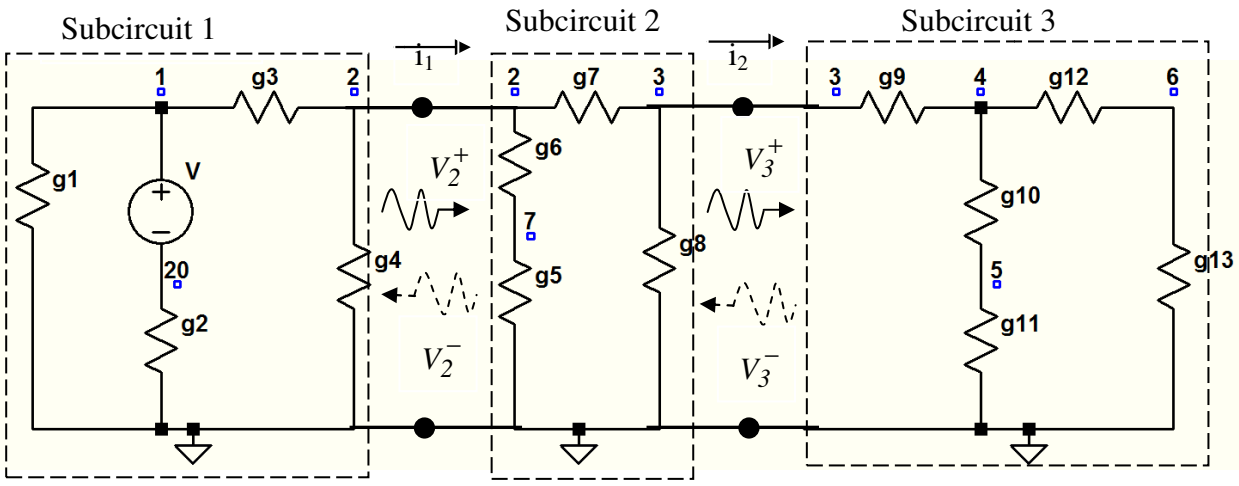


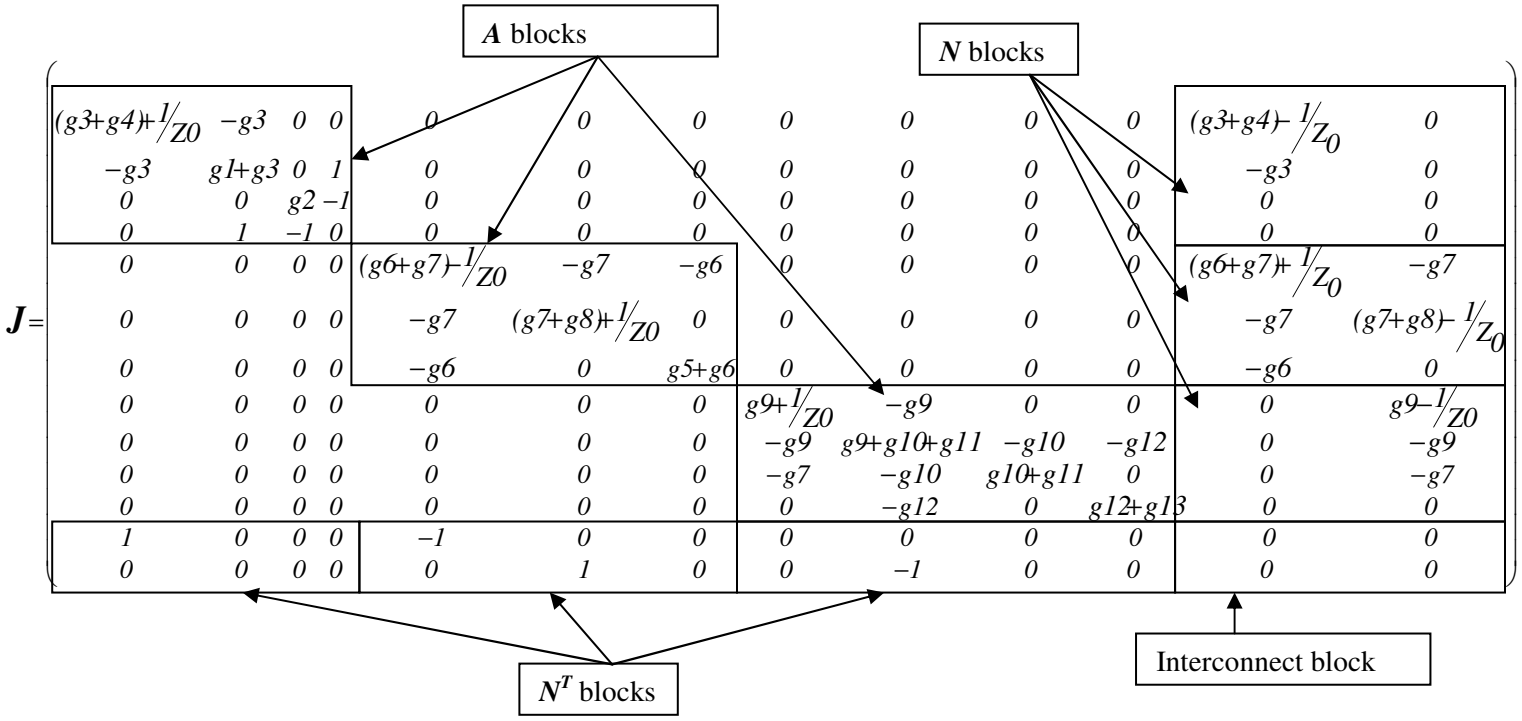
Figure 3.18 Partitioned linear circuit with waves

Jacobian matrix for analysis with waves is slightly different than analysis with nodal variables. It is more dense than Jacobian matrix of analysis with nodal variables. From Equation (3.18) system of equations for analysis based on waves can be written as :

$$\boldsymbol{J}\ \boldsymbol{v}=\boldsymbol{s} \tag{3.23}$$

where unknown vector $\boldsymbol{v}=\left(v_2^-\ \ x_{20}\ \ x_1\ \ i_v\ \ v_2^-\ \ v_3^-\ \ x_7\ \ v_3^-\ \ x_4\ \ x_5\ \ x_6\ \ v_2^+\ \ v_3^+\right)^{\mathrm{T}}$,

source vector $\boldsymbol{s}=(0\ \ 0\ \ V\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0)^{\mathrm{T}}$ and Jacobian matrix J is described on following page.



where Z_0 is reference impedance. The difference between jacobian matrix of analysis with waves and analysis with nodal variables is in subcircuit blocks $\boldsymbol{A}_j$ and $\boldsymbol{N}_j$ blocks. In analysis based on nodal variables $\boldsymbol{N}_j$ blocks contain all entries '0', except for only one or more of the entries in them containing '±1' depending on external currents, whereas in analysis with waves, $\boldsymbol{N}_j$ blocks are extracted from subcircuit block $\boldsymbol{A}_j$ with sign convention shown in Equations (3.9) and (3.10). And hence one of the advantages of the partitioning of analysis with nodal variables i.e. constant $\boldsymbol{N}_j$ blocks over each Newton iteration is lost. Modifying $\boldsymbol{A}_j$ and $\boldsymbol{N}_j$ require extra processing time compared to analysis based on nodal variables. $\boldsymbol{N}_j^{\boldsymbol{T}}$ blocks structure in analysis with waves are same as analysis with nodal variables.

3.3 Code Implementation

Cardoon is a general circuit simulator developed in-house. It is coded in Python but uses C/C^{++} libraries for efficiency. Presently, Cardoon simulator supports nonlinear models such as diode, BJT, MESFET and MOSFET. The operating point analysis methods, developed for this research are: EOP and WAVEOP. EOP is the operating point analysis based on nodal variables and WAVEOP is the operating point analysis based on waves. Code for EOP analysis contains approximately 365 lines and code for WAVEOP analysis contains 384 lines of code. These codes are written in Python but matrix handling and matrix multiplication have been done by C/C^{++} libraries. These libraries perform calculations much faster than writing vector multiplication function in Python. This code uses the following libraries: numpy (matrix and vector support) [35], pycppad (automatic differentiation) [36], scipy (sparse matrix support) [37] and ipython (iterative shells) [38]. These libraries are interfaces between python and C/C^{++} language.

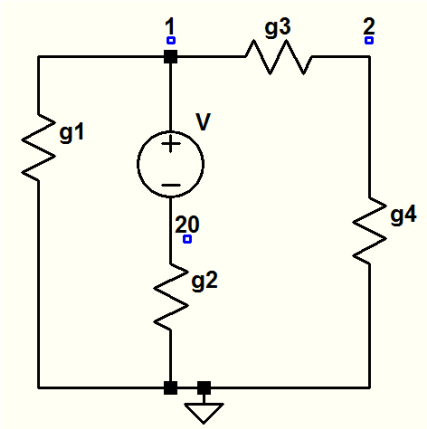
Parameters such as *maxiter*, *maxdelta*, *reltol* and *abstol* are used to control analysis. Parameters such as *reltol* and *abstol* including sparsity can be changed using *.options* keyword in netlist. Table 3.1 shows such parameters used for simulation with their values and description [14].

Table 3.1 Parameters and their default values

Variable name	Default Parameter value	Description
maxdelta	50	Maximum allowed deviation in one Newton iteration
reltol	1e-4	Relative tolerance for nodal variables
abstol	1e-07	Absolute tolerance for nodal variables
maxiter	100	Set maximum number of iterations
gcomp	1e-6 S	Add compensation network for EOP analysis
Sparse	1	Change sparsity of analysis

To simulate a circuit, it must be described in a netlist file. The program reads a netlist file (Figure 3.19(b)), builds the circuit described there and runs any specified analysis. Figure 3.19(b) shows the corresponding netlist of subcircuit shown in Figure 3.19(a). The first line of netlist defines

subcircuit with external nodes, elements should be described with node numbers at which they are connected and subcircuit description ends with *.ends*.



(a)

```
.subckt subciruit1 2
res:r1 1  gnd  r=50
res:r2 20 gnd  r=50
res:r3  1   2  r=50
res:r4  2 gnd  r=50
vdc:vdd 1  20  vdd=2v
.ends
```

(b)

Figure 3.19 Netlist Example (a) Subcircuit block (b) Netlist of subcircuit

Each analysis type is implemented by adding a specialized class to the code. The formulation using nodal variables is implemented in a class named DCOP that contains three main methods:

1. run () : It's main entry point of the reference algorithm and includes Newton's method
   - Check convergence for Newton method.
2. init_blocks () : Initialize class attributes that are needed for subcircuit decomposition and create incidence matrices
   - Initialize $\boldsymbol{N_j}$, and $\boldsymbol{N_j^T}$ blocks
3. get_deltax() : This function creates Jacobian matrix and calculates nodal equations.
   - Solve Equations (3.5), (3.6) & (3.8) and send $\boldsymbol{\Delta x}$ to run() method to check convergence, where $\Delta \boldsymbol{x} = (\Delta \boldsymbol{x_1}, \Delta \boldsymbol{x_2}, \ldots, \Delta \boldsymbol{x_J}, \Delta \boldsymbol{i_I})^{\mathbf{T}}$.

Compensation Network for EOP analysis

Another feature of EOP analysis is to add a compensation network to the external nodes of subcircuits to solve $\boldsymbol{A}_j$ singularity. If a subcircuit has a floating external node or a node internally loaded with a very high impedance, it produces an ill-conditioned matrix. In such case $\boldsymbol{A}_j$ singularity will arise and solution of such circuit network is not possible. EOP analysis has parameter called g_{comp} to add compensation network at the external nodes of subcircuits. Transconductances are added to external nodes of each subcircuits to prevent floating nodes. As shown in Figure 3.20, EOP analysis add $+g_{comp}$ to Subcircuit 1 and $-g_{comp}$ to Subcircuit 2. $-g_{comp}$ will compensate the effect of g_{comp}.

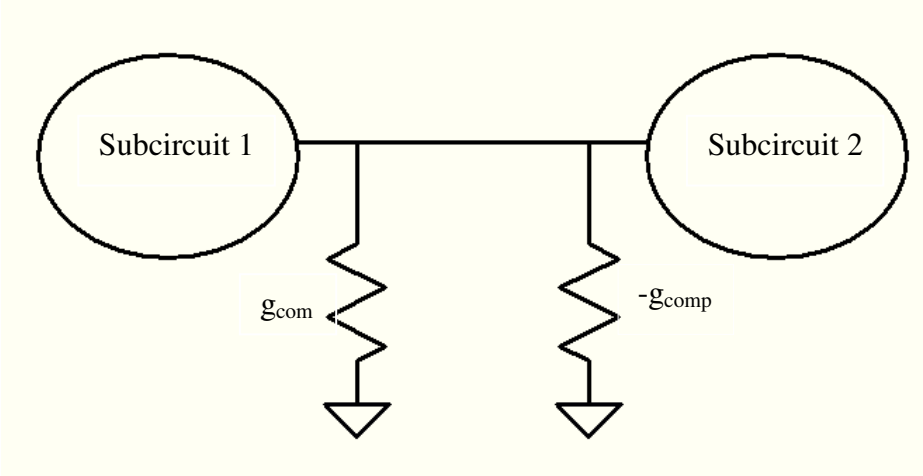


Figure 3.20 General circuit diagram with compensation network for EOP analysis

A flowchart for this analysis is presented in Figure 3.21.

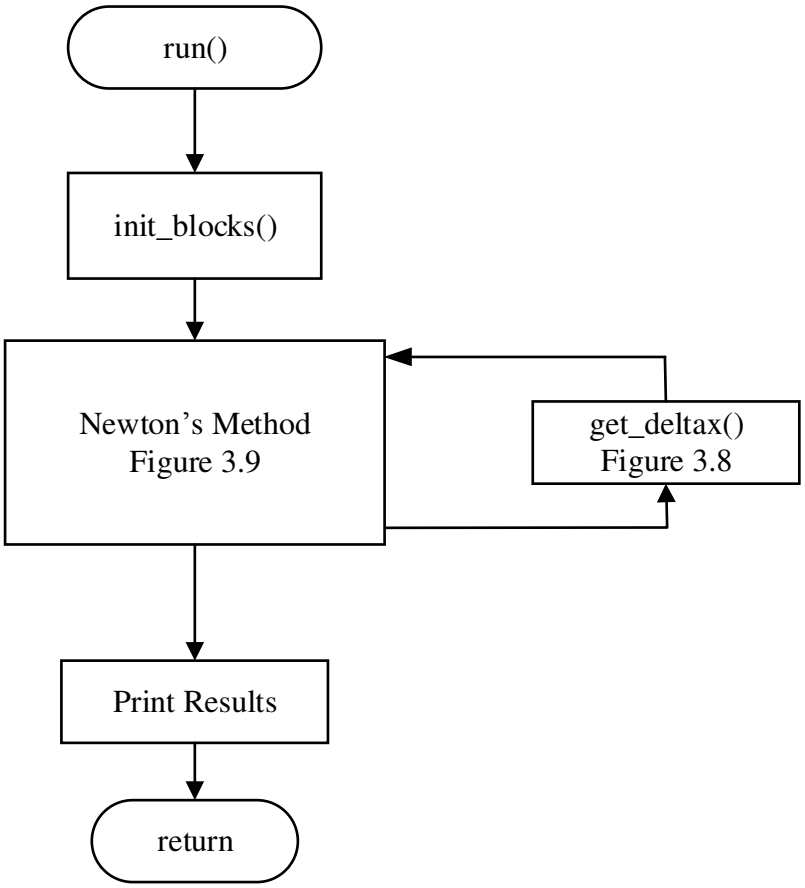


Figure 3.21 Flowchart of EOP analysis

Now for WAVEOP analysis there are three main methods:

1. run () : It's main entry point of the reference algorithm and includes Newton's method
   - Check convergence for Newton method.
   - Convert waves back to the nodal voltages for WAVEOP analysis.
2. init_blocks () : Initialize class attributes that are needed for subcircuit decomposition and create incidence matrices for both analyses

- Initialize $\boldsymbol{N_j}$, $\boldsymbol{N_j^T}$ blocks and incident waves list

3. get_deltax() : This function creates Jacobian matrix and calculates nodal equations.
   - Create $\boldsymbol{A_j^*}$ and $\boldsymbol{N_j^*}$ blocks, perform column permutation of $\boldsymbol{(A_{EE})_k}$ and $\boldsymbol{(A_{EI})_k}$, solve Equations (3.19), (3.20) & (3.21) and send $\boldsymbol{\Delta x}$ vector to run() method, where, $\Delta \boldsymbol{x} = ((\Delta \boldsymbol{v_1^-}, \Delta \boldsymbol{x_1}),\ (\Delta \boldsymbol{v_2^-}, \Delta \boldsymbol{x_2}), \ldots., (\Delta \boldsymbol{v_k^-}, \Delta \boldsymbol{x_k}), (\Delta \boldsymbol{v_1^+}, \Delta \boldsymbol{v_2^+}, \ldots, \Delta \boldsymbol{v_k^+}))^{\mathbf{T}}$

The analysis flowchart is presented in Figure 3.22.

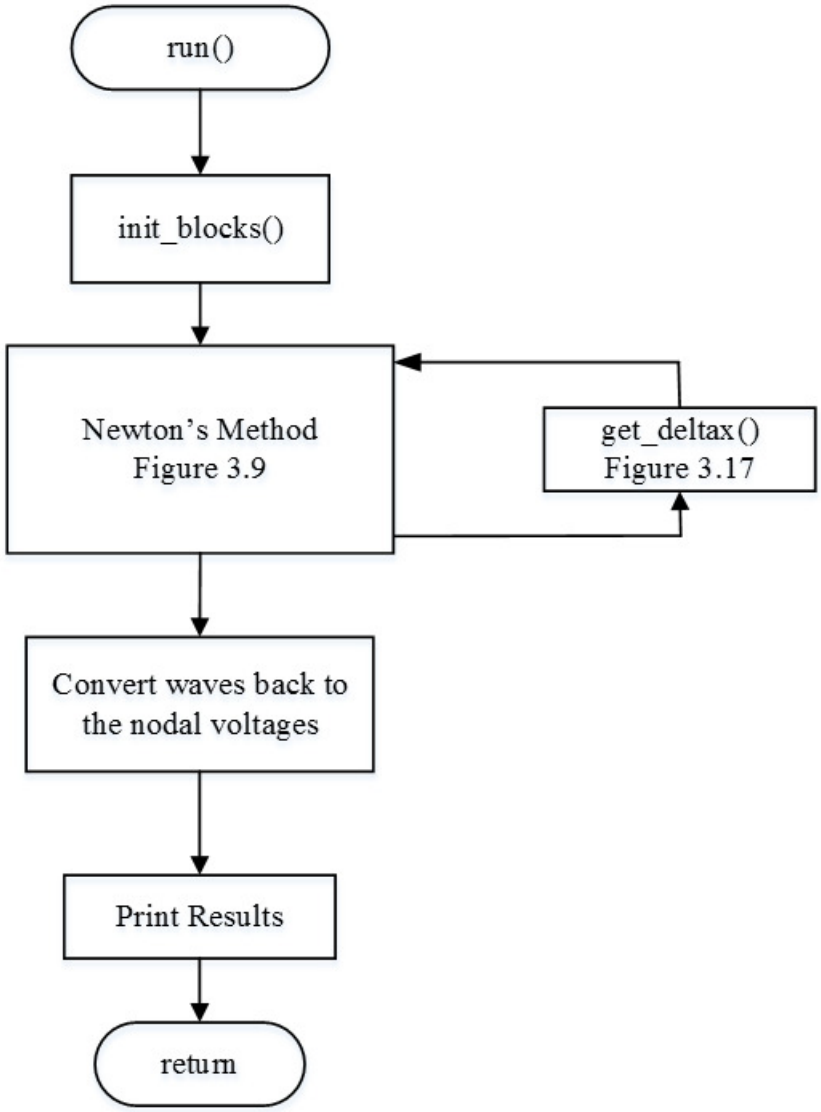


Figure 3.22 Flowchart of WAVEOP analysis

Overall, this code is proof of concept. It is not most efficient implementation and there is lots of unnecessary overheads e.g. handling incidence matrices ($\boldsymbol{N}_j$ blocks). In both analyses, $\boldsymbol{N}_j$ blocks are treated as dense matrix, but it is mostly zeros. So, implementation is basic, not optimised.

To run the analysis, the lines shown in Figure 3.23 can be written in netlist file with convergence parameters like *maxiter*, *maxdelta*, *gcomp* for EOP analysis to set required value if default value is not sufficient to get convergence and reference impedance (Z_0) can be set for WAVEOP analysis. Parameter *gcomp* is for EOP analysis to add compensation network at the port of each subcircuit. This compensation network is optional. The reference impedance (Z_0) parameter in WAVEOP is called `z0`. This parameter is essential for every circuit that simulates for WAVEOP. There is no optimum reference impedance value that works for any circuit. These both parameters can be accessed with analysis line shown in Figure 3.23.

```
.analysis eop gcomp=0.01 maxiter=250 maxdelta=3

.analysis waveop z0=100 maxiter=250 maxdelta=3
```

Figure 3.23 Reference analysis code lines

3.4 Simulation Results and Discussion

Operating point analysis of several circuits performed using the methods proposed in this research are presented in this section. The regular operating point analysis without system decomposition is named OP analysis. Simulation result comparison of EOP and WAVEOP analyses with regular OP analysis are given in this section. All circuit examples are presented with a simulation result summary, which compares number of subcircuits, number of iterations indicates number of Newton iterative steps required to get solution of circuit network and simulation time is the time required by analysis to create matrix blocks, solve Equations (3.5), (3.6) & (3.8) for EOP analysis and Equations (3.19), (3.20) & (3.21) for WAVEOP analysis and run Newton method till convergence.

Simulation result of all analyses are tested serially on one processor. Nodal voltages of OP, EOP and WAVEOP analysis are same for all circuit examples discussed in this section. Parameters like *maxdelta*, *reltol* and *abstol* are kept same for all analyses. All circuit examples presented in this section are simulated using sparse matrices.

3.4.1 Linear Circuit

Figure 3.10 shows a linear circuit example. This linear circuit is divided in three subcircuits. It is excited with an ideal voltage source V = 5 volts and all resistor values are same and equal to 10 Ω. Table 3.2 shows simulation result summary of EOP and WAVEOP analyses compared with OP analysis. Simulation result of WAVEOP is given with different values of reference impedance (Z_0) in Table 3.2. Here, number of iterations and simulation time with Z_0 = 100 Ω and Z_0=1 kΩ are same.

Table 3.2 Simulation result summary of linear circuit

Analysis	Number of subcircuits	Number of Iterations	Simulation time
Operating Point Analysis	0	8	0.01s
EOP Analysis	3	8	0.02s
WAVEOP Analysis	3	(Z_0=10Ω) 20	0.09s
		(Z_0=100Ω, 1kΩ) 18	0.08s
		(Z_0=10kΩ) 22	0.1s

3.4.2 Nonlinear Circuit

This reference approach can also be used to simulate nonlinear circuits. Figure 3.24 shows nonlinear circuit with two 2N2222 BJTs in Darlington pair, DC source and three resistors. This nonlinear circuit is divided into two subcircuits as shown in Figure 3.24. Subcircuit 1 contains BJT Q_1, two resistors R_1, R_2 and Subcircuit 2 has transistor Q_2 and resistor R_3. This circuit has total of 19 nodes. Subcircuit 1 has 9 nodes whereas Subcircuit 2 has 10 nodes. Resistor R_1, R_2 = 100 kΩ & R_3 = 3 kΩ, power supply V_{CC} = 5 volts.

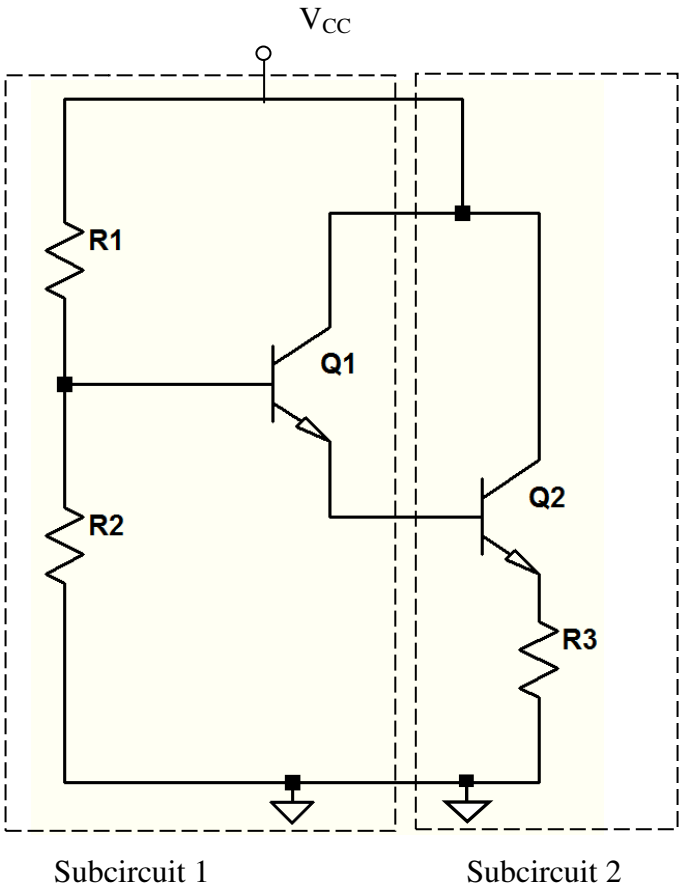


Figure 3.24 Nonlinear circuit partitioned into two subcircuits

Simulation result of nonlinear circuit (Figure 3.24) for regular operating point analysis and reference analysis with nodal variables (EOP) are shown in Table 3.3. Simulation result of WAVEOP is given with different values of reference impedance (Z_0) in Table 3.3. Here, number of iterations and simulation time with Z_0= 100 Ω and Z_0 =1 kΩ are same.

Table 3.3 Simulation result summary of nonlinear circuit

Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP analysis	0	12	0.02s
EOP Analysis	2	12	0.03s
WAVEOP Analysis	2	(Z_0=10Ω) 26	0.12
		(Z_0=100Ω, 1kΩ) 27	0.12s
		(Z_0=10kΩ) 24	0.11s

3.4.3 Soliton Line

Figure 3.25 shows a soliton network/ nonlinear transmission line. Nonlinear transmission line are high impedance waveguides which are periodically loaded with reverse biased diodes. These diodes appear as variable capacitors (varactors) [39]. This circuit network can be divided up to 48 subcircuits. Transmission line is modeled with 20 cascade sections and each section contains a R-L-G-C circuit. Transmission line has total of 3025 nodes. Here two separate examples are given with soliton network divided into different number of subcircuits. In Figure 3.25 Soliton network is divided into 4 subcircuits. Subcircuit 1 has 64 nodes, Subcircuit 2 has 946 nodes, Subcircuit 3 has 1009 nodes and Subcircuit 4 has 1006 nodes.

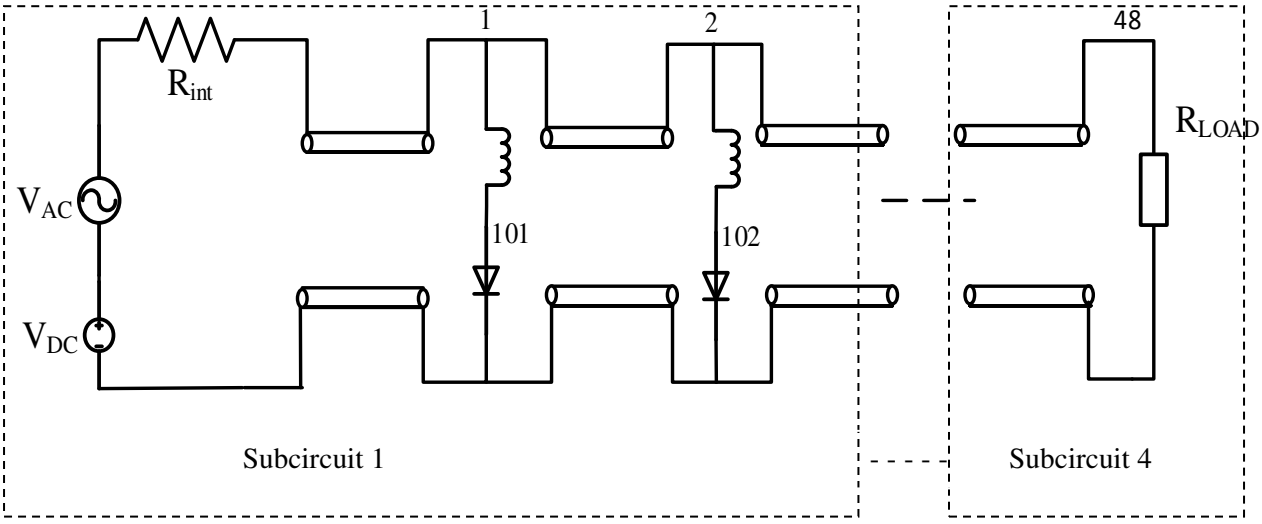


Figure 3.25 Soliton circuit network divided in four subcircuits

Table 3.4 shows simulation result summary of OP analysis, EOP analysis and WAVEOP analysis for soliton circuit divided in four subcircuits. Reference impedance (Z_0) for WAVEOP analysis is 100 Ω. Simulation result of WAVEOP is given with different values of reference impedance (Z_0) in Table 3.4.

Table 3.4 Simulation result summary of soliton network divided in 4 subcircuits

Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP Analysis	0	3	0.18s
EOP Analysis	4	3	0.17s
WAVEOP Analysis	4	(Z_0=10Ω) 17	0.44s
		(Z_0=100Ω) 15	0.32s
		(Z_0=1kΩ) 14	0.32s
		(Z_0=10kΩ) 21	0.44s

Soliton network shown in Figure 3.25 can be further divided into 12 subcircuits. There are total of 3034 nodes. Subcircuits 1, 2, 3, 5, 6, 7, 8, 11 and 12 have 253 nodes, Subcircuit 4 has 250

nodes, Subcircuit 9 has 313 nodes and Subcircuit 10 has 194 nodes. Number of iterations and simulation time for soliton network partitioned into 12 subcircuits will be same as soliton network partitioned into 4 subcircuits for OP analysis. But for EOP and WAVEOP require more simulation time for soliton network partitioned into 12 subcircuits compared to soliton network partitioned into 4 subcircuits, as now analyses have to construct and calculate 12 subcircuit blocks instead of 4. Table 3.5 shows simulation result comparison of regular operating point analysis (OP) with reference approaches EOP and WAVEOP for soliton network divided in 12 subcircuits. Simulation result of WAVEOP is given with different values of reference impedance (Z_0) in Table 3.5.

Table 3.5 Simulation result summary of soliton network divided in 12 subcircuits

Analysis	Number of subcircuits	Number of Iterations	Simulation time
Operating Point Analysis	0	3	0.18
EOP Analysis	12	3	0.20
WAVEOP Analysis	12	(Z_0 =1kΩ) 16	0.70s
		(Z_0 =10kΩ) 18	0.77s

3.4.4 Summing Amplifier

Figure 3.26 shows summing amplifier. It is implemented with a 741 operational amplifier (LM741) and feedback network with resistors. LM741 has 26 BJTs. This circuit contains large number of nonlinear elements (BJTs) and hence it is used to test response of proposed EOP and WAVEOP analysis. The resistor values of the summing amplifier are as follows: R_{1ext} = 5 kΩ, R_{2ext} = 20 kΩ, R_{3ext} = 20 kΩ, R_{4ext} = 3.3 kΩ. There are total of 192 nodes in this circuit. This circuit is divided into 2 subcircuits. Subcircuit 1 consists of 3 external resistors R_{1ext} , R_{2ext} , R_{3ext} and voltage sources V_1 and V_2. This subcircuit has 8 nodes. Subcircuit 2 consists of operational amplifier LM741 which has 184 nodes. Because of uneven partitioning blocks, load balancing is not good in this circuit partitioning. If these subcircuits are assigned to two different processors then processor with three resistors and voltage sources will complete its calculations faster than processor with operational amplifier . Processor with smaller subcircuit has to wait for results from other processor. Hence parallel simulation of this circuit is not efficient.

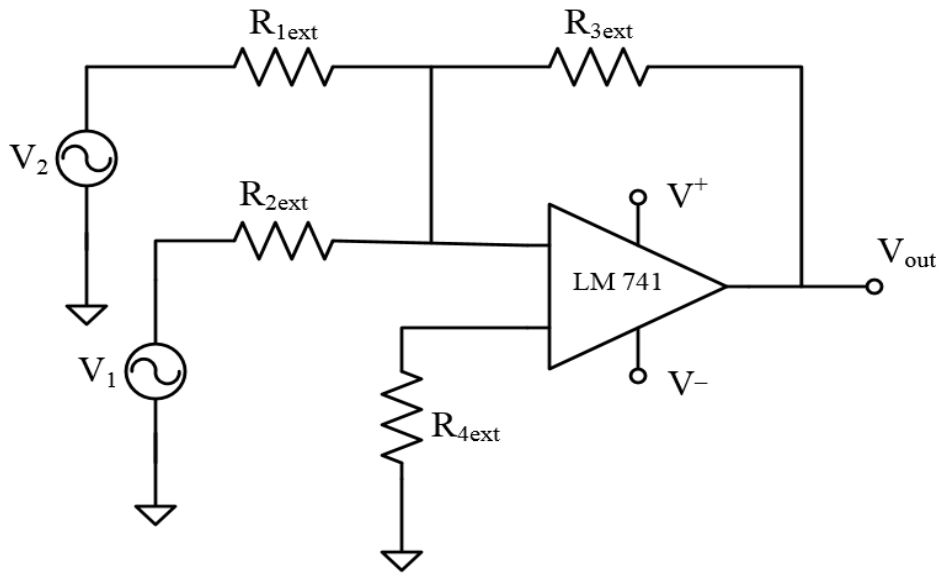


Figure 3.26 Summing amplifier

In EOP analysis compensation network has been added for this circuit example. Results of EOP analysis with and without *gcomp* are shown in Table 3.6. Simulation results of WAVEOP analysis is given with different values of reference impedance (Z_0) in Table 3.6. Here, number of iterations and simulation time with Z_0 = 100 Ω and Z_0 =10 kΩ are same. Simulation result summary of summing amplifier (Figure 3.25) with OP, EOP and WAVEOP analysis are shown in Table 3.6. *maxdelta* is 3 for this example.

Table 3.6 Simulation result summary of summing amplifier circuit

Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP analysis	0	59	0.16s
EOP Analysis	2	72	0.32s
		42 (*gcomp* = 1mS)	0.20s
WAVEOP Analysis	2	(Z_0=10Ω) 62	0.56s
		(Z_0=100Ω, 10kΩ) 59	0.55s
		(Z_0=1kΩ) 60	0.55s

For this circuit example adding *gcomp* reduces number of iteration for EOP analysis. In the subcircuit 2 containing operational amplifier, base of two BJTs, which are connected at the input of LM 741, are floating. These two nodes see infinite impedance, as load connecting to those

nodes are located at Subcircuit 1 with external resistors and voltage sources. If Subcircuit 2 tries to deliver current to a load but it cannot deliver, as it is in another subcircuit and hence circuit might have convergence problem. If we add *gcomp* at the floating node, then subcircuit sees some load impedance and it helps Newton method for fast convergence. And hence, by adding *gcomp* in the above circuit example number of Newton iteration reduces compare to analysis without *gcomp*.

3.4.5 Microwave Low Noise Amplifier

The following example is a low noise microwave amplifier which is tested for EOP analysis. Figure 3.27 shows low noise amplifier using two LMA411 low noise microwave amplifiers. 0.25 um CMOS technology is used for LMA411 amplifier. This circuit network is divided in three subcircuits: Subcircuit 1 contains voltage source and resistors, Subcircuits 2 and 3 have low noise microwave amplifiers. These two amplifiers and Subcircuit 1 is connected via transmission line as shown in Figure 3.27. This circuit has total of 851 nodes. Subcircuit 1 has 5 nodes, Subcircuit 2 and 3 have 423 nodes.

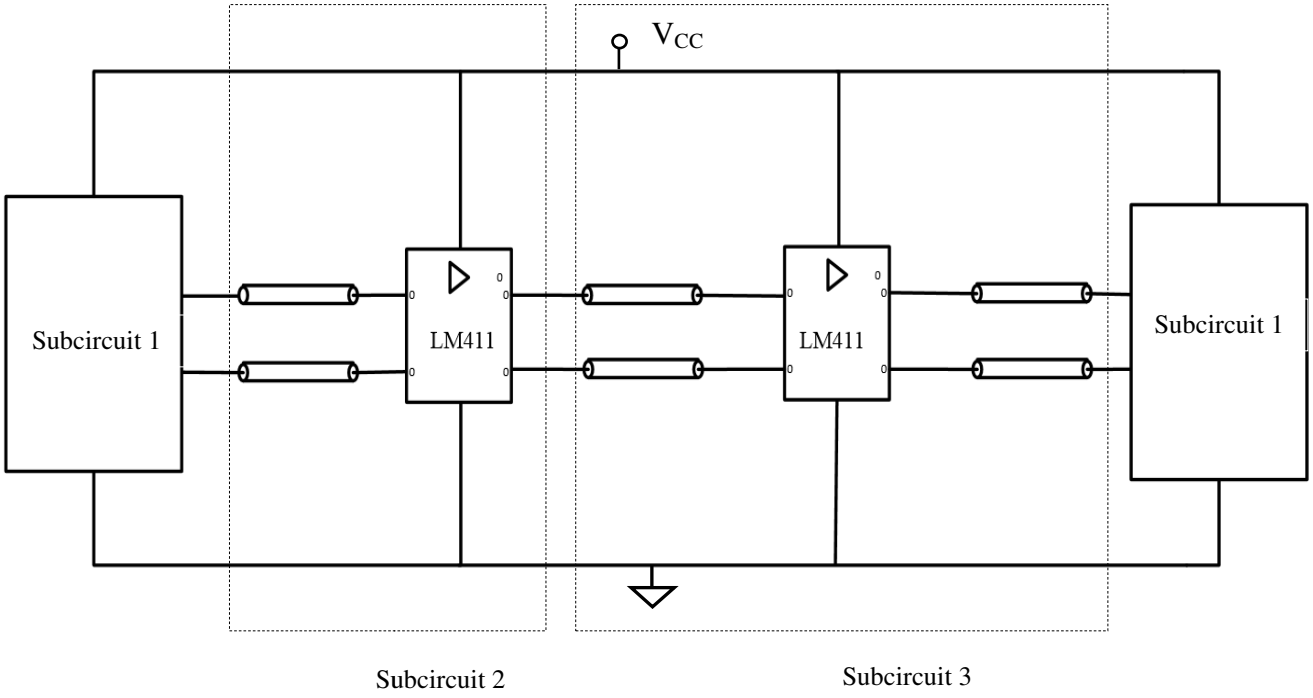


Figure 3.27 Low noise microwave amplifier circuit (double line indicate transmission line)

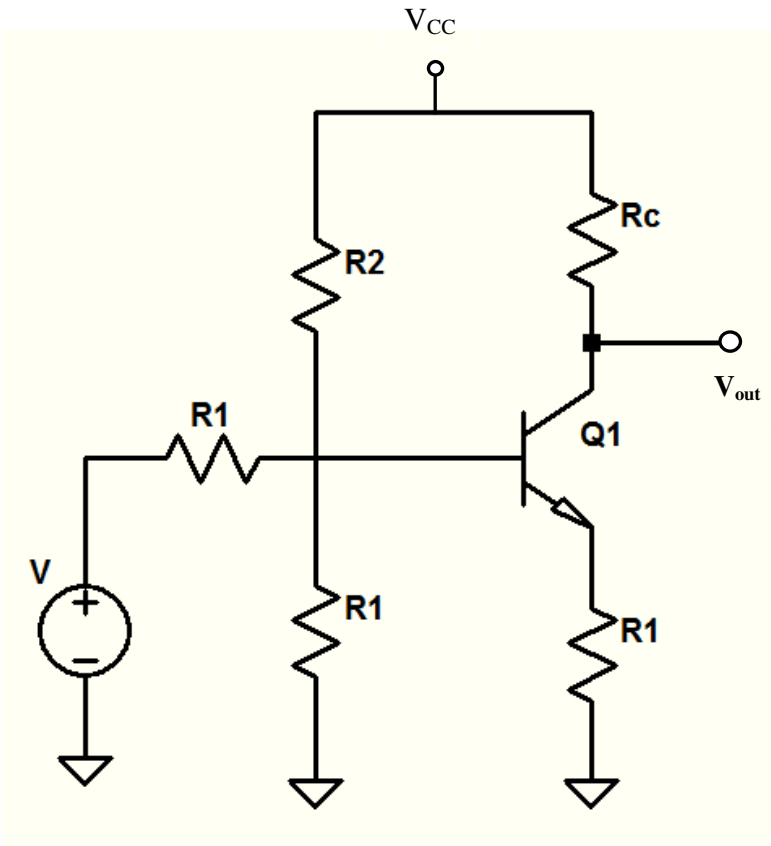
Table 3.7 shows simulation results of low noise amplifier circuit compared with OP and EOP analysis. To perform EOP analysis, compensation network has been added to each subcircuit as there is capacitive coupling between each subcircuit due to transmission line. Capacitors are open circuit in DC analysis and hence each subcircuit sees infinite impedance at external ports. Compensation network removes capacitive coupling by adding *gcomp* at external ports of each subcircuit. In this circuit example three subcircuits are sharing common node which is power supply and hence as explained in Section 3.2, WAVEOP analysis won't work for this circuit. With this type of connection, formation of Jacobian matrix becomes more complex and consequently it makes the code harder to implement.

Table 3.7 Simulation result summary of low noise amplifier circuit

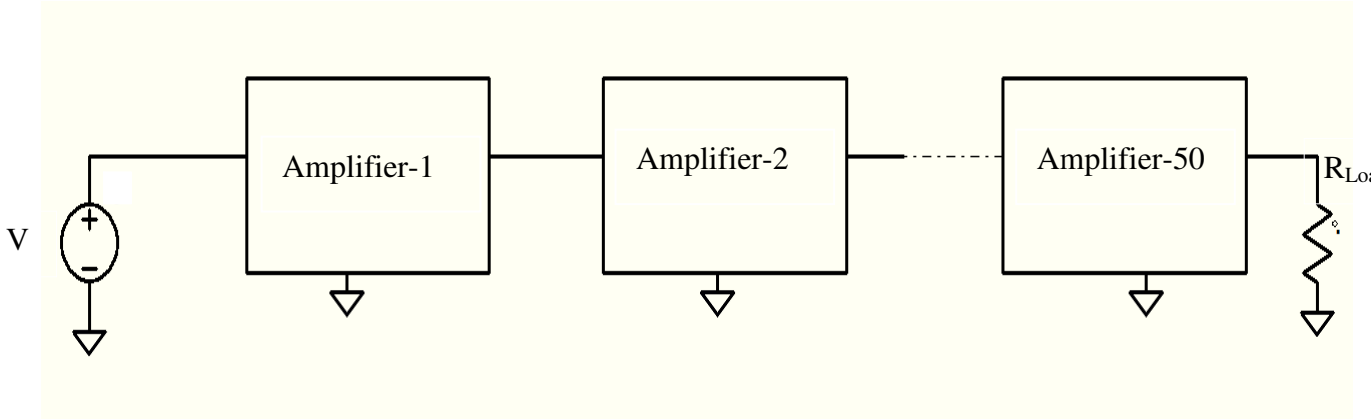
Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP Analysis	0	8	0.05s
EOP analysis (gcomp = 1mS)	3	8	0.07s

3.4.6 Transistor Amplifier

Figure 3.28 (a) shows a transistor amplifier circuit schematic. Resistors values are as follows : R_1 = 1.67 kΩ, R_2 = 6.66 kΩ and R_c = 900 Ω. Power supply voltage Vcc = 10 V, input voltage V = 5 V. To test correctness of our code, chain of such amplifier is made with 50 amplifiers (Figure 3.28 (b)). Each amplifier resides in one subcircuit and hence there are 50 amplifiers connected in cascade. Each subcircuit has 14 nodes and total number of nodes in 50 cascade amplifier circuit are 700.



(a)



(b)

Figure 3.28 Circuit example (a) Transistor amplifier Circuit (b) 50 cascade amplifier chain

Table 3.8 presents the simulation result summary of OP, EOP and WAVEOP analysis with number of iterations and CPU time for 50 cascade amplifier circuit. Simulation result of WAVEOP is given with different values of reference impedance (Z_0) in Table 3.8.

Table 3.8 Simulation result summary of 50 cascade amplifiers

Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP analysis	0	4	0.30s
EOP analysis	50	5	0.42s
WAVEOP analysis	50	(Z_0 =10Ω) 25	4.06s
		(Z_0 =100Ω) 22	3.47s
		(Z_0 =10kΩ) 19	2.88s

Now 500 amplifiers are cascaded instead of 50 cascade amplifiers shown in Figure 3.28 and its simulation result is shown in Table 3.9. This circuit network has 500 subcircuits and each subcircuit has one transistor amplifier. There are around 7000 nodes in this circuit network and each subcircuit has 14 nodes. Simulation results of WAVEOP with Z_0 = 10 Ω, 100 Ω, and Z_0 =10 kΩ are same.

Table 3.9 Simulation Results Summary of 500 Cascade Amplifiers divided in 500 subcircuits

Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP analysis	0	4	1.01s
EOP analysis	500	4	30.33s
WAVEOP analysis	500	(Z_0 =10Ω, 100 Ω, 10k Ω) 26	188.80s

Diakoptics is not implemented in OP analysis. This analysis solves whole Jacobian matrix as is without decomposing in blocks, whereas simulation time for EOP and WAVEOP also includes the time to decompose the system of equations in blocks, solve each block separately and perform global updates. In both analyses each partition is solved serially and synchronized later on to obtain the solution of the original circuit at each NR iteration. Hence, if there are large

number of subcircuits then EOP and WAVEOP analysis cannot improve simulation time significantly compared to regular OP analysis when executed on a single processor (Tables 3.2 to 3.9). However performance of EOP analysis is not worse. If it is implemented for parallel simulation then it would be a lot faster compared to regular OP analysis except for simulation results shown in Table 3.10.

Now consider 500 cascade amplifier circuit divided in 5 subcircuits. Table 3.10 shows simulation result of 500 amplifiers divided in 5 subcircuits. This circuit network has total of 5506 nodes. Subcircuit 1 has 1102 nodes, Subcircuit 2, 3,4 and 5 have 1101 nodes each. Simulation result of WAVEOP is given with different values of reference impedance (Z_0) in Table 3.10. Here WAVEOP analysis is simulated with different values of reference impedance (Z_0).

Table 3.10 Simulation results summary of 500 cascade amplifiers divided in 5 subcircuits

Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP analysis	0	4	1.05s
EOP analysis	5	4	1.48s
WAVEOP analysis	5	(Z_0 =10Ω) 22	3.84s
		(Z_0 =100Ω) 19	3.53s
		(Z_0 =1kΩ) 16	2.77s
		(Z_0 =10kΩ) 14	2.93s

From simulation results Table 3.10, it is clear that if the number of partitioned blocks per circuit are reduced then EOP analysis can gain significant speed up compared to circuit with more number of partitioned blocks. Consider the simulation result (Table 3.9) of 500 amplifiers cascade circuit divided into 500 subcircuits. EOP analysis requires 30.33 seconds to get solution. Now if the same circuit is partitioned into 5 subcircuits then EOP analysis gets significant speedup and takes only 1.48 seconds to get solution (Table 3.10), as there are only 5 subcircuit blocks to create and calculate compared to 500 subcircuit blocks. Similarly if any industrial circuit has thousands of subcircuits for example then it is impractical to assign each subcircuit to one processor. If EOP simulated in parallel by dividing it into a limited number of subcircuits

and solve each subcircuit block in parallel, then it will be significant speedup in simulation time. Another reason for slow response of EOP analysis is the insufficient handling of N_j blocks. Presently code treat N_j blocks as dense matrices despite their most of the entries are zeros. It could be optimised with much better performance.

All these examples discussed here are simulated with sparse matrix. In EOP and WAVEOP analysis, there is sparse parameter to set preference whether this analysis is simulated with sparse matrix or without sparse matrix. This variable can be accessed by *.options* keyword. If equation is solved with sparse matrix then analysis won't consider zeros in the calculation and on the other hand analysis with dense matrix will consider zeros in the calculation. For example, if system of equation shown in Figure 3.12 is solved with sparse matrix then EOP analysis solves only nonzero blocks and eliminates lots of blocks which are zeros. Dividing circuits into subcircuits gives similar effect as sparse matrix. This would save time as simulator has less work to do and would gain speed up compared to regular operating point analysis. Table 3.11 shows simulation result of 500 cascade transistor amplifiers circuit simulated with dense matrix. This circuit is divided in 5 subcircuits. WAVEOP analysis is simulated with different values of reference impedance (Z_0) and it is shown in Table 3.11.

Table 3.11 Simulation results summary of 500 cascade amplifiers divided into 5 subcircuits with dense matrix

Analysis	Number of subcircuits	Number of Iterations	Simulation time
OP analysis	0	4	198.71s
EOP analysis	5	4	6.46s
WAVEOP analysis	5	(Z_0 =10Ω) 22	29.35s
		(Z_0 =100Ω) 19	25.56s
		(Z_0 =1kΩ) 16	21.80s
		(Z_0 =10kΩ) 14	19.16s

From simulation result shown in Table 3.11, it is clear that simulation using dense matrix is much faster for EOP analysis compared to OP and WAVEOP analysis even if it is simulated serially on one processor. EOP analysis is approximately 30 times faster than OP analysis. WAVEOP analysis needs more simulation time compared to both analyses. As discussed in

Chapter 1 simulation cost is proportional to S^a, where S represents the original matrix size and a depends on the sparsity of the circuit matrix. For sparse matrix a varies from 1.1 to 2.4 and for dense matrix a = 3. For 500 cascade amplifiers divided in 5 subcircuits simulated with dense matrix,

$$\textit{(simulation cost of OP analysis)} \propto (5506)^3 \qquad \text{and,}$$

$$\textit{max (simulation cost of EOP analysis)} \propto (1102)^3$$

From simulation results, it is clear that WAVEOP analysis is not efficient compared to OP and EOP analysis. WAVEOP analysis requires more number of iterations and simulation time for solution than OP and EOP analysis. The reason is lying around creating $\boldsymbol{N}_j^w$ block. Unlike $\boldsymbol{N}_j$ block in EOP analysis, $\boldsymbol{N}_j^w$ block in WAVEOP is not an incidence matrix but consists of elements that depends on circuit elements connected to external nodes of subcircuit and reference impedance. Specially for nonlinear circuits where nonlinear elements change at every iteration and hence this $\boldsymbol{N}_j$ matrix has to be rebuilt for each iteration. So, one iteration using waves is more expensive. Furthermore, writing code to construct such $\boldsymbol{N}_j$ blocks is complex. As subcircuit columns which belong to external nodes should be extracted and placed in $\boldsymbol{N}_j$ block depending on external currents with proper sign convention. However, the simulations presented here indicate that the concept is correct.

Chapter 4

Conclusion and Future Research

Techniques for parallel circuit analysis with emphasis in the formulation of equations for a circuit decomposed in subcircuit blocks have been reviewed and evaluated. For manually decomposed circuit two approaches to formulate circuit equations have been proposed and developed in this thesis. Both of them rely on a node-tearing formulation. In the first approach, nodal voltages and currents are exchanged between subcircuit blocks. This approach is not new but it has been developed independently in this thesis. This approach leads to interfacing vectors between various partitioned blocks, $\boldsymbol{N_j}$ and $\boldsymbol{N_j^T}$, with all entries '0', except for only one or more of the entries in them containing '±1' depending on external currents. This reduces some computation and communication cost among processors during parallel computation.

In second approach, a nodal formulation with waves (WAVEOP) is presented for the first time. In this analysis, each subcircuit iterates with incident waves received from another subcircuits and send waves back to the neighbour subcircuits. But WAVEOP analysis is not efficient compared to OP and EOP analysis. Because of the structure of $\boldsymbol{N_j^w}$ block, this analysis requires a greater number of iterations and simulation time for convergence than OP and EOP analysis. In WAVEOP, $\boldsymbol{N_j^w}$ block is not an incidence matrix but consists of elements related to the subcircuit components connected to external nodes of subcircuit. For nonlinear circuits nonlinear elements change at every iteration and hence this $\boldsymbol{N_j^w}$ matrix has to be rebuilt for each Newton iteration. So, one iteration using waves is more expensive compared to OP and EOP analysis. Furthermore, writing code to construct such $\boldsymbol{N_j^w}$ block is complex. Subcircuit columns belonging to external nodes must be extracted and placed in $\boldsymbol{N_j^w}$ block depending on external currents with proper sign convention.

Both formulations have been implemented in a general circuit simulator (EOP and WAVEOP analyses). Currently both implementations use serial code. In this case, each partition is solved serially, and synchronized later on to obtain the solution of the original circuit at each NR

iteration. EOP and WAVEOP do not yield speed-ups compared to regular OP analysis which simulates the original circuit without dividing equations in blocks. However, if parallel version of the EOP analysis is implemented, a significant speed-up could be achieved.

Suggestions for the Future Work

The main pending issue for this work is to implement a parallel version of the proposed algorithms. After that is achieved, a number of research directions will be open for exploration. One such direction would be to investigate solving nonlinear equations using a combination of fixed-point wave relaxation and Newton method using nodal variables. In this approach, subcircuits are iterated with relaxation method using waves for few iterations [32] at the beginning of the simulation and once it gets close to the solution, circuit decomposition based on nodal variables should be adopted. Another aspect is to implement an optional multilevel Newton algorithm [24] in the EOP analysis, in which each subcircuit is iterated for a fixed number of iterations. This may reduce the total number of global Newton iterations and therefore achieve a simulation speed-up. The EOP analysis could also be combined with waveform relaxation [30]. Finally, to make the parallel analysis practical, an automatic partitioning algorithm must be studied and implemented. Running relaxation method initially gets a good initial guess for EOP analysis. We expect that convergence would be faster.

Appendix A

The netlist of source codes of EOP and WAVEOP analysis and circuits used for simulation results can be obtained from:

```
git clone git://github.com/cechrist/cardoon.git
```

The netlist of the linear circuit used to explain complete example is provided below. The source codes of EOP and WAVEOP analysis can be found in following repository: src/cardoon/analysis and all netlists of circuits can be found from : src/cardoon/workspace/tapan.

Linear Circuit example

```
*** OP analysis ***
#.options maxdelta=50. maxiter= 100
#.analysis op

*** EOP analysis ***
.analysis eop

*** WAVEOP analysis ***
#.analysis waveop z0=10

*** Subcircuit instantiations***
x1 2 subcircuit1
x2 2 3 subcircuit2
x3 3 subcircuit3

*** Subcircuit definitions***

.subckt subcircuit1   2          # 2 is external node of Subcircuit 1 (Figure 3.10)

*** Element lines***

res:r1  1 gnd   r=10
res:r2 1 2 r=10
vdc:vdd 1 0 vdc=2
res:r3 2 0 r=10

.ends
```

```
.subckt  subcircuit2   2  3        #2 and 3 are external nodes of Subcircuit 2 (Figure 3.10)
res:r12 2 3 r=10
res:r4 3 4 r=10
res:r7 4 0 r=10
res:r6 5 0 r=10
res:r5 3 5 r=10
.ends

.subckt  subcircuit3    3          #3 is external node of Subcircuit 3  (Figure 3.10)
res:r8 3 6 r=10
res:r9 6 0 r=10
res:r10 6 7 r=10
res:r11 7 0 r=10
.ends

.end
```

Bibliography

[1] L. W. Nagel, SPICE2: A Computer Program to Simulate Semiconductor Circuits, CA, USA: University of California, Berkeley, 1975.

[2] U. Wever and Q. Zheng, "Parallel Transient Analysis for Circuit Simulation," in *IEEE Hawaii International Conference on System Sciences*, pp. 442-447, January 1996.

[3] J. Vlach and K. Singhal, Comput. Methods for Circuit Analysis and Design 2nd ed., Boston, USA: Kluwer, 1993.

[4] M. Chang and I. Hajj, "iPRIDE: A parallel integrated circuit simulator," *IEEE International Conference on Computer-Aided Design of Integrated Circuits and Systems*, pp. 304-307, November 1988.

[5] P. F. Cox, R. G. Burch, D. E. Hocevar, P. Yang and B. D. Epler, "Direct Circuit Simulation Algorithms for Parallel Processing [VLSI]," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10. no. 6, pp. 714-725, June 1991.

[6] J. T. Deutschand and A. R. Newton, "A Multiprocessor Implementation of Relaxation-based Electrical Circuit Simulation," in *21st Conference on Design Automation*, pp. 350-357, June 1984.

[7] G. K. Jacob, A. R. Newton and D. O. Pederson, "An Empirical Analysis of the Performance of a Multiprocessor-based Circuit Simulator," in *23rd Conference on Design Automation*, pp. 588-593, June 1986.

[8] R. A. Saleh, K. A. Gallivan, M. C. Chang, I. N. Haji, Smart, D. and T. N. Trick, "Parallel Circuit Simulation on Supercomputers," in *Proceedings of the IEEE*, vol. 77, no. 12, pp. 1915-1931, December 1989.

[9] C. P. Yuan, R. Lucas, P. Chan and R. Dutton, "Parallel Electronic Circuit Simulation on the IPSC System," in *IEEE Custom Integrated Circuits Conference*, pp. 6.5/1-6.5/4, May 1988.

[10] U. Gajanan, M. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar and H. Park, "“An 8-core 64-thread 64b Power-efficient SPARC, SoC," *IEEE International Solid-State Circuits Conference (ISSCC),* pp. 108-109, February 2007.

[11] N. A. Kurd, S. Bhamidipati, C. Mozak, J. L. Miller, T. M. Wilson, M. Nemani and M. Chowdhury, "Westmere: A family of 32nm IA Processors," in *in IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 96-97, February 2010.

[12] C. McNairy and R. Bhatia, "Montecito: A Dual-Core, Dual-Thread Itanium Processor," in *IEEE Micro*, vol. 25, no. 1, pp. 10-20, March 2005.

[13] D. F. Wendel, R. Kalla, J. Warnock, R. Cargnoni, S. Chu, J. Clabes, D. Dreps, D. Hrusecky, J. Friedrich, S. Islam, J. Kahle, J. Leenstra, G. Mittal, J. Paredes, J. Pille, P. Restle, B. Sinharoy, G. Smith, W. Starke, S. Taylor, J. Van Norstrand, S. Weitzel, P. Williams and V. Zyuban, "Power7TM: A Highly Parallel, Ccalable Multi-core High End Server Processor," *IEEE Journal of Solid-State Circuits,* vol. 46, no. 1, pp. 145-161, January 2011.

[14] D. C. E. Christoffersen, "Cardoon Simulator," [Online].
Available: http://vision.lakeheadu.ca/cardoon/.

[15] P. J. Rodrigues, Computer-aided analysis of onlinear microwave circuits, London: Artech House Publishers, 1998.

[16] G. Kron, Tensor Analysis of Networks, New York, 1939.

[17] G. Kron, Diakoptics, The Piecewise Solution of Large-Scale Systems, London: MacDonald & Co., 1963.

[18] A. Brameller, M. N. John and M. R. Scott, "Practical Diakoptics for Electrical Networks," London, Chapman & Hall, 1969.

[19] B. Smith, P. Bjørstad and W. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 2004.

[20] Y. Saad, Iterative methods for sparse linear systems, SIAM, 2003.

[21] N. B. Rabbat, A. L. Sangiovanni- Vincentelli and H. Y. Hsieh, "A Multilevel Newton Algorithm with Macromodeling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain," in *IEEE Transactions on Circuit and Systems*, vol. cas-26, no. 9, pp. 733-741, September 1979.

[22] M. Heydemann, in *Functional Macromodeling of Electrical Circuits*, ibid., pp. 532-535.

[23] N. Frohlich, B. M. Riess, U. A. Wever and Q. Zheng, "A New Approach for Parallel Simulation of VLSI Circuits on a Transistor Level," in *IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications*, vol. 45, no. 6, pp. 601-613, June 1998.

[24] M. Honkala, J. Ross and M. Valtonen, "New Multilevel Newton-Raphson Method for Parallel Circuit Simulation," in *European Conference on Circuit Theory and Design*, Espoo, Finland, pp. 113-116, Aug 28-31, 2001.

[25] D. Paul, M. S. Nakhla, R. Achar and N. M. Nakhla, "Parallel Circuit Simulation via Binary Link Formulations (PvB)," *IEEE Transactions on Components, Packaging and Manufacturing Technology,* vol. 3, no. 5, May 2013.

[26] R. A. Rohrer, "Circuit partitioning simplified," in *IEEE Trans. Circuit System*, pp. 2-5, January 1988.

[27] G. Karypis, "hMETIS," 1998. [Online].
Available: http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview.

[28] N. Selvakkumaran and G. Karypis, "Multiobjective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization," vol. 26, no. 3, pp. 504-517, March 2006.

[29] S. Priyadarshi, C. S. Saunders, N. M. Kriplani, H. Demircioglu, W. R. Davis, P. D. Franzon and M. B. Steer, "Parallel Transient Simulation of Multiphysics Circuits Using Delay-Based Partitioning," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 10, pp. 1522-1535, October 2012.

[30] E. Lelarasmee, A. E. Ruehli and A. L. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time-domain Analysis of Large Scale Integrated Circuits," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. cad-1, no. 3, pp. 131-145, July 1982.

[31] X. Ye, P. Li and S. Nassif, "Hierarchical Multialgorithm Parallel Circuit Simulation," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 45-58, January 2011.

[32] M. Kabir, C. Christoffersen and N. Kriplani, "Transient Simulation Based on State Variables and Waves," *Inernational Journal of RF and Microwave Computer-Aided Engineering,* vols. 21, no. 3, pp. 314-324, May 2011.

[33] X. Chen and Y. Wang, "NICSLU: An Adaptive Sparse Matrix Solver For Parallel Circuit Simulation," in *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 2, pp. 261-274, February 2013.

[34] C.W. Ho, A. E. Ruehli and P. A. Brennan, "The Modified Nodal Approach to Network Analysis," in *IEEE Transactions on Circuits and Systems,* vol. cas-22*,* no. 6, pp. 504-509, June 1975.

[35] "NumPy," [Online]. Available: http://www.numpy.org/.

[36] "pycppad-20121020: A Python Algorithm Derivative Package," [Online]. Available: http://www.seanet.com/~bradbell/pycppad/pycppad.xml.

[37] [Online]. Available: http://www.scipy.org/ .

[38] [Online]. Available: http://ipython.org/.

[39] D. M. Pozar, in *Microwave Engneering*, United States of America, 1998.

[40] J. Vlach and K. Singhal, Computer Methods for Circuit Analysis and Design, 2nd ed., Boston, USA: Kluwer, 1993.

[41] C. E. Christoffersen, "Transient Analysis of Nonlinear Circuits Based on Waves," in *Proceeedings of the 7th International Conference on Scientific Computing in Electrical Engineering (SCEE 2008)*, Helsinki Institute of Technology, Finland, pp. 1-2, September 28 - October 3 2008.

[42] R. A. ROHRER, "Circuit Partitioning Simplified," in *IEEE Transactions on Circuits and System*, vol. 35, no. 1, pp. 2-5, January 1988.

[43] Y. W. Xiaoming Chen, "NICSLU: An Adaptive Sparse Matrix Solver For Parallel Circuit Simulation," in *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 2, pp. 261-274, February 2013.

[44] C. E. Christoffersen and M. B. Steer, "Implementation of the Local Reference Node Concept for Spatially Distributed Circuits," in *Inernational Journal of RF and Microwave Computer-Aided Engineering*, vol. 9, no. 5, pp. 376-384, July 199.

[45] P. Gray, P. Hurst, S. Lewis and R. Meyer, in *Analysis and Design of Analog Integrated Circuits*, John Wiley & Sons, Inc., 2009.