POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Accelerated Simulation of Large Scale Power System Transients

ANAS ABUSALAH

Département de Génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*

Génie électrique

Avril 2019

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

Accelerated Simulation of Large Scale Power System Transients

présentée par **Anas ABUSALAH**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*

a été dûment acceptée par le jury d'examen constitué de :

Houshang KARIMI, président

Jean MAHSEREDJIAN, directeur de recherche

Ilhan KOCAR, membre et codirecteur de recherche

Omar SAAD, membre et codirecteur de recherche

Sébastien DENNETIÈRE, membre

Mario PAOLONE, membre externe

DEDICATION

To my parents, wife, son and siblings…

ACKNOWLEDGEMENTS

To my main supervisor Prof. Jean Mahseredjian for all the support and encouragements throughout my PhD studies. This work won't be possible without your supervision and input, I would like to thank you for allowing me to work under your supervision and giving me the trust and friendship during the past four years.

To my co-supervisor Mr. Omar Saad, your support and dedication kept me going at times of despair. I want to thank you from the bottom of my heart for all the help you offered me and without your hands on experience this work will not be the way it is now.

To my co-supervisor Prof. Ilhan Kocar, I would like to thank you for all the discussion and advices your offered me during my studies.

To all organizations involved in this project EDF, IREQ, OPAL-RT, Polytechnique Montreal and RTE for giving me the opportunity to work on this challenging project and for all financial support you offered me during the past four years.

To all my friends and colleagues whom I met at Polytechnique Montreal during my studies, to all students, post docs and research associates. Thank you all for all special moments we spent together in the lab and for being true friends.

To my brothers Issam and Mohammad and my sisters Lama and Shurouq for encouraging me and keeping me in your thoughts throughout my PhD studies.

To my wife and son who have been so patient along this journey, I would like to thank you from the bottom of my heat for being there for me whenever I needed any kind of support.

To my parents, My father Ahmad Abusalah, your advice, guidance and words were inspiring to me and I only dream of one day being as great and you are. My mother Wafaa Mousa, you raised me with the entire world kindness and love, and I owe it all to you for being the person I am today. I want to thank you both for keeping me in your prayers and never letting me down at any time.

RÉSUMÉ

Le temps de simulation est un paramètre crucial de l'analyse des transitoires dans les réseaux électriques et il est en train de devenir l'un des facteurs les plus importants pour mesurer les performances et la fiabilité des logiciels. Actuellement, la vitesse et les performances des processeurs ont atteint un point où l'accélération de gain en vitesse et d'opérations en virgule flottante peut être réduite en se concentrant uniquement sur l'aspect vitesse des processeurs individuels. Au contraire, la recherche en informatique et le développement de matériel informatique tendent de plus en plus à rendre les processeurs parallèles plutôt que plus rapides. D'autre part, la simulation des systèmes électriques devient de plus en plus complexe avec l'introduction de modèles complexes tels que les énergies renouvelables, les composantes de réseaux intelligents et l'électronique de puissance. En outre, la demande de puissance sans cesse croissante et l'augmentation de la zone de couverture des réseaux de distribution d'énergie contribuent à l'augmentation de la taille des réseaux de distribution d'énergie et ralentissent encore plus la simulation électromagnétique transitoire de ces réseaux.

De nombreux -logiciels de simulation de type EMT effectuent actuellement leurs opérations de manière séquentielle en utilisant un seul - processeur, plutôt que tous les processeurs de la machine. Ce comportement entraîne un temps de simulation long et introduit des difficultés pour simuler des réseaux de systèmes d'alimentation plus avancés et complexes. Ce type de délai devient un obstacle lorsque de grands réseaux, réels ou existants, sont utilisés. Par exemple, simuler le réseau d'Hydro-Québec doté d'une matrice de taille 41555 × 41555 et contenant un grand nombre de dispositifs de commutation et des éléments non linéaires nécessite 1765 secondes pour simuler une seconde avec un pas de temps de 50us.

La programmation parallèle multithread est maintenant disponible dans les compilateurs modernes. Elle peut être utilisée pour améliorer de manière significative les performances des calculs EMT. La recherche actuelles dans ce domaine est principalement appliqué à des systèmes moins complexes qui nécessitent l'intervention de l'utilisateur pour le découpage parallèle et manque de généralisation pour toute topologie rencontrée dans les études réels. Cette thèse développe une méthode de parallélisation entièrement automatique applicable aux systèmes à grande échelle avec des topologies arbitraires sans aucune intervention de l'utilisateur.

Cette thèse présente les avancées existantes dans le domaine de l'accélération de la simulation des transitoires électromagnétiques et met en évidence les différentes approches adoptées pour obtenir une simulation plus rapide de l'EMT. L'accent est principalement mis sur le threading à travers le processeur exclusivement sur les ordinateurs de bureau modernes utilisés quotidiennement par les ingénieurs.

Ce document portera principalement sur le threading exclusivement via le processeur. Dans cette thèse, deux approches sont adoptées pour améliorer les performances et le temps de calcul de la simulation EMT. La première approche est axée sur la recherche d'un solveur simple, rapide et efficace, qui servira de base à ce travail de recherche. Ce solveur est entièrement étudié et personnalisé pour éviter tout calcul inutile qui n'est pas nécessaire pour les simulations de type EMT. Différents solveurs linéaires de matrices creuses sont considérés dans cette thèse. Ces solveurs sont traditionnellement divisés en deux catégories, les solveurs directs et itératifs. Dans cette étude, l'accent sera mis sur la sélection du meilleur solveur direct parmi KLU et SuperLU deux solveurs basés sur l'utilisation de l'ordonnancement de degrés minimum,.

La deuxième approche pour obtenir une accélération de la simulation EMT consiste à appliquer une technique de calcul parallèle au processus de simulation et à permettre à différentes tâches d'être résolues en parallèle sur différents processeurs. De nombreuses techniques de parallélisation sont étudiées pour trouver la plus performante avec le moins de modifications possibles du code du solveur et exigeany le moin de temps d'implémentation . De nombreux standards de programmation multithreading sont pris en compte, tels que le multithreading C ++ 11 et le standard OpenMP.

Le nouveau solutionneur proposé (SMPEMT) est validé et testé sur un large éventail de points de repère. Cette validation est effectuée à l'aide du logiciel de simulation EMT EMTP-RV en tant que support de test. Tous les résultats des tests SMPEMT sont comparés aux résultats de l'EMTP et la vitesse de la simulation et le gain d'accélération sont également vérifiés.

ABSTRACT

Simulation time is a crucial parameter in power system transient analysis. The simulation needs for electromagnetic transients are continuously increasing. The electromagnetic transient (EMT) type tools are now also used for the simulation of slower electromechanical transients in large scale power systems. The EMT approach for power system analysis is the most accurate approach, but it suffers from computation performance issues. Research on this aspect is currently of crucial importance. Research is timely and should increase the application range of EMT-type tools. In fact very fast EMT-type tools can have a major impact on the simulation and analysis of modern power grids with increased penetration of renewables.

Currently, computer processor speed and performance reached a point where not much speed gain and floating-point operation acceleration can be achieved by only focusing on the speed aspect of individual processors. Rather, the trend in computer research and hardware development is becoming more and more focused on making processors parallel rather than faster.

Many EMT-Type simulation packages currently perform their operations sequentially by using only one CPU core rather than all machine processors. This behaviour results in long simulation time and introduces major difficulties when simulating large and complex power grids. This type of delay becomes a show stopper when large, real and existing networks are used.

Multithreaded parallel programming is now available in modern compilers. It can be used to significantly improve the performance of EMT computations.

Current research in this field has been mostly applied to less complicated systems and requires user intervention. This thesis develops a fully automatic parallelization method that is applicable to large scale systems with arbitrary topologies.

This PhD thesis presents existing progress in the field of electromagnetic transient simulation acceleration and highlights the different approaches that are adopted to achieve faster EMT simulation. The focus is mainly on threading through CPU exclusively on modern desktop computers used by engineers on daily basis.

In this thesis, two approaches are adopted to improve EMT simulation performance and computation time. The first approach is focused on finding a sparse solver that is fast and efficient

to act as a baseline for all computations. This solver is studied throughout and customized to improve performance for EMT computation needs.

The second approach to achieve acceleration is by applying parallel computation techniques on the computation process and allow different tasks to be solved in parallel on different processors. Parallelization techniques are studied to find the best performing parallelization technique with the least changes to the solver code and minimum implementation time.

The outcome of research is a new parallel solver, named SMPEMT. It is demonstrated and tested on practical large-scale benchmarks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

EMT: Electromagnetic transient

MANA: Modified augmented nodal analysis

EMTP: Electromagnetic transient program

BTF: Block triangular form

BDF: Block diagonal form

AMD: Approximate Minimum Degree

COLAMD: Column Approximate Minimum Degree

$\mathbf{A}$: Matrix with no permutation

$\hat{\mathbf{A}}$: Matrix with BTF permutation

CPU: Central processing unit

GPU: Graphical processing unit

TVMs: time-varying models

ITVM: iterative time-varying method

CSC: Compressed Column Format

CSR: Compressed Raw Format

NNZ: non-zero element

NE: Network equations

NM: Non-linear models

TVM: Time-varying models

BDF: Block diagonal Format

HQ: Hydro-Quebec

MDO: Minimum degree ordering

FLCC: First left changed column

FLDC: First left dynamic column

SMPEMT: Sparse matrix package for EMTs

IBP: In block permutation

NFPO: Number of floating-point operations

DFS: Depth first search

KLU-FF: KLU Full Factorization technique

KLU-RF: KLU Re-Factorization technique

CHAPTER 1 INTRODUCTION

The circuit based electromagnetic transient (EMT) simulation approach is a powerful approach for studying power transmission and distribution grids. The range of applications of EMT-type tools varies from very fast transients to slower electromechanical transients. Typical studies include switching transients, lighting transients, HVDC transmission, wind generation and electromechanical transients from small to very large-scale systems. EMT simulation is also used in the design and sizing of power network components such as insulation levels and energy absorption capabilities. EMT-type simulation tools are subdivided into two main categories: off-line and real-time. The main goal of performing off-line is to perform simulations on generic computers that are easily available to engineers. Real-time simulation tools are capable of generating results in synchronism with a real-time clock. Such tools have the advantage of being capable of interfacing with physical devices and maintaining data exchanges within the real-time clock. The capability to compute and interface within real-time, imposes important restrictions on the design of such tools. Current off-line EMT-type simulation tools remain more accurate than the real-time counterparts. They are also capable of solving much larger power grids and maintain higher accuracy. Nevertheless, research on the acceleration of off-line tools is also applicable to the eventual acceleration of real-time tools. Convergence of these tools into a single environment is inevitable in the near future.

Instead of using EMT-type tools in time-domain, it is also possible to simulate large power grids through phasor-domain computations. Phasor-domain tools are also referred to as transient stability (TS) tools. The TS approach can be very fast, especially when solving very large-scale systems, but it suffers from important accuracy issues. This is becoming nowadays an important issue with the increased usage of power electronics-based components (wind generation, HVDC, photovoltaics...) in modern power systems. In fact, in more and more applications, the much more accurate EMT-type methods and models are called to replace the usage of TS-type simulation and modeling. This trend will subsist, and EMT-type tools will receive wider and wider acceptance in practical applications, especially when they become capable of much higher efficiency for networks of very large dimensions.

This thesis presents the implementation of a parallel sparse matrix solver used for improving the computational speed of EMT-type tools. The new approach contributes in enhancing the overall

quality of EMT simulation by reducing the simulation time while maintaining the simulation accuracy and reliability. Unlike other solvers published in the literature that are demonstrated by repeating a small network multiple number of times, the proposed approach can be generalized and is valid on any power system network. The proposed new method is also capable of automatically parallelizing networks of arbitrary topologies without any user intervention.

The new method presented in this thesis is based on the KLU sparse matrix solver which is currently the most suitable for circuit-based simulation methods [1]. The solver is programmed using parallelization algorithm that can automatically detect independent parts of the sparse matrix separated by the natural decoupling available in transmission line/cable models. This decoupling technique can be detected without any user intervention and pre-determination of different subnetworks.

Due to the iterative process required for solving nonlinearities in various models, this thesis also contributes modifications into the KLU solver for improving its performance when repetitive matrix refactorizations are requested.

The proposed new approach is demonstrated using an EMT-type software (i.e EMTP) that uses a fully iterative solution method for all nonlinear models [2]. It remains however applicable to any EMT-type software tool that uses sparse matrices. A modular sparse matrix package can be replaced easily by the package elaborated in this thesis.

1.1 Thesis Outline

This thesis is divided into four chapters that are summarized below.

Chapter 1: Introduction

This chapter introduces the concept of EMT simulation and the modified-augmented nodal analysis (MANA) approach used in the EMTP simulation package to form its sparse matrix [3]. This approach is explained in detail and illustrated with an example. In addition, different sparse solvers are introduced in this chapter including the minimum degree ordering (MDO) based approach used in EMTP [4]. These solvers are used and compared to select the fastest package and enhance it as it will be demonstrated in the following sections.

In the second part of this chapter, different methods such as BTF, MDO, SSN and Compensation theory are introduced as well.

The last part of this chapter discusses different threading algorithms used in implemented the multi-threaded sparse solver used in this thesis.

Chapter 2: Implementation of Sparse Matrix Package for EMTs

In this chapter the approaches used to accelerate the simulation process are explained and the implementation of a new sparse solver that is customized only for EMT-type simulations is introduced and explained. In addition, a comparison between the new sparse solver and other already existing ones is presented and discussed.

Chapter3: Test Results

In this chapter, different benchmarks used in the process of validating the new sparse solver will be presented. These test cases consist of real and existing networks with complex models, including nonlinearities and power-electronics converters for wind generator applications. Each network's topology is described with related matrices and complexity level. Computational timings are used to demonstrate the advantages of the approach presented in this thesis.

The results of each test case are analyzed and studied. The acceleration rate (gain) for each case will be looked at in depth and compared with other cases. Observations and limitations will be address herein as well.

Chapter 4: Conclusion and Future work

This chapter provides a quick summary of the overall work done throughout this PhD work and it highlights the main milestones that were achieved during this project. In addition, it provides recommended future work.

1.2 Contributions

In this thesis the multithreading approach used for programming a parallel sparse solver is based on the OpenMP standard and the use of distributed memory design. Thanks to this design an efficient parallel solution is achieved, and the effect of overhead timings is kept at minimum. This parallel model design minimizes shared memory between different threads and allows each thread to store its own data on its own designated memory. This approach makes it easier for all threads to fetch and write data to memory without the need to communicate with the master thread or any other threads for that matter.

Another noticeable contribution in this thesis is the fact that the proposed method is tested on realistic large-scale network benchmarks. Parallelization is achieved without any user intervention. Such practical networks allow to derive more realistic conclusions on the potential gains in EMT-type solver parallelization.

1.3 Literature review

The computing time reduction for the simulation of electromagnetic transients[2][5] (EMTs) is a crucial research topic. The EMT-type[5] simulation methods are circuit based and can use very accurate models for an extended frequency range of power system phenomena. This qualifies them as being of wideband type. In fact, the EMT approach is applicable to both slower electromechanical transients and much faster electromagnetic transients. The computation of electromechanical transients can be achieved with EMT-type solvers for very large networks [6] and requires significant computing time when compared to phasor-domain approaches, but even for smaller networks, the computing time can become a key factor due to numerical integration time-step constraints or model complexity level. More and more challenging simulation cases are created for studying modern power systems, those include, for example, HVDC systems and wind generation[7].

There are several techniques for improving computational performance in EMT-type solvers. Such techniques include improvements in model performance using, for example, average-value models[8] for power-electronics based systems or circuit reduction [9]. Network reduction can be also achieved using frequency domain fitting[10], or through dynamic equivalents[11]. Other approaches include usage of multiple time-steps[12], waveform relaxation [13] and combinations of different methods [14]. An important problem in network solution parallelization methods, such as[15], is that user intervention is required for setting the network separation locations and task scheduling. The user should be aware of the case details in order to best allocate the separation locations and optimize the performance of the parallel solution. It is also necessary to program network topology analysis and, in some methods such as in [16], analysis can be used for automatic task scheduling.

A more direct path towards computational speed improvement in EMT-type numerical methods is through efficient sparse matrix solvers and parallelization. This chapter introduces different types of sparse solvers used in general circuit analysis. These solvers are currently implemented in different simulation tools and each has its own advantages and disadvantages. Moreover, parallel computation concept will be discussed, and different parallel programming techniques will be introduced. These techniques will be used in this thesis to implement the EMT customized sparse

solver. In addition, different ordering techniques will be discussed such as AMD, COLAMD, BTF and METIS.

This work targets off-line simulation methods and presents CPU-based parallelization for conventional multi-core computers using a sparse matrix solver, named KLU [1].

1.3.1 Modified-Augmented-Nodal Analysis (MANA)

The modified-augmented-nodal analysis (MANA) method is briefly recalled in this section.

The traditional approach for the formulation of main network equation is based on nodal analysis. The network admittance matrix $\mathbf{Y_n}$ is used for computing the sum of currents entering each electrical node and the following equation results from classical nodal analysis.

$$\mathbf{Y_n v_n = i_n} \tag{1.1}$$

where, $\mathbf{v_n}$ is the vector of node voltages and the members of $\mathbf{i_n}$ holds the sum of currents entering each node. It is assumed that the network has a ground node at zero voltage which is not included in (1.1) . Since the network may contain voltage sources (known node voltages), equation (1.1) must be normally partitioned to keep only the unknown voltages on the left hand side

$$\begin{bmatrix} \mathbf{Y_n'} & \mathbf{Y_{ns}} \\ \mathbf{Y_{sn}} & \mathbf{Y_{ss}} \end{bmatrix} \begin{bmatrix} \mathbf{v_n'} \\ \mathbf{v_s} \end{bmatrix} = \begin{bmatrix} \mathbf{i_n'} \\ \mathbf{i_s} \end{bmatrix} \tag{1.2}$$

$$\mathbf{Y_n' \, v_n' = i_n' - Y_{ns} \, v_s} \tag{1.3}$$

where $\mathbf{Y_n'}$ is the coefficient matrix of unknown node voltages $\mathbf{v_n'}$, $\mathbf{i_n'}$ holds the sum of currents entering nodes with unknown voltage, $\mathbf{Y_{ns}} \in \mathbf{Y_n}$ and relates to known voltages $\mathbf{v_s}$. It is noticed that $\mathbf{v_n} = \begin{bmatrix} \mathbf{v_n'} & \mathbf{v_s} \end{bmatrix}^{\mathbf{T}}$ [20].

Equation (1.3) has several limitations. It does not allow, for example, to model branch relations instead of nodal relations and it assumes that every network model has an admittance matrix representation, which is not possible in many cases. This is where the modified augmented nodal analysis comes into play. The MANA formulation method [20][21][22] is a relatively new approach to formulate network equations. This method offers several advantages [3] over classical

nodal analysis. Its formulation is recalled here to relate to material presented in the following sections. In MANA the system of equations is generic and can use different types of unknowns in addition to voltage. Equation (1.3) is augmented to include generic device equations and the complete system of network equations can be rewritten in the more generic form as seen in (1.4).

$$\mathbf{A_N}\,\mathbf{x_N} = \mathbf{b_N} \tag{1.4}$$

In this equation $\mathbf{Y_n} \in \mathbf{A_N}$, $\mathbf{x_N}$ contains both unknown voltage and current quantities and $\mathbf{b_N}$ contains known current and voltage quantities. The matrix $\mathbf{A_N}$ is not necessarily symmetric and it is possible to directly accommodate non-symmetric model equations. Equation (1.4) can be written explicitly as

$$\begin{bmatrix} \mathbf{Y_n} & \mathbf{A_c} \\ \mathbf{A_r} & \mathbf{A_d} \end{bmatrix} \begin{bmatrix} \mathbf{v_n} \\ \mathbf{i_x} \end{bmatrix} = \begin{bmatrix} \mathbf{i_n} \\ \mathbf{v_x} \end{bmatrix} \tag{1.5}$$

where the matrices $\mathbf{A_r}$, $\mathbf{A_c}$ and $\mathbf{A_d}$ (augmented portion, row, column and diagonal coefficients) are used to enter network model equations which are not or cannot be included in $\mathbf{Y_n}$, $\mathbf{i_x}$ is the vector of unknown currents in device models, $\mathbf{v_x}$ is the vector of known voltages, $\mathbf{x_N} = \begin{bmatrix} \mathbf{v_n} & \mathbf{i_x} \end{bmatrix}^T$ and $\mathbf{b_N} = \begin{bmatrix} \mathbf{i_n} & \mathbf{v_x} \end{bmatrix}^T$ [23].

It is emphasized that the system in (1.5) is non-symmetric and can also accommodate generic equations, such as

$$k_1 v_k + k_2 v_m + k_3 i_x + k_4 i_y + ... = b_z \tag{1.6}$$

Where the terms on the left contribute coefficients (k_j) into the $\mathbf{A}$ matrix for voltage (v_j) and current (i_j) unknowns, and b_z is a cell in the $\mathbf{b}$ vector. This equation allows integrating directly source and switching equations. For example, an ideal switch can be represented by

$$k v_k - k v_m - k_s i_{km} = 0 \tag{1.7}$$

When the ideal switch is in closed position, $k = 1$ and $k_s = 0$. When the ideal switch is open, $k = 0$ and $k_s = 1$. It is also possible to model non-ideal switches by setting $k = 1$ and replacing k_s by

high and low resistance values. Other, models, such as ideal transformers with tap control can be easily accommodated [3].

Single phase and three-phase transformers can be built in EMTP using the ideal transformer unit shown in Figure 1.1. It consists of dependent voltage and current sources. The secondary branch equation is given by

$$v_{k_2} - v_{m_2} - g v_{k_1} + g v_{m_1} = 0 \tag{1.8}$$

Where, g is the transformation ratio. This equation contributes its own row into the matrix $\mathbf{A_r}$ whereas the matrix $\mathbf{A_c}$ contains the transposed version of that row. It is possible to extend to multiple secondary windings using parallel connected current sources on the primary side and series connected voltage sources on the secondary side. Leakage losses and the magnetization branch are added externally to the ideal transformer nodes.

Three-legged core-form transformer models or any other types can be included using coupled leakage matrices and magnetization branches.

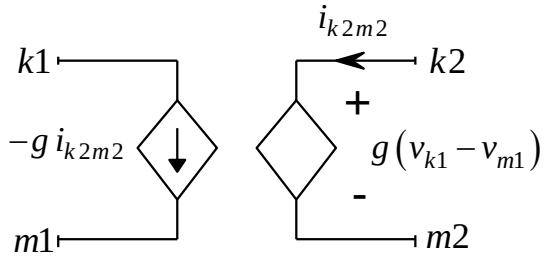

Figure 1.1 Ideal transformer model unit

The MANA formulation (1.4) is completely generic and can easily accommodate the juxtaposition of arbitrary component models in arbitrary network topologies with any number of wires and nodes. It is not limited to the usage of the unknown variables presented in (1.5) and can be augmented to include different types of unknown and known variables. The MANA formulation is conceptually simple to realize and program [23][24].

In order to provide a better understanding of the MANA formulation, the following example illustrates a simple circuit with its MANA formulation. Figure 1.2 illustrates the example circuit

structure with all its components names and ratings. The analysis of this circuit starts by forming the submatrix $\mathbf{Y_n}$ that contains the admittance matrix of the MANA main matrix.

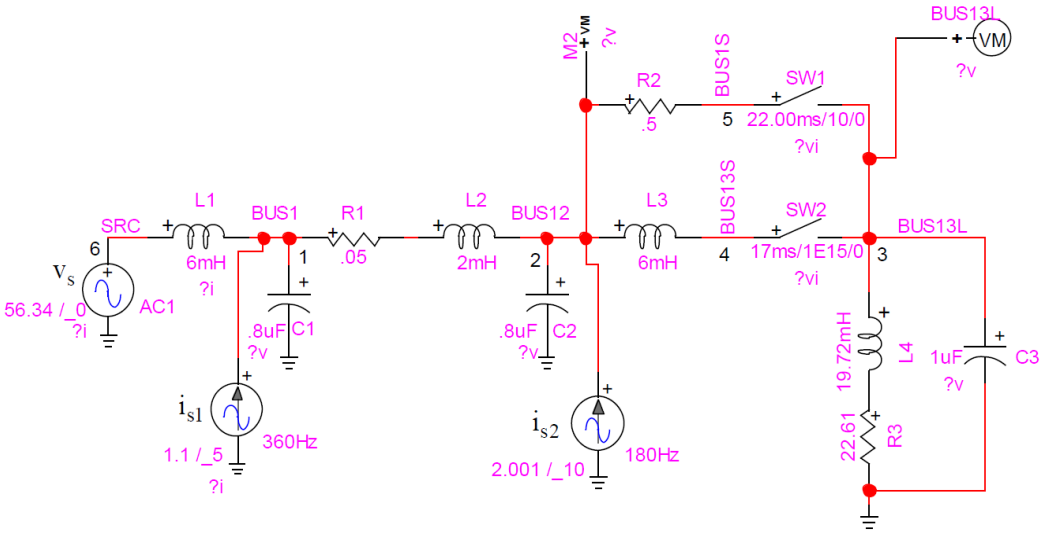

Figure 1.2 MANA Formulation Example

In order to find the time domain system of equations using the MANA formulation, the linear components of the circuit above (i.e inductors and capacitors) need to be discretized for a given integration time step Δt using a numerical integration method. Although any discretization rule can be used, the trapezoidal rule technique has been used herein to discretize nonlinear model of the circuit into linear representation. The inductor equation shown in (1.9) is discretized into a linear format shown in (1.10), this discretization allow to model the inductor as shown in Figure 1.3. Similarly, the capacitor equation can be discretized in a similar fashion as well.

$$v_{km} = L\frac{di_{km}}{dt} \tag{1.9}$$

$$i_{km_t} = \frac{\Delta t}{2L}v_{km_t} + \frac{\Delta t}{2L}v_{km_{t-\Delta t}} + i_{km_{t-\Delta t}} \tag{1.10}$$

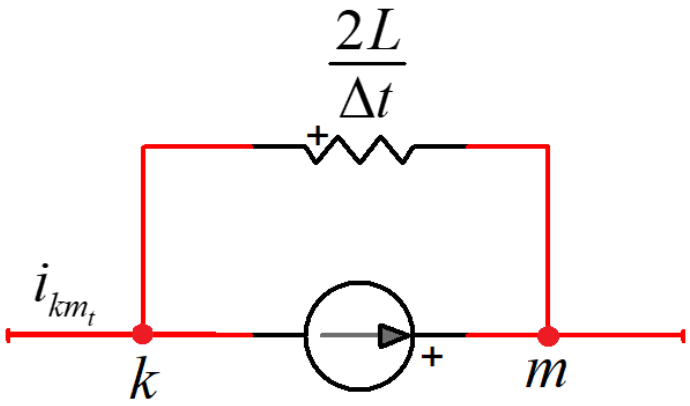

Figure 1.3 Discretized inductance model for time domain MANA solution

Equation (1.11) shows the MANA formulation of the circuit shown in Figure 1.2. The equation of each node has been written by replacing the inductors and capacitors connected to it by their discretized model. Node 1 for example (which represents BUS 1 in Figure 1.2), has L, C and RL branches connected to it in addition to the current source i_{s1}. Hence, replacing the L, C and RL branches with their discretized model produces the node junction shown in Figure 1.4 where R1, R3 and R4 represent the resistors in the discretized model of L1, C1 and L2 respectively.

$$\begin{bmatrix} y_{11} & y_{12} & 0 & 0 & 0 & y_{16} & 0 & 0 & 0 \\ y_{21} & y_{22} & 0 & y_{24} & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & y_{33} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & y_{42} & 0 & y_{44} & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ y_{61} & 0 & 0 & 0 & 0 & y_{66} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ i_{Vs} \\ i_{SW1} \\ i_{SW2} \end{bmatrix} = \begin{bmatrix} i_{s1}+i_{h61}-i_{h12}-i_{h10} \\ i_{s2}+i_{h12}-i_{h20}-i_{h24} \\ -i_{h30_1}-i_{h30_2} \\ i_{h24} \\ 0 \\ -i_{h61} \\ v_s \\ 0 \\ 0 \end{bmatrix} \quad (1.11)$$

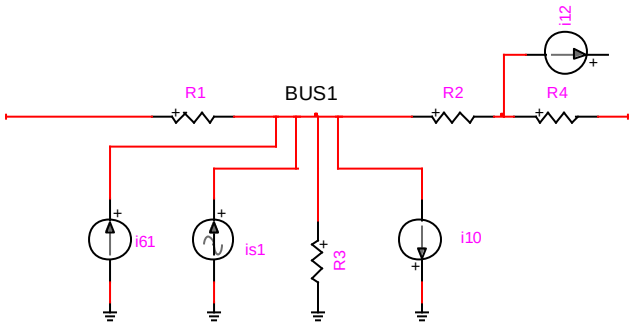

Figure 1.4 BUS1 branches discretized model

Hence, the value of y_{11} shown in equation (1.11) is the summation of R1, R2, R3 and R4 admittances that can be calculated using the following equation:

$$y_{11} = \frac{\Delta t}{2\times 6\times 10^{-3}} + \frac{2\times 0.8\times 10^{-6}}{\Delta t} + \frac{1}{0.05 + \dfrac{2\times 2\times 10^{-3}}{\Delta t}}$$

In a similar manner all quantities in the $\mathbf{Y_n}$ part of the MANA matrix can be found as follows:

$$y_{12} = -\frac{1}{0.05 + \dfrac{2\times 2\times 10^{-3}}{\Delta t}}$$

$$y_{16} = \frac{-\Delta t}{2\times 6\times 10^{-3}}$$

$$y_{22} = y_{11} + \frac{1}{0.5}$$

$$y_{21} = y_{12}$$

$$y_{24} = y_{42} = \frac{-\Delta t}{2\times 6\times 10^{-3}}$$

$$y_{33} = \frac{2\times 1\times 10^{-6}}{\Delta t} + \frac{1}{22.61 + \dfrac{2\times 19.72\times 10^{-3}}{\Delta t}}$$

$$y_{44} = -y_{24}$$

$y_{66} = -y_{16}$

$y_{61} = y_{16}$

The terms on the right of (1.11) are the contributions from independent current sources and history current sources resulting from component discretization (inductances and capacitances).

In the current project, EMT simulation is solved at each time step after updating $\mathbf{A}$ for switches position changes, transformer tap changes or any other modifications in model equations (including nonlinear devices). For nonlinear models (NMs), the NEs must be solved iteratively to achieve an accurate simultaneous solution. This is done by linearizing each model at each operating point and solving iteratively [3]. Model linearization results into a Norton equivalent with the Norton resistance contributing changes into the $\mathbf{A}$ matrix and the Norton current contributing updates into the $\mathbf{b}$ vector.

It means that at each time-point it is necessary to resolve (1.11) iteratively until convergence for all nonlinear models is achieved.

For time-varying models (TVMs), such as switches or transformer tap positions, it is also possible to update $\mathbf{A}$ iteratively without advancing to the next time-point. This accuracy option, marked as iterative time-varying method (ITVM), allows achieving a simultaneous solution for the determination of all changes and dependencies between models at the same time-point. This process also includes the sequential re-calculation of control system equations [3].

1.4 Parallelization and network tearing

In order to be able to solve power systems in parallel, the network system of equations needs to be subdivided into multiple subnetworks. This division process allows distributing different parts of the network on different CPU cores and solving these subnetworks independently. Several schemes can be used to achieve this goal. A list of known methods is presented in this section.

1.4.1 Block Triangular Format (BTF)

The block triangular formulation of a matrix is an approach specialized in permuting the matrix by putting as much non-zero elements of the matrix along the diagonal [25]. This reordering allows

the flexibility of partially decoupling the matrix into different submatrices and allows the solving of these submatrices separately. Figure 1.5 shows a generic representation of the block triangular format with blocks $\mathbf{A}_{ii}$ aligned along the diagonal and some off-diagonal elements $\mathbf{A}_{ij}$. These off diagonal blocks/elements arise due to light links between different parts of the matrix such as block $\mathbf{A}_{11}$ and $\mathbf{A}_{33}$ that are linked through $\mathbf{A}_{13}$ block, and $\mathbf{A}_{22}$ and $\mathbf{A}_{55}$ that are linked through $\mathbf{A}_{25}$ block. In this thesis all cases used have no off-diagonal elements in them thanks to the time domain decoupling produced by transmission lines. Although, the parallelization of matrices shown in Figure 1.5 is still feasible, it involves more restrictions and complications.

$$\mathbf{A}=\begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{25} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{33} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{44} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{55} \end{bmatrix}$$

Figure 1.5 An example of a matrix in BTF form

Another type of block triangular form that is more interesting for parallelization and mainly used herein, is the type where there are no off diagonal blocks/elements. This type of format is called block diagonal form BDF since all matrices elements are aligned along the matrix diagonal and the rest are zeros as can be seen in Figure 1.6 [25].

$$\mathbf{A}=\begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{33} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{44} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{55} \end{bmatrix}$$

Figure 1.6 An example of a matrix in BDF

In order to transform a matrix into the BTF form, the KLU package uses a special technique that is based on Duff and Reid's algorithm [25]. This algorithm finds any matrix BTF form (if applicable) by finding all strongly connected vertices of the matrix. It starts by preparing the matrix adjacency graph which guides the algorithm by moving from one graph vertex to another. Then a depth first search is launched starting from a random vertex and tries to visit/reach the maximum number of graph vertices of which there exists a path.

The design of KLU BTF algorithm uses a user-built stack that keeps track of all visited and unvisited vertices and avoids many run time errors such as stack over flow and memory shortage. The algorithm uses depth first search (DFS) topology that is based on a recursive algorithm to find all possible strongly connected vertices in the graph and keeps track of all visited and non-visited vertices. Once all connected vertices are labeled as visited, those vertices (nodes) form a block in the BTF form and the DFS algorithm begins again starting from an arbitrary non-visited vertex. The vertex graphs are explored and all efforts to try all combinations of connections is exhausted.

The following example gives a better visualization of how the BTF algorithm calculates strongly connected regions of the matrix. It is noticed here that the transmission line models are of distributed parameter type. In fact, any such model, either with constant parameters or with frequency dependent parameters, offers an important property for parallelization. The line (or cable) model provides a delay between its left (k-side)) and right (m-side) hand sides. This means that the k-side network can be solved completely independently from the m-side network without any approximation. This well-known property is the key ingredient used in this thesis for delivering parallelization. Nevertheless, the independent subnetworks created by transmission lines, must be found automatically.

The sparsity pattern of the matrix of the network shown in Figure 1.7 is presented in Figure 1.8, and the BTF version of this matrix is presented in Figure 1.9. The BTF method can automatically derive the block-diagonal (BD) without any user intervention as long as the case has at least one transmission line implemented in it.

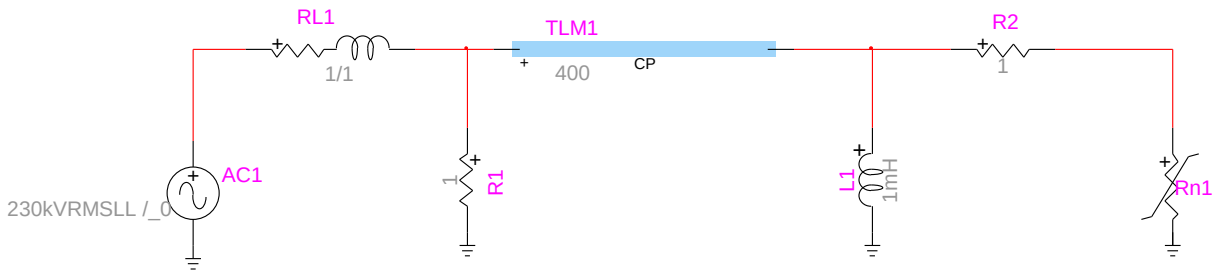

Figure 1.7 BTF format test case

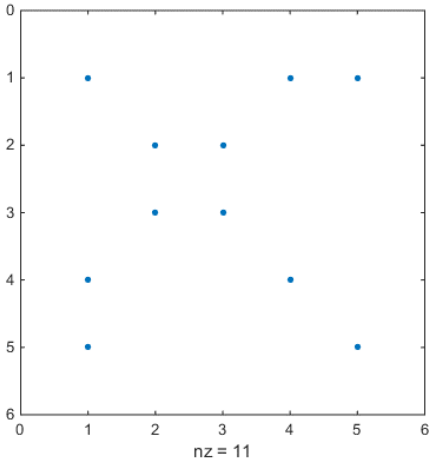

Figure 1.8 Sparsity pattern of matrix **A** of circuit shown in Figure 1.7

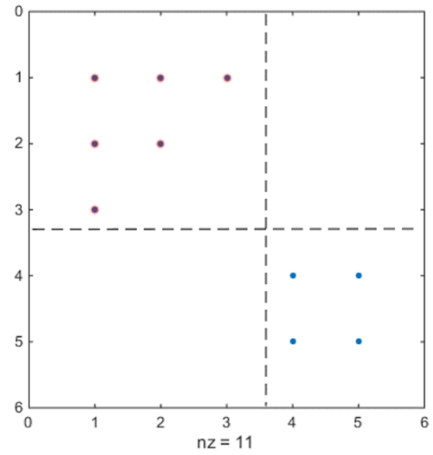

Figure 1.9 BTF Sparsity pattern of matrix **A** of circuit shown in Figure 1.7

In order to facilitate the differentiation between a BTF ordered matrix with a non-ordered matrix, the "hat" symbol is used from now on to represent all BTF ordered matrices (i.e $\hat{\mathbf{A}}$). In addition, the second digit in the BTF block index will be dropped due to the fact that all cases used herein have no off-diagonal blocks and both digits used to refer to a BTF block in this case are the same (i.e $\hat{\mathbf{A}}_i$ refers to block i in the BTF ordered matrix $\hat{\mathbf{A}}$). The BTF ordering is similar to graph traversal ordering that is based on a depth first algorithm to find all decoupled subnetworks and used in [16]. This ordering tries first to decouple the network based on the presence of the existing transmission lines and detect each subnetwork by the end of traversal, and at the same time they apply a heuristic calculation on the time cost for each component type (R, machine, inductance, etc...) in the subnetwork to make the simulation fit to real time simulation. Based on the execution cost, it can decide to join several subnetwork in one cpu and put this in one matrix if the resolution will fit in one step.

In KLU solver package, the BTF ordering is followed by another ordering that aims at reducing the $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ matrices fill-in. There are three ordering techniques that are already implemented in KLU package which are AMD, COLAMD and a user pre-defined ordering. This step plays a major role in reducing computational load during KLU numerical solution by reducing the number of floating-point operations required to solve the system. This type of ordering will be discussed in the coming sections.

1.4.2 METIS

METIS is an efficient algorithm that allows the partitioning of a matrix into multiple submatrices that are either independent of each other or share elements with other submatrices with all shared elements aligned along the submatrices boarders. An advantage of METIS is the feasibility of increasing the degree of parallelism with the existence of only one partitioned matrix in the system. METIS is specialized in partitioning large-scale irregular graphs or meshes and providing a permutation that provides an efficient partitioning as well as a reduction of fill-in of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ factors [26].

Unlike other traditional ordering techniques that work on the graph directly to provide a portioning by one step operation, and hence provide a low quality and less efficient partitioning, METIS is

based on multi-level graph partitioning technique that adopts a totally different technique that works on the graph and reduces the size of the graph as much as possible, by collapsing graph vertices and edges and partitioning the small graph and re-ordering it to produce the partitioning of the original graph [26]. Figure 1.10 shows matrix $\mathbf{A}$ of the IEEE1138 bus system ordered by METIS algorithm.

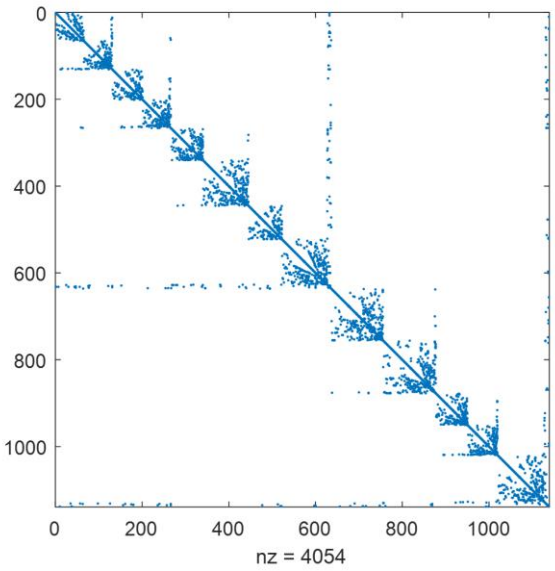

Figure 1.10 IEEE-1138 network ordered by METIS

Not only METIS can provide a high-quality portioning over other ordering techniques, it is considered one of the fastest ordering techniques that can provide its partitioning results in one or two orders faster that other traditional algorithms. Moreover, METIS ordering contributes in reducing the fill-in of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ factors without the need of using other techniques to do this task. METIS has the ability to produce more blocks along the diagonal compared to BTF. This phenomenon is due to the fact that METIS does not require a complete decoupling of blocks like BTF, but rather it can still reorder a block (that BTF was not able to partition) into sub-blocks and align all shared elements between these sub-blocks along the matrix or block border.

However, given the types of problems this thesis deals with, and the fact that BTF blocks are totally independent of each other, the use of METIS becomes less significant for cases that have multiple transmission lines that allow decoupling the case in time domain into relatively small independent regions. The importance of this ordering technique arises when a large-scale case with no

transmission line in its structure (or have very few of them) is being studied. Using Metis in this case helps introduce some degree of parallelization into the solution. In addition, this partitioning technique can help reduce the effect of large limiting blocks that prevent parallel simulation as will be seen in chapter 3. In addition, different fill-in reduction techniques that were tested herein (such as AMD) were found to be more efficient and produce around 15% less fill-in compared to METIS.

1.4.3 SSN and MANA

Another parallelization approach can be achieved through the combination nodal or MANA equations with state-space equations. This approach, name state-space nodal (SSN), is explained in [27]. The basic principle is that the network is separated (cutting) into state-space groups that are solved independently in parallel and combined through MANA equations.

Although the SSN method is perfectly accurate, it has two drawbacks. First the network separation locations must be determined manually. Another problem is that the usage of state-space equations is typically inappropriate for solving large scale grids. Other complications arise when the state-space equations must be reformulated for nonlinear models and time-varying models.

1.4.4 Scotch

Scotch [28] is yet another sparse matrix package that focuses on solving graph theory-based problems using divide and conquer approach. It is used in wide range of applications and not limited to electrical or power circuit problems, this package is based mainly on nested dissection approach to permute the application sparse matrix into a format that allows certain degree of parallelization. The nested dissection starts by forming the matrix undirected graph in which the vertices represent rows and columns of the matrix, and an edge/connection in the graph represents a nonzero entry in the sparse matrix. Once the graph is formed, the nested dissection algorithm uses a divide and conquer strategy on the graph in order to remove a set of vertices to result in two new graphs that are independent of each other. This algorithm uses a recursive technique that partitions the graph into subgraphs by selecting barriers or separators that consist of small set of graph vertices. The removal of these separators creates independent subgraphs. Applying factorization and solving the matrix parts that represent the new sub-graphs can be done

independently and in parallel. The results of the two new graphs can then be combined to find the overall matrix results.

In order to better understand the nested dissection ordering, the following matrix shown in Figure 1.11 gives an example of a matrix graph (mesh) that is ordered by nested dissection. In this figure, the graph is partitioned into four subgraphs (A, B, D and E) by three different separators (C, F and G). The matrix shown in equation (1.12) is a representation of the matrix after being reordered.

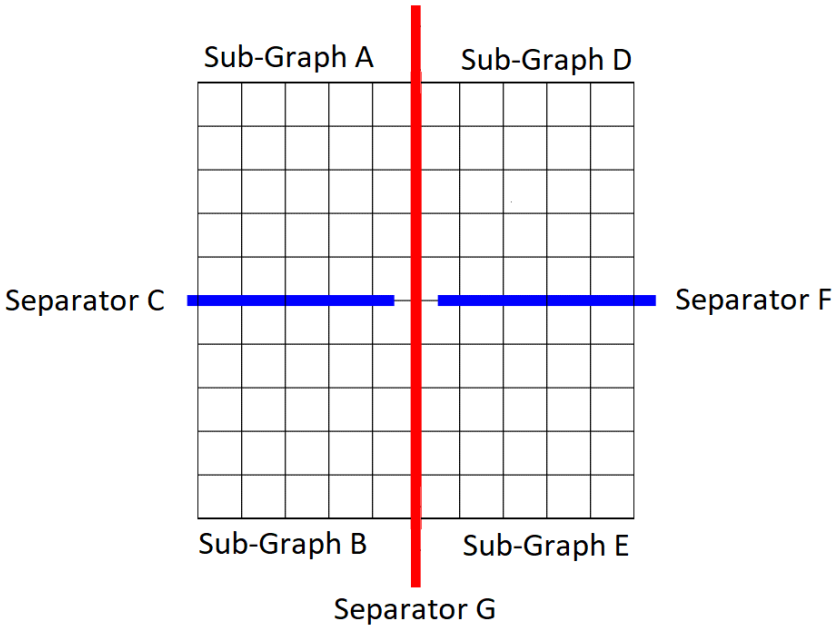

Figure 1.11 Scotch example - matrix graph

$$\mathbf{A}=\begin{bmatrix} S_{AA} & & S_{AC} & & & & S_{AG} \\ & S_{BB} & S_{BC} & & & & S_{BG} \\ S_{CA} & S_{CB} & S_{CC} & & & & S_{CG} \\ & & & S_{DD} & & S_{DF} & S_{DG} \\ & & & & S_{EE} & S_{EF} & S_{EG} \\ & & & S_{FD} & S_{FE} & S_{FF} & S_{FG} \\ S_{GA} & S_{GB} & S_{GC} & S_{GD} & S_{GE} & S_{GF} & S_{GG} \end{bmatrix} \quad (1.12)$$

Equation (1.12) shows the matrix after being ordered by nested dissection. All black elements in the matrix represent sub-graphs created after adding the separators, and all blue and red elements represent elements that are located across the separators (C, F and G) and they are linking different black blocks together.

Comparing this ordering technique with other ordering techniques, it is found that the nested dissection ordering is only applied to symmetric matrices, and that is a condition that can't be met and guaranteed in many EMT simulation tools including EMTP.

1.4.5 Bordered Block Diagonal matrix

This ordering scheme is a methodology that permutes the power system network matrix **A** into a doubly bordered block diagonal (DBBD) or a single bordered block diagonal (SBBD). Figure 1.12 shows the typical structure of a DBBD permuted matrix [29].

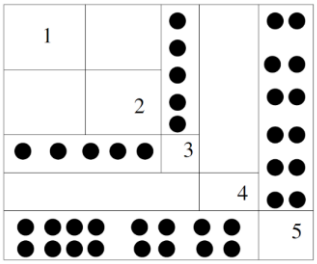

Figure 1.12 Doubly bordered block diagonal (DBBD)

It can be seen from Figure 1.12 that a doubly bordered block diagonal form is similar to a block upper triangular form but has non-zeros on the sub diagonal region. These nonzero elements found in the lower section of the diagonal form a horizontal strip resembling a border. The same thing applies for nonzero elements above the diagonal, these elements form a vertical strip that resemble a vertical border. Many ordering techniques can be used to produce DBBD permuted matrix such as METIS and nested dissection [26].

Generically speaking, when a complete network or a network portion does not contain delay-based transmission line models, it will not be possible to create a BD matrix for its equations. It can be demonstrated that for such cases, it is possible to derive a BBD matrix as seen in following compensation theory section.

1.4.6 Compensation Theory

The Compensation method theory is presented in [31]-[33] and it was used in [34]. The application described in [34] is for the solution of nonlinear models in an EMT-type code. The limitations that this method may encounter for solving nonlinearities, or in general, are described in [35]. It is

shown in [35] that the compensation method although very powerful, is not conformable to the topological proper-tree and therefore has topological limitations. The hybrid analysis method [36]-[38] has been shown in [40] to be more general than the Compensation algorithm. The work in [35][39] relates the more general hybrid analysis to the Compensation method.

Despite the limitations of the Compensation method for solving nonlinear systems, it will be used below to demonstrate how it links to other methods in the literature and how it can be used to decouple networks when transmission line delay-based decoupling is not possible.

The basic idea of the Compensation method is illustrated in Figure 1.13. In this figure the dash line shows cutting through wires. It is assumed that a linear or nonlinear network N2 is connected to network N1 through one or more wires. In some publications [34] it is assumed that N2 can contain only (one type of) nonlinear component, but N2 can actually contain any number of nonlinear components and it fact it can contain complete arbitrary (except for cases explained in [35]) networks. In the following, it will be assumed that N2 is actually a grid with any number of components. N2 can be purely linear.

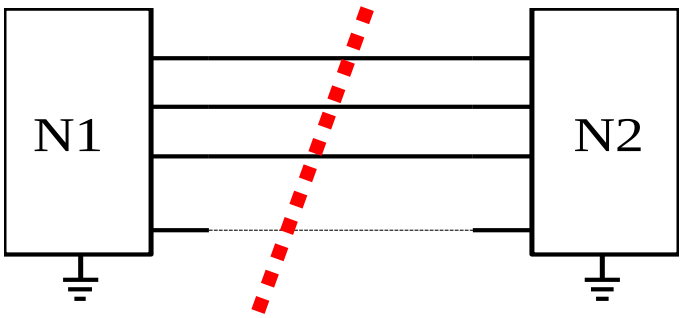

Figure 1.13 Separation of two networks using the compensation method

Let us assume the wires ($\hat{n}$ wires) connecting N2 to N1 are connected to a set of nodes $\hat{N}$ in N1. It can be written that for any time-point solution

$$\mathbf{v}_{\hat{N}}^{final} = \mathbf{v}_{\hat{N}} + \mathbf{v}_{\hat{N}_\phi} \tag{1.13}$$

where $\mathbf{v}_{\hat{N}}$ is the solution vector of node voltages for N1 when it is disconnected from N2, $\mathbf{v}_{\hat{N}_\phi}$ is the solution found from the contributions of currents entering N1 through $\hat{n}$ wires and $\mathbf{v}_{\hat{N}}^{final}$ is the final solution through the superposition theorem. In this presentation, it is assumed that N1 does

not contain any nonlinearities, whereas N2 may contain nonlinear components that require iterations for an accurate solution. It can be further shown that

$$\mathbf{v}_{\hat{N}_\phi} = \mathbf{Z}_\phi \mathbf{i}_\phi \tag{1.14}$$

where $\mathbf{Z}_\phi$ is an impedance matrix relating the currents entering the set of nodes $\hat{N}$ to the contributions on voltages $\mathbf{v}_{\hat{N}_\phi}$. By using an incidence matrix, the branch voltages in N2 are related by

$$\mathbf{v}_\phi = \hat{\mathbf{A}}_{n\phi}^T \mathbf{v}_{\hat{N}}^{final} \tag{1.15}$$

where $\hat{\mathbf{A}}_{n\phi}$ is the nodal incidence matrix for the nodes in N2. If all the $\hat{n}$ wires are connecting from node to ground, then $\hat{\mathbf{A}}_{n\phi}$ becomes unitary and diagonal. By combining equation (1.13), (1.14) and (1.15)

$$\mathbf{v}_\phi = \hat{\mathbf{v}}_{th} + \hat{\mathbf{A}}_{n\phi}^T \mathbf{Z}_\phi \mathbf{i}_\phi \tag{1.16}$$

where $\hat{\mathbf{v}}_{th}$ is the vector of Thevenin voltages as found from N1. It is apparent that the Thevenin impedance matrix $\hat{\mathbf{Z}}_{th}$ is given by

$$\hat{\mathbf{Z}}_{th} = \hat{\mathbf{A}}_{n\phi}^T \mathbf{Z}_\phi \tag{1.17}$$

and consequently

$$\mathbf{v}_\phi = \hat{\mathbf{v}}_{th} + \hat{\mathbf{Z}}_{th} \mathbf{i}_\phi \tag{1.18}$$

Finally, it is noted that the currents $\mathbf{i}_\phi$ and voltages $\mathbf{v}_\phi$ are related through a function $\mathbf{\Phi}$ that could be linear or nonlinear:

$$\mathbf{\Phi}\left(\mathbf{v}_\phi, \mathbf{i}_\phi\right) = 0 \tag{1.19}$$

If $\mathbf{\Phi}$ is nonlinear then (1.18) must be solved using iterations and the Newton method.

The vector $\hat{\mathbf{v}}_{th}$ is time-dependent and must be found at each time-point solution. The matrix $\hat{\mathbf{Z}}_{th}$ may also have time-dependency due to switching devices in N1.

In Figure 1.13 it is assumed that N1 includes coupled (no delay-based transmission lines) networks. It is however possible that N1 contains decoupled networks or wires are used to connect separate networks. Let us assume that there are now two networks in N1 and N2 that are connected together

using the circuit of network N3. In that case the new representation of relations between networks is shown in Figure 1.14.

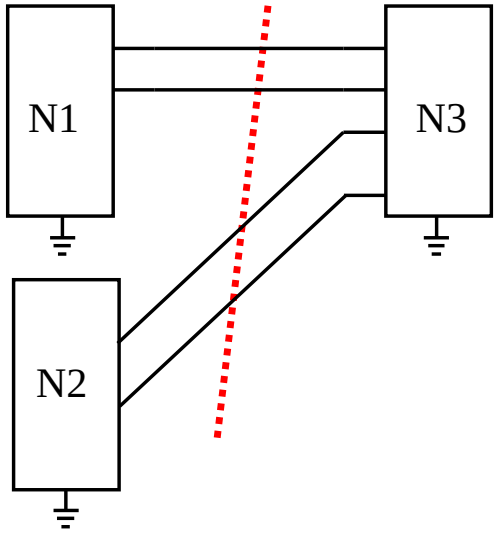

Figure 1.14 Two networks N1 and N2 connected through wires in network N3.

The MANA formulation of network equations for Figure 1.14 is given by

$$\begin{bmatrix} \mathbf{A_1} & \mathbf{0} & \mathbf{S}_{c_k} \\ \mathbf{0} & \mathbf{A_2} & \mathbf{S}_{c_m} \\ \mathbf{S_k} & \mathbf{S_m} & \mathbf{S_d} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \end{bmatrix} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \\ \mathbf{b_3} \end{bmatrix} \tag{1.20}$$

In the above system, $\mathbf{A_1}$ is the matrix of N1, $\mathbf{A_2}$ is the matrix of N2, the $\mathbf{S}$ matrices are the connecting matrices from network N3. It is possible that some off-diagonal $\mathbf{S}$ matrices are nullified due to disconnection between N1 and N2. Equation (1.20) is generic and allows N3 to contain longitudinal impedances, but for the following text and without any lack of generality, it is assumed that the impedances in N3 are simply zero, meaning that N1 and N2 are interconnected through ideal wires. For ideal wires, equation (1.20) becomes

$$\begin{bmatrix} \mathbf{A_1} & \mathbf{0} & \mathbf{S_k} \\ \mathbf{0} & \mathbf{A_2} & \mathbf{S_m} \\ \mathbf{S}_k^T & \mathbf{S}_m^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \end{bmatrix} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \\ \mathbf{0} \end{bmatrix} \tag{1.21}$$

The Compensation based solution of (1.20) (or (1.21)) can proceed as follows at each solution time-point. First it is necessary to solve with switches open (cutting the wires) by using

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1' \\ \mathbf{x}_2' \\ \mathbf{i}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{0} \end{bmatrix} \tag{1.22}$$

In this way the unknowns $\mathbf{x}_1'$ and $\mathbf{x}_2'$ are found before compensation and $\mathbf{x}_3 = \mathbf{i}_3$ for the wire currents (zero in this solution stage). From the found vectors $\mathbf{x}_1'$ and $\mathbf{x}_2'$ it is possible to directly extract the network Thevenin voltages $\hat{\mathbf{v}}_{th_1}$ and $\hat{\mathbf{v}}_{th_2}$, respectively. Then using current injection method in $\mathbf{b}_1''$ and $\mathbf{b}_2''$ for each network, it is possible to derive the Thevenin impedances. In the following equations the double-primed vectors signify the current injection method for finding the Thevenin impedances $\hat{\mathbf{Z}}_{th_1}$ and $\hat{\mathbf{Z}}_{th_2}$ (column-by-column process):

$$\begin{aligned} \mathbf{A}_1\mathbf{x}_1'' &= \mathbf{b}_1'' \\ \mathbf{A}_2\mathbf{x}_2'' &= \mathbf{b}_2'' \end{aligned} \tag{1.23}$$

At this stage it is possible to solve for the wire currents $\mathbf{i}_3$ with

$$\left[\hat{\mathbf{Z}}_{th_1} + \hat{\mathbf{Z}}_{th_2}\right]\mathbf{i}_3 = \hat{\mathbf{v}}_{th_1} - \hat{\mathbf{v}}_{th_2} \tag{1.24}$$

The above relation is illustrated in Figure 1.15 and it is assumed that the wire currents are oriented from left to right. It is also assumed that the coefficient matrix resulting in (1.24) is not singular.

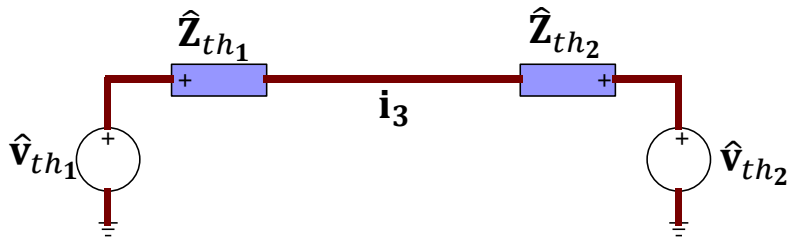

Figure 1.15 Compensation based equivalent of network in Figure 1.14.

After solving for $\mathbf{i}_3$ in (1.24), it is now possible to solve for the contributions ($\mathbf{x}_{1\phi}$ and $\mathbf{x}_{2\phi}$) of $\mathbf{i}_3$ on N1 and N2:

$$\begin{aligned} \mathbf{A}_1\mathbf{x}_{1\phi} &= -\mathbf{S}_\mathbf{k}\mathbf{i}_3 \\ \mathbf{A}_2\mathbf{x}_{2\phi} &= -\mathbf{S}_\mathbf{m}\mathbf{i}_3 \end{aligned} \tag{1.25}$$

Finally, we can apply superposition (compensation) to find

$$\begin{aligned}\mathbf{x_1} &= \mathbf{x_1'} + \mathbf{x_{1\phi}} \\ \mathbf{x_2} &= \mathbf{x_2'} + \mathbf{x_{2\phi}}\end{aligned} \tag{1.26}$$

The above procedure must be applied at each solution time-point. Any number of networks can be used and interconnected using wires (or impedances). Equations (1.23) and (1.25) can be solved in parallel. If there is any topological change in N1 or/and N2, it is necessary to recalculate $\hat{\mathbf{Z}}_{th_1}$ or/and $\hat{\mathbf{Z}}_{th_2}$. This is an important limitation and can become computationally very intensive with power-electronics based systems.

The above solution steps can be explained and performed differently. Equation (1.21) can be rewritten as follows

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{S_1} \\ \mathbf{0} & \mathbf{1} & \mathbf{S_2} \\ \mathbf{0} & \mathbf{0} & \mathbf{S_3} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{i_3} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{b}}_1 \\ \hat{\mathbf{b}}_2 \\ \hat{\mathbf{b}}_3 \end{bmatrix} \tag{1.27}$$

with

$$\mathbf{S_1} = \mathbf{A_1^{-1}} \mathbf{S_k} \tag{1.28}$$

$$\hat{\mathbf{b}}_1 = \mathbf{A_1^{-1}} \mathbf{b_1} \tag{1.29}$$

$$\mathbf{S_2} = \mathbf{A_2^{-1}} \mathbf{S_m} \tag{1.30}$$

$$\hat{\mathbf{b}}_2 = \mathbf{A_2^{-1}} \mathbf{b_2} \tag{1.31}$$

$$\mathbf{S_3} = \mathbf{S_k^T} \mathbf{S_1} + \mathbf{S_m^T} \mathbf{S_2} \tag{1.32}$$

$$\hat{\mathbf{b}}_3 = \mathbf{S_k^T} \hat{\mathbf{b}}_1 + \mathbf{S_m^T} \hat{\mathbf{b}}_2 \tag{1.33}$$

From (1.24) and (1.32) it is seen that

$$\mathbf{S_3} = \hat{\mathbf{Z}}_{th_1} + \hat{\mathbf{Z}}_{th_2} \tag{1.34}$$

From (1.24) and (1.33) it is apparent that

$$\hat{\mathbf{b}}_3 = \hat{\mathbf{v}}_{th_1} - \hat{\mathbf{v}}_{th_2} \tag{1.35}$$

because $\hat{\mathbf{b}}_1$ is actually $\mathbf{x_1'}$ in (1.22). The same applies for $\hat{\mathbf{b}}_2$ and $\mathbf{x_2'}$. It is noted that the coefficients of $\mathbf{S_m^T}$ are negative (ideal switch equations) and that explains the corresponding negative sign in (1.35). Finally, it is clear from (1.27), (1.25) and (1.26) that

$$\mathbf{x_1} = -\mathbf{A_1^{-1}} \mathbf{S_k} \mathbf{i_3} + \mathbf{A_1^{-1}} \mathbf{b_1} = \mathbf{x_{1_\phi}} + \mathbf{x_1'} \tag{1.36}$$

$$\mathbf{x_2} = -\mathbf{A_2^{-1}S_m i_3} + \mathbf{A_2^{-1}b_2} = \mathbf{x_{2_\phi}} + \mathbf{x'_2} \tag{1.37}$$

The approach derived with (1.27) is actually called MATE (Multi Area Thevenin Equivalent) [40][41]. As proven above with (1.36) and (1.37), and contrary to what is written in the literature, MATE is not a new theory or approach, it is in fact the Compensation method that was available in the literature much before!

The formulation of (1.20) indicates that if it is possible to find the bordered-block-diagonal matrix of a network, then it is possible to solve it in parallel even when distributed-parameter lines are not available. That solution uses the Compensation method (or MATE). Any number of networks can be separated (cut) and solved. The above illustration was made for two networks N1 and N2.

But there is a fundamental flaw in this approach. In a typical network, the networks N1 and N2 may encounter topological changes and require recalculating $\mathbf{S_3}$ in (1.34), which is computationally inefficient and even catastrophic if repetitive switching occurs due to power-electronics converters, for example. Moreover, all of the above is assuming linear networks and becomes inapplicable for practical problems with nonlinearities. It is possible in theory to extend the above Compensation based network tearing to include nonlinearities, but that may result into significant computational inefficiencies and annihilate the gains due to parallelization.

As a final demonstration, one can notice that the presentation given for (1.27) is simply the symbolic solution of (1.21). The steps are written here for convenience:

$$\mathbf{S_3 i_3} = \hat{\mathbf{b}}_\mathbf{3} \tag{1.38}$$

$$\mathbf{x_1} = -\mathbf{S_1 i_3} + \hat{\mathbf{b}}_\mathbf{1} \tag{1.39}$$

$$\mathbf{x_2} = -\mathbf{S_2 i_3} + \hat{\mathbf{b}}_\mathbf{2} \tag{1.40}$$

The solution order is

1. solve in parallel: equation (1.29) for $\hat{\mathbf{b}}_\mathbf{1}$ and (1.31) for $\hat{\mathbf{b}}_\mathbf{2}$
2. solve for $\hat{\mathbf{b}}_\mathbf{3}$ with (1.35) (the two parts of this equations can be calculated in parallel and then combined).
3. use (1.38) to find $\mathbf{i_3}$
4. solve (1.39) in parallel with (1.40).

In reality it is not possible to implement symbolic matrix inversions in actual software codes, as shown in (1.28)-(1.31). This is obvious for power system software developers. In fact LU decomposition must be used for solving (1.21) by re-writing it as follows

$$\begin{bmatrix} \mathbf{L_1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{L_2} & \mathbf{0} \\ \mathbf{L}_{31} & \mathbf{L}_{32} & \mathbf{L}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{U_1} & \mathbf{0} & \mathbf{U}_{13} \\ \mathbf{0} & \mathbf{U_2} & \mathbf{U}_{23} \\ \mathbf{0} & \mathbf{0} & \mathbf{U}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \end{bmatrix} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \\ \mathbf{b_3} \end{bmatrix} \tag{1.41}$$

where the coupling matrices in $\mathbf{L}$ and $\mathbf{U}$ are resulting from the interconnecting switch equations. It is noted that $\mathbf{L}_{33}$ and $\mathbf{U}_{33}$ are not zero even if $\mathbf{S_d} = \mathbf{0}$ (ideal wires). The purpose here is to implement the solution of (1.41) in parallel. This can be done by realizing that

$$\begin{bmatrix} \mathbf{L_1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{L_2} & \mathbf{0} \\ \mathbf{L}_{31} & \mathbf{L}_{32} & \mathbf{L}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{y_1} \\ \mathbf{y_2} \\ \mathbf{y_3} \end{bmatrix} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \\ \mathbf{b_3} \end{bmatrix} \tag{1.42}$$

The solutions of $\mathbf{y_1}$ and $\mathbf{y_2}$ are found in parallel. The solution of $\mathbf{y_3}$ can be found from

$$\mathbf{L}_{33}\mathbf{y_3} = \mathbf{b_3} - \mathbf{L}_{31}\mathbf{y_1} - \mathbf{L}_{32}\mathbf{y_2} \tag{1.43}$$

At this stage we have

$$\begin{bmatrix} \mathbf{U_1} & \mathbf{0} & \mathbf{U}_{13} \\ \mathbf{0} & \mathbf{U_2} & \mathbf{U}_{23} \\ \mathbf{0} & \mathbf{0} & \mathbf{U}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \end{bmatrix} = \begin{bmatrix} \mathbf{y_1} \\ \mathbf{y_2} \\ \mathbf{y_3} \end{bmatrix} \tag{1.44}$$

The solution of $\mathbf{x_3}$ is found from the last set of equations in (1.44):

$$\mathbf{U}_{33}\mathbf{x_3} = \mathbf{y_3} \tag{1.45}$$

Once $\mathbf{x_3}$ is known, it is possible to solve for $\mathbf{x_1}$ and $\mathbf{x_2}$ in parallel since

$$\begin{aligned} \mathbf{U_1}\mathbf{x_1} &= \mathbf{y_1} - \mathbf{U}_{13}\mathbf{x_3} \\ \mathbf{U_2}\mathbf{x_2} &= \mathbf{y_2} - \mathbf{U}_{23}\mathbf{x_3} \end{aligned} \tag{1.46}$$

This idea of parallelization outlined above is also said to be based on diakoptics [42][43]. It has been discussed in [42][43] (also other publications) and recently re-used also in [19]. It is the same idea as in (1.38)-(1.40).

Contrary to what is said in [19] it is obvious that LU decomposition was and must be used to solve (1.21). In addition, as it is said in [19], it is not necessary to derive the $\mathbf{L}_{3k}$ and $\mathbf{U}_{k3}$ (for $k = 1,...3$ in this case) matrices explicitly, since these matrices can be found directly from a sparse matrix

solver. Moreover, it is again emphasized that the time-consuming LU decomposition must be repeated in the presence of switches and nonlinearities. This important aspect is not considered in [42] and it will be even more inefficient with the approach proposed in [19] for finding $\mathbf{L}_{3k}$ and $\mathbf{U}_{k3}$.

In the above theory, there are no restrictions in the number of interconnected networks. One fundamental issue to be automate the derivation of (1.21). Switches can be inserted manually for parallel computations, but ideally it should be done automatically. It is possible to use tools like METIS to find bordered-block-diagonal matrices (as shown in (1.21)), but there are no demonstrations on the capabilities for arbitrary topology networks. The work in [19] uses the trivial duplication of a small network and no conclusions can be derived from such work.

The efficiency of bordered-block-diagonal formulation depends on the contents of the borders. The larger borders may require too many operations (see (1.43) and (1.45)-(1.46)). The resulting sparsity patterns must be analysed. In conclusion, significant further research is needed before applying this approach for practical systems.

Finally, it has been shown above that the Compensation method is also indirectly related to the formulation and solution of (1.21).

1.5 Sparse Matrices

Matrices in general have different types and different usage in many scientific fields. Sparse matrix is a term used to represent matrices with high number of zeros among its elements. These types of matrices appear in many scientific applications such as power systems, thermodynamics and different types of physical modelling.

A typical power grid matrix is typically more than 98% sparse. This means that most elements in the matrix are zero.

Sparse matrices possess specific characteristics that can be exploited to accelerate the solution process of very large-scale linear algebra problems. Sparse matrices require less storage memory. Using sparse matrices can dramatically improve the computational speed of large-scale linear

algebra problems. In fact, it is essential to apply sparse matrices for solving large scale power systems in an EMT-type method.

Several packages are available for solving sparse matrix problems.

A simple electrical circuit, its sparse matrix (MANA formulation) and its sparsity pattern are shown below in Figure 1.16 and Figure 1.17.

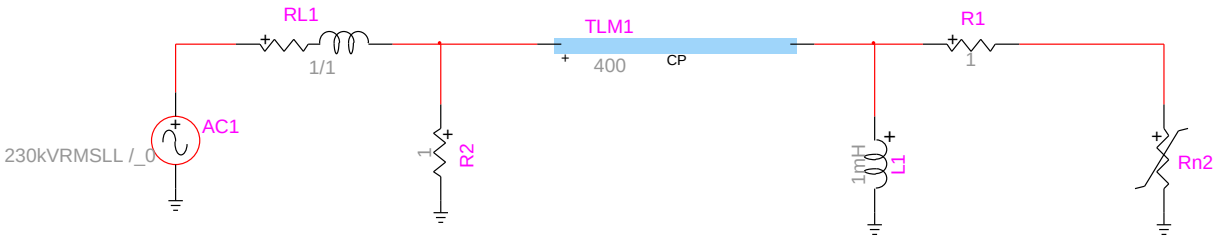

Figure 1.16 Small scale circuit with CP transmission line

$$\mathbf{A}=\begin{bmatrix} 0.5\times10^{-6} & 0 & 0 & -0.5\times10^{-6} & 1 \\ 0 & 0.50205 & -0.5 & 0 & 0 \\ 0 & -0.5 & 2.5 & 0 & 0 \\ -0.5\times10^{-6} & 0 & 0 & 1.0016 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.47}$$

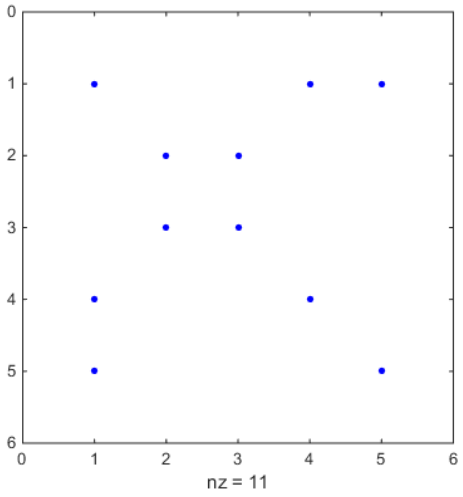

Figure 1.17 Non-zero pattern of matrix for the circuit of Figure 1.16.

1.5.1 Sparse Matrix representation

Sparse matrices can be presented is many different formats depending on the application they are used in. Different types of applications have different requirements in terms of ordering of matrix elements and representation in memory. One of these presentations is the classical way of matrix presentation that is stored in memory as a two-dimensional array. This type is often used in dense matrix memory storage and applications. However, adapting such a storage topology results in waste of memory resources due to the high number of zeros that will be stored, and the expanding of cache segments required for access during solution.

Other representations exist to reduce memory storage and consider only the non-zero elements in the matrix. The following two sections provide details about two sparse matrix representations that are often used in circuit analysis. These two storage techniques reduce the size of memory required to store sparse matrices.

1.5.1.1 Compressed Column Format

The compressed column (CSC) format allows storing a sparse matrix using three single dimensional vectors that include only nonzero elements in the sparse matrix and their locations. To fully represent each element location three vectors are used. Let the sparse matrix be $\mathbf{A}$, the size of the matrix $\mathbf{A}$ is $n \times n$ and the number of nonzero elements in $\mathbf{A}$ is nnz.

Let the three vectors representing $\mathbf{A}$ be $\mathbf{A}_P$, $\mathbf{A}_i$ and $\mathbf{A}_X$; where:

$\mathbf{A}_P$: $n+1$ long integer vector that contains indices of the starting nonzero elements of each column. This first element of this vector ($\mathbf{A}_P(0)$) is zero and the last element ($\mathbf{A}_P(n)$) is nnz.

$\mathbf{A}_i$: nnz long integer vector that stores the row number of each nonzero element in $\mathbf{A}$.

$\mathbf{A}_X$: nnz long vector that stores the numerical values of all nonzero elements in $\mathbf{A}$ in the same sequence they are listed in $\mathbf{A}_i$.

The matrix shown in (1.47) is used herein to illustrate the concept of CSC format. In order to better understand the explanation in this section, the non-zero elements of the matrix have been numbered in a sequential manner as shown in Table 1.1.

Table 1.1 Matrix (1.47) nonzero elements order

Element Number	**Element**
0	$\mathbf{A}(0,0)$
1	$\mathbf{A}(3,0)$
2	$\mathbf{A}(4,0)$
3	$\mathbf{A}(1,1)$
4	$\mathbf{A}(2,1)$
5	$\mathbf{A}(1,2)$
6	$\mathbf{A}(2,2)$
7	$\mathbf{A}(0,3)$
8	$\mathbf{A}(3,3)$
9	$\mathbf{A}(0,4)$
10	$\mathbf{A}(0,0)$

The $\mathbf{A}_P$ vector is formed by listing the sequential number of each column's first nonzero element. For example, $\mathbf{A}_P(2)$ is the sequential number of the first nonzero element of column 2 which is according to Table 1.1 is equal to 4. Hence $\mathbf{A}_P$ is equal to the vector shown in (1.48).

$$\mathbf{A}_P = \begin{bmatrix} 0 \\ 3 \\ 5 \\ 7 \\ 9 \\ 10 \end{bmatrix} \tag{1.48}$$

$\mathbf{A}_i$ vector is formed by listing the row number of all (1.47) nonzero elements in the same sequence they are listed in Table 1.1. For example, $\mathbf{A}_i(6)$ is the row number of the sixth element $\mathbf{A}(2,2)$ which is in this case row number 2. Therefor the vector $\mathbf{A}_i$ is formed as shown in (1.49).

$$\mathbf{A}_i = \begin{bmatrix} 0 \\ 3 \\ 4 \\ 1 \\ 2 \\ 1 \\ 2 \\ 0 \\ 3 \\ 0 \end{bmatrix} \tag{1.49}$$

Vector $\mathbf{A}_X$ stores the numerical values of all nonzero elements in matrix (1.47). The ordering of elements listed in $\mathbf{A}_X$ is done in the same sequence as $\mathbf{A}_i$.

$$\mathbf{A}_X = \begin{bmatrix} 0.5\times10^{-6} \\ -0.5\times10^{-6} \\ 1 \\ 0.50205 \\ -0.5 \\ -0.5 \\ 0.5 \\ -0.5\times10^{-6} \\ 1.0016 \\ 1 \end{bmatrix} \tag{1.50}$$

1.5.1.2 Compressed Row Format

The compressed row (CSR) format is similar to CSC in terms of methodology, however, the sequence of listing the non-zero elements is by rows instead of columns. This format lists the numerical values of all non-zero elements in $\mathbf{A}_X$, the column number (not row as CSC) of all non-zero elements in $\mathbf{A}_i$ and the index of starting nonzero element of each row $\mathbf{A}_P$. Equations (1.51) to (1.53) provide the CSR presentation of matrix (1.47).

$$\mathbf{A}_P = \begin{bmatrix} 0 \\ 3 \\ 5 \\ 7 \\ 9 \\ 10 \end{bmatrix} \tag{1.51}$$

$$\mathbf{A}_i = \begin{bmatrix} 0 \\ 3 \\ 4 \\ 1 \\ 2 \\ 1 \\ 2 \\ 0 \\ 3 \\ 0 \end{bmatrix} \tag{1.52}$$

$$\mathbf{A}_X = \begin{bmatrix} 2.5\times10^{-5} \\ -2.5\times10^{-5} \\ 1 \\ 1.0266 \\ -1 \\ -1 \\ 1 \\ -2.5\times10^{-5} \\ 1.0016 \\ 1 \end{bmatrix} \tag{1.53}$$

The computational performances of the above two sparse matrix representations are similar and one can use any one of them to code any sparse solver algorithm. However, it is very crucial when using an open source solver to know what representation the solver is expecting as an input, otherwise the solution results given by that solver will be wrong.

1.5.1.3 Solving a Sparse matrix

Solving a sparse matrix is the same as solving a dense matrix in terms of general steps and topology. Both types of matrices need to be factorized to two factors that have similar size as the original matrix and differ in structure from each other. These two factors are the upper factor $\mathbf{U}$ and lower factor $\mathbf{L}$. Equations (1.54) and (1.55) show the structure of a system of equation before and after factorization.

$$\mathbf{Ax = b} \tag{1.54}$$

$$\begin{bmatrix} \mathbf{L}_{1,1} & & & & \mathbf{0} \\ \mathbf{L}_{2,1} & \mathbf{L}_{2,2} & & & \\ \mathbf{L}_{3,1} & \mathbf{L}_{3,2} & \mathbf{L}_{3,3} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \mathbf{L}_{n,1} & \mathbf{L}_{n,2} & \mathbf{L}_{n,3} & \cdots & \mathbf{L}_{n,n} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{1,1} & \mathbf{U}_{1,2} & \mathbf{U}_{1,3} & \cdots & \mathbf{U}_{1,n} \\ & \mathbf{U}_{2,2} & \mathbf{U}_{2,3} & \cdots & \mathbf{U}_{2,n} \\ & & \mathbf{U}_{3,3} & \cdots & \mathbf{U}_{3,n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & \mathbf{U}_{n,n} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \vdots \\ \mathbf{b}_5 \end{bmatrix} \tag{1.55}$$

It can be seen from (1.55) that the upper factor (**U**) has nonzero elements only above the diagonal; whereas the lower factor (**L**) has nonzero elements under the diagonal line. Solving the system shown in (1.55) is done in two steps: Forward and backward substitution. During the forward substitution the equation shown in (1.56) is solved. While in the backward substitution the system shown in (1.58) is solved.

$$\begin{bmatrix} \mathbf{L}_{1,1} & & & & \mathbf{0} \\ \mathbf{L}_{2,1} & \mathbf{L}_{2,2} & & & \\ \mathbf{L}_{3,1} & \mathbf{L}_{3,2} & \mathbf{L}_{3,3} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \mathbf{L}_{n,1} & \mathbf{L}_{n,2} & \mathbf{L}_{n,3} & \cdots & \mathbf{L}_{n,n} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \\ \vdots \\ \mathbf{y}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \vdots \\ \mathbf{b}_5 \end{bmatrix} \tag{1.56}$$

Where:

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \\ \vdots \\ \mathbf{y}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{U}_{1,1} & \mathbf{U}_{1,2} & \mathbf{U}_{1,3} & \cdots & \mathbf{U}_{1,n} \\ & \mathbf{U}_{2,2} & \mathbf{U}_{2,3} & \cdots & \mathbf{U}_{2,n} \\ & & \mathbf{U}_{3,3} & \cdots & \mathbf{U}_{3,n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & \mathbf{U}_{n,n} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_5 \end{bmatrix} \tag{1.57}$$

$$\begin{bmatrix} \mathbf{U}_{1,1} & \mathbf{U}_{1,2} & \mathbf{U}_{1,3} & \cdots & \mathbf{U}_{1,n} \\ & \mathbf{U}_{2,2} & \mathbf{U}_{2,3} & \cdots & \mathbf{U}_{2,n} \\ & & \mathbf{U}_{3,3} & \cdots & \mathbf{U}_{3,n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & \mathbf{U}_{n,n} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \\ \vdots \\ \mathbf{y}_5 \end{bmatrix} \tag{1.58}$$

1.6 Sparse Solvers

In order to achieve a higher speed of EMT simulation a suitable and efficient solver package must be used. Accelerating the performance of any EMT simulation package starts by selecting a solver that is in line with the matrix structure of that EMT simulation package. In this section, a brief presentation of three sparse solvers is given and supported with different types of examples. These sparse solvers are KLU [1], SuperLU [44], and the minimum degree ordering based solver that is currently used in EMTP [4].

1.6.1 SuperLU

SuperLU [45][46] is a sparse solver package that is proven to be efficient and reliable when solving different types of sparse matrices in different applications such as fluid dynamics, structural mechanics, chemical process simulation, circuit simulation, electromagnetic fields and so on [45]. The SuperLU package is an open source solver that is available online for download. In order to find the solution of a system of equations, SuperLU performs the following steps:

1. Minimize the number of fill-in elements in matrices $\mathbf{L}$ and $\mathbf{U}$. This step is used to manipulate the matrix and permute it in such a way that it reduces the number of non-zero elements in $\mathbf{L}$ and $\mathbf{U}$ factors, and hence reduces overall solution time. SuperLU offers the use of many techniques that are integrated inside the package and can reduce fill-in in quick and reliable manner without affecting the solution quality or numerical stability [46].
2. Once the fill-in ordering is determined, SuperLU runs a symbolic algorithm to define the non-zero pattern of $\mathbf{L}$ and $\mathbf{U}$ factors. This algorithm helps in allocating all fill-ins that are introduced in $\mathbf{L}$ and $\mathbf{U}$ factors and estimating the size of memory storage the problem in hand requires before even starting the numerical step [45]. The nonzero pattern found in this step is used during the numerical factorization in order to find the numerical coefficients of $\mathbf{L}$ and $\mathbf{U}$

factors. All other elements that are flagged as non-zero in this step will not be calculated and will be treated as zeros.

3. Allocate all memory required for factorization work and for storing **L** and **U** matrices. SuperLU package uses the compressed row storage CRS format to store sparse matrices as seen in section 1.5.1.2.
4. Numerically factorize the matrix **A** into **L** and **U**. This step is the most time-consuming step among all other tasks and operations in the package. It starts by running a symbolic analysis on the permuted **A** matrix (permuted in step 1) and determining the location of all Supernodes (explained below) [45].

   The use of Supernodes allows to create dense nodes (regions) in the matrix in order to use packages such as BLAS level 2 that is suitable for dense matrices. Supernodes have many types and take many forms. Figure 1.18 shows different types of Supernodes that may be encountered in a matrix.

   The dense nodes shown in Figure 1.18 represent Supernodes that may occur in different formats. The Supersnode T1 shown in Figure 1.18 - (a) illustrates a dense matrix that is full (with all elements in the Supernode being nonzero) and nonzero elements along the columns of **L** and rows of **U**. T2 shown in Figure 1.18 - (b) illustrates a Supernode that has a dense **L** matrix along the diagonal that is full and non-zero elements scattered in the off-diagonal columns of **L**. However, no non-zero elements exist in the rows associated with **U**. T3 shown in Figure 1.18-(c) illustrates a Supernode that has a dense **L** matrix along the diagonal that is full with non-zero elements scattered in the off-diagonal columns of **L** and a full **U** block with no off-diagonal elements along its rows. The last type of Supernode T4 is shown in Figure 1.18 - (d) where full **L** and **U** blocks can be found along the diagonal. The **L** has non-zero elements scattered along its columns and a stretch of non-zero elements scattered in the columns associated with the full part of **U** [45][46].

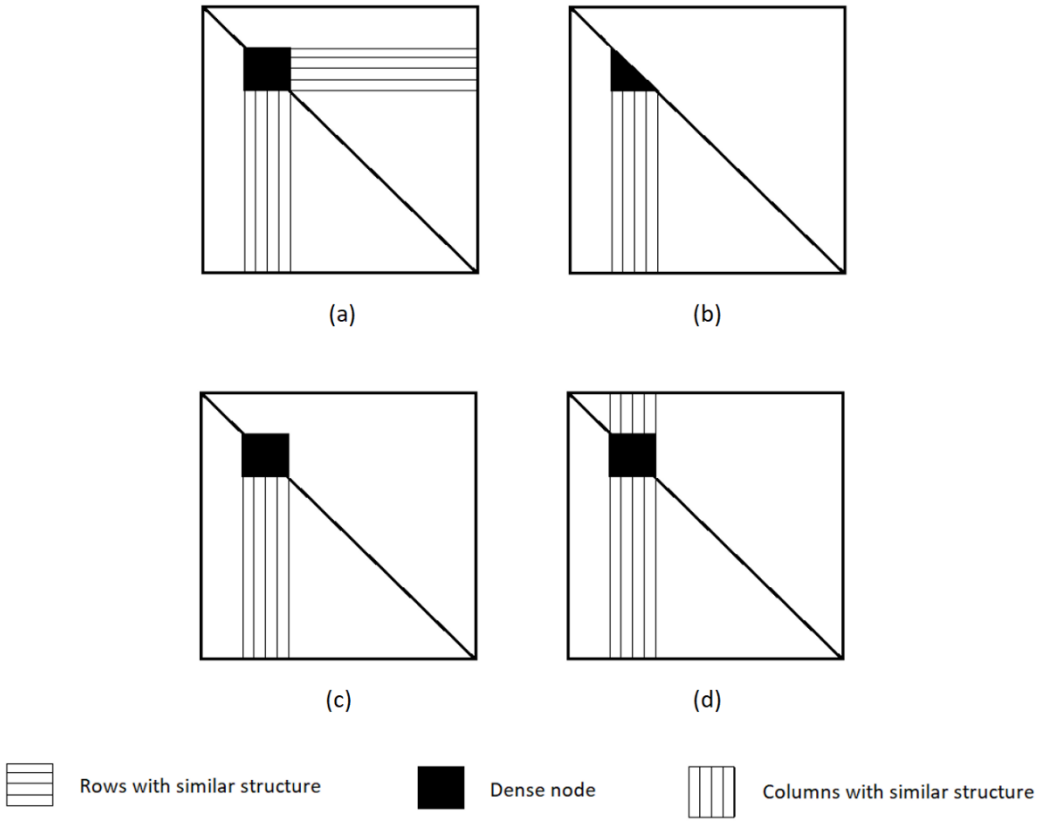

Figure 1.18 Types of Supernodes T1, T2, T3 and T4 respectively

The following example [46] provides a better understanding of the concept of Supernodes. Let us take the matrix **A** shown in Figure 1.19 in its initial form without any ordering

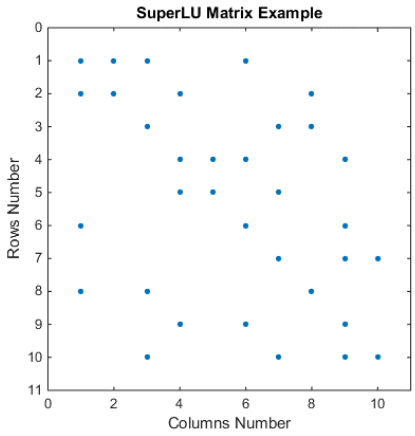

Figure 1.19 SuperLU matrix example

According to the SuperLU steps mentioned above, the matrix undergoes symbolic analysis that applies a fill-in reduction ordering and it determines the $\mathbf{L}$ and $\mathbf{U}$ non-zero patterns. Figure 1.20 and Figure 1.21 show the $\mathbf{L}$ and $\mathbf{U}$ matrices and their non-zeros pattern. The Supernode allocation uses a special matrix that is called the filled matrix, to find all possible Supernodes. According to [46], the filled matrix can be found by (1.59).

$$\mathbf{F} = \mathbf{L} + \mathbf{U} - \mathbf{I} \tag{1.59}$$

where $\mathbf{I}$ is an identity matrix of size $n \times n$ subtracted from $\mathbf{L}$ and $\mathbf{U}$ in order to remove all elements along the diagonal. According to the type of Supernode selected (T1, T2, T3 or T4), the SuperLU algorithm finds all possible Supernodes in the matrix $\mathbf{F}$. Figure 1.20 and Figure 1.21 below show the sparsity pattern of $\mathbf{L}$ and $\mathbf{U}$ of matrix $\mathbf{A}$, Figure 1.22 shows the sparsity pattern of matrix $\mathbf{F}$, and Figure 1.23 shows all Supernodes of type T1 that were found in the matrix $\mathbf{F}$.

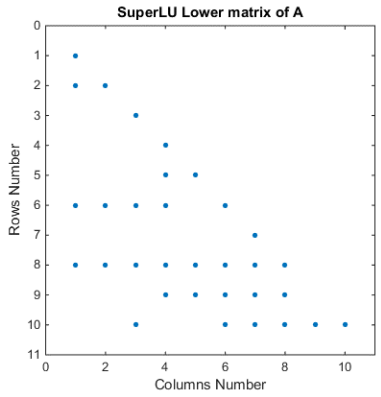

Figure 1.20 SuperLU Example $\mathbf{L}$ matrix (symbolic version)

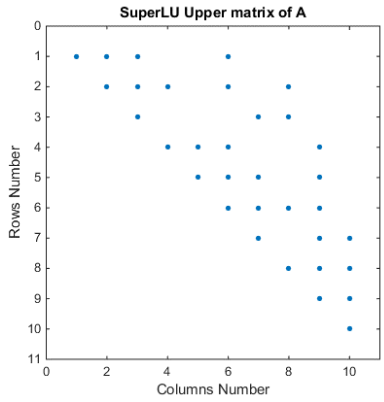

Figure 1.21 SuperLU Example $\mathbf{U}$ matrix (symbolic version)

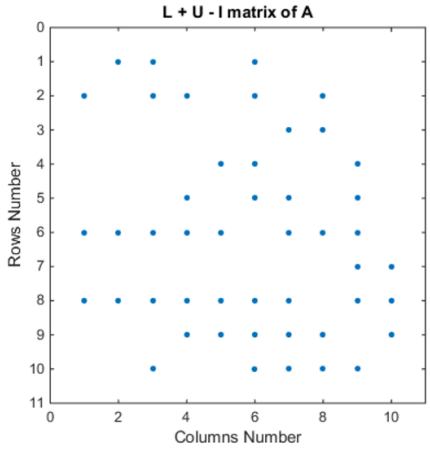

Figure 1.22 **L** + **U** - **I** of matrix **A** (symbolic)

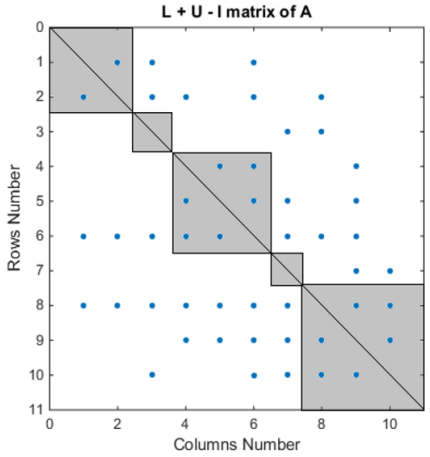

Figure 1.23 T1 Supernodes of matrix **A**

Based on the type of Supernode selected, SuperLU runs a search technique that explores all possible Supernodes that fits all criteria of the selected Supernode type. In Figure 1.23 for example, the Supernode that was selected is T1 and as can be seen in the figure there are five Supernodes found in the matrix. The first Supernode is a 2×2 node with scattered non-zero elements along the columns and rows corresponding to this full Supernode. Once the Supernodes are determined, they are treated as dense matrices for storage and computation. SuperLU uses different types of left looking algorithms that factors the matrix **A** into **L** and **U**. Depending on the user selection or the degree of Supernodes density, different types of standard dense matrix-vector multiplication kernels are used such as level 2 BLAS and level 3 BLAS. This algorithm treats the Supernode and its corresponding columns/rows as single

elements and a call to BLAS algorithm will expand these elements into their actual structure and perform the appropriate computation on them to find the actual $\mathbf{L}$ and $\mathbf{U}$. This algorithm is proven to be very efficient in factorizing dense matrices and sparse matrices with less than 90% sparsity [46].

5. The last step of SuperLU solution is performing a backward and forward substitution to find the results. This step uses the traditional substitution techniques that is based on the $\mathbf{L}$ and $\mathbf{U}$ factors found in step 4 and the right hand vector of the system.

1.6.2 KLU

KLU [1] is a sparse matrix solver that employs hybrid ordering mechanisms and elegant factorization to solve any sparse system. It has been tested on several simulation packages and proven to be a fast and reliable solver especially when solving circuit analysis problems. It is based on Gilbert-Peierls' algorithm [47] with partial pivoting that aims at computing the nonzero pattern of the $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ factors and the numerical values in a total time of $O(flops(LU))$. This technique consists of two major stages, the symbolic analysis and the numerical analysis. Throughout this thesis all the cases listed herein have only blocks along the diagonal without any off diagonal nonzero elements. This is mainly due to the fact that all subnetworks separated by a delay-based transmission line are strongly connected and the strongly connected subnetworks are decoupled from each other. This means that the matrix $\mathbf{A}$ in its block triangular format (BTF) has N number of blocks along its diagonal as can be seen in (1.60):

$$\hat{\mathbf{A}}=\begin{bmatrix} \hat{\mathbf{A}}_1 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \hat{\mathbf{A}}_2 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \hat{\mathbf{A}}_3 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \hat{\mathbf{A}}_4 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \hat{\mathbf{A}}_5 \end{bmatrix} \tag{1.60}$$

The KLU solver will be applied to each diagonal block $\hat{\mathbf{A}}_i$ separately and they can be solved in parallel due to the fact that their solution is independent of each other.

1.6.2.1 KLU Symbolic Analysis

During the symbolic analysis, block $\hat{\mathbf{A}}_i$ will be analyzed to find its nonzero pattern. This analysis is becoming more and more challenging with the integration of partial pivoting in the sparse solver. The nonzero pattern of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ factors is hard to predict with a dynamic pivoting order that keeps changing and for that reason this symbolic stage is being computed and updated every time a pivot for $\hat{\mathbf{A}}_i$ is updated.

The Gilbert-Peierls' algorithm uses graph theory to calculate the nonzero pattern of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$

[47] which is based on finding the reachability of any nonzero element of $\hat{\mathbf{A}}_i$. The reachability calculation starts by assuming the lower factor $\hat{\mathbf{L}}_i$ to be equal to a unity matrix and then starts processing the block $\hat{\mathbf{A}}_i$ in sequence order column by column. As seen in Figure 1.24, if block $\hat{\mathbf{A}}_i$ column k (shown at the right side of Figure 1.24) has a nonzero element at row j and factor $\hat{\mathbf{L}}_i$ has a non-zero element at element (i, j) then the element at row i of column k must be non-zero. By applying this algorithm, the location of all non-zero elements in $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ can be determined before the numerical step even starts, and the calculation of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ elements will be only for those nonzero elements found during this symbolic stage.

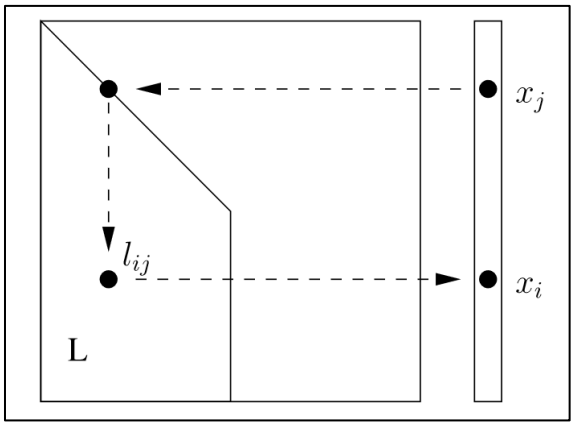

Figure 1.24 Nonzero pattern of $\mathbf{x}$ when solving $\mathbf{Lx=b}$

Once the locations of non-zero elements of each column are determined, the non-zero elements numerical values are determined as shown in Figure 1.25.

```
L̂ = I
for k = 1 : n
        x = L̂ \ Â( : , k)
        % (partial pivoting on x can be done here)
        Û(1 : k , k) = x(1 : k)
        L̂(k : n , k) = x(k : n) / U(k , k)
end
```

Figure 1.25 $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ non-zero pattern allocation

The simple circuit example shown in Figure 1.16 is used in section 1.6.2.2 to illustrate KLU symbolic analysis in a very detailed manner.

1.6.2.2 KLU Numerical Analysis

Once the nonzero pattern is found, a left looking numerical factorization with partial pivoting is conducted to calculate the factors $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ numerical values. The matrix $\hat{\mathbf{A}}_i$ now becomes:

$$\hat{\mathbf{A}}_i = \hat{\mathbf{L}}_i \hat{\mathbf{U}}_i \tag{1.61}$$

In equation (1.61), $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ contain the upper and lower factors of BTF diagonal blocks respectively. It is worth mentioning that KLU solver has two types of factorization, namely Full-Factorization (KLU-FF) and Re-Factorization (KLU-RF).

During the KLU-FF, a symbolic analysis is done on the matrix to determine the non-zero pattern of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ (as seen in section 1.6.2.1), followed by numerical analysis involves a partial pivoting to select the pivot of each column being factorized.

KLU-RF on the other hand, assumes that the non-zero pattern of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ calculated in the previous iteration and the pivoting order of the previous iterations are still valid and can be used. The KLU-RF function only updates the numerical values of $\hat{\mathbf{U}}_i$ and $\hat{\mathbf{L}}_i$ based on the changes to block $\hat{\mathbf{A}}_i$.

In order to fully understand both the symbolic analysis of KLU and KLU-FF, the system of equations of Figure 1.16 circuit is written for a given operating condition of Rn2. This system was built using the MANA approach discussed earlier.

$$\begin{bmatrix} 0.5\times10^{-6} & 0 & 0 & -0.5\times10^{-6} & 1 \\ 0 & 0.50205 & -0.5 & 0 & 0 \\ 0 & -0.5 & 2.5 & 0 & 0 \\ -0.5\times10^{-6} & 0 & 0 & 1.0016 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} -2.64183 \\ 3.46719 \\ 187794.2 \\ 0.01497 \\ 0 \end{bmatrix} \tag{1.62}$$

After running BTF symbolic analysis, the vector shown in (1.63) was calculated to be the row permutation vector that is used to sort the rows to obtain BTF form.

$$\mathbf{P} = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 2 \\ 3 \end{bmatrix} \tag{1.63}$$

This vector determines the location of each row in the permuted matrix. For example, $\mathbf{P}(2) = 4$ means that row 4 in $\mathbf{A}$ will be row 2 in the $\hat{\mathbf{A}}$. Hence the row permutation matrix of $\mathbf{A}$ is shown in (1.64). The permutation matrix (1.64) is formed by reallocating the diagonal elements of a given identity matrix column to the row number indicated in (1.63). For example, second column's diagonal element is moved to the forth row since $\mathbf{P}(2) = 4$.

$$\mathbf{A}_P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{1.64}$$

The column permutation is the transposed version of the matrix shown in (1.64) as can be seen in (1.65) and (1.66).

$$\mathbf{A}_C = \mathbf{A}_P^T \tag{1.65}$$

$$\mathbf{A}_C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{1.66}$$

The following equation shows the permuted version of system (1.62):

$$\hat{\mathbf{A}} = \mathbf{A}_P \mathbf{A} \mathbf{A}_C \tag{1.67}$$

$$\hat{\mathbf{A}} = \begin{bmatrix} 0.5\times10^{-6} & -0.5\times10^{-6} & 1 & 0 & 0 \\ -0.5\times10^{-6} & 1.0016 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.50205 & -0.5 \\ 0 & 0 & 0 & -0.5 & 2.5 \end{bmatrix} \tag{1.68}$$

The matrix given in (1.68) is factorized and solved below with pivot tolerance equal to 0.01 and all pivots elements are initially assumed to be along the diagonal of $\hat{\mathbf{A}}$ as shown in (1.69).

$$\mathbf{P}_{Pivot} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \tag{1.69}$$

$\mathbf{P}_{Pivot}$ shown in (1.69) indicates that the pivot of column 1 is located at row 1, the pivot of column 2 is located at row 2, the pivot of column 3 is located at row 3 and so on. The factorization starts by assuming that both $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ are equal to an identity matrix as shown in (1.70) and (1.71)

$$\hat{\mathbf{L}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.70}$$

$$\hat{\mathbf{U}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.71}$$

Since KLU is a left looking solver and it factorize the matrix $\hat{\mathbf{A}}$ one column at a time, the factorization process is done in the following five steps:

1. Factorizing the 1st column of matrix (1.68):

   The factorizing process starts by finding the location of non-zero elements in $\hat{\mathbf{L}}_1$ and $\hat{\mathbf{U}}_1$ as follow:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5\times10^{-6} \\ -0.5\times10^{-6} \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Figure 1.26 Analysis of 1st column of matrix (1.68)

   In the above symbolic analysis, the launch of the maximum reach from row 1, 2 and 3 of the right hand side was not able to find any non-zero elements below the diagonal elements, and was not able to introduce any non-zero elements into $\hat{\mathbf{L}}_1$ and $\hat{\mathbf{U}}_1$ other than the already non-zero valued elements in row 1, 2 and 3. Hence, the location of non-zero elements of $\hat{\mathbf{L}}_1$ and $\hat{\mathbf{U}}_1$ are shown in (1.72) and (1.73):

$$\hat{\mathbf{L}}_1 = \begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & 1 & 0 & 0 & 0 \\ \times & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.72}$$

$$\hat{\mathbf{U}}_1 = \begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.73}$$

The calculation of the numerical values of $\hat{\mathbf{L}}_1$ and $\hat{\mathbf{U}}_1$ is done by solving the system shown in (1.74).

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0.5\times10^{-6} \\ -0.5\times10^{-6} \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{1.74}$$

The symbolic analysis of column 1 above found that only x_1, x_2 and x_3 are non-zero elements, hence the system of equations in (1.74) is only solved for x_1, x_2 and x_3. This solution results in $x_1 = 0.5\times10^{-6}$, $x_2 = -0.5\times10^{-6}$ and $x_3 = 1$.

Applying partial pivoting on the solution of (1.74) starts by finding the largest element in the $\mathbf{x}$ vector and comparing it with the element stored at the pivot location. Using the assumption used at the beginning of this example, the assumed pivot is stored at element $\mathbf{x}(1)$ and the largest element in $\mathbf{x}$ is found to be $\mathbf{x}(3)$. Testing the pivot criteria on both pivot candidates ($\mathbf{x}(1)$ and $\mathbf{x}(3)$) as shown in (1.75) it turns out that the pivot of column 1 must be replaced with the element of row 3.

$$\varepsilon_p \mathbf{x}(3) > \mathbf{x}(1) \tag{1.75}$$

This change in pivoting order updates $\mathbf{P}_{Pivot}$ as shown in (1.76) and the rows of system of equations shown in (1.68) are permuted according to the new $\mathbf{P}_{Pivot}$ as shown in (1.77).

$$\mathbf{P}_{Pivot} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 4 \\ 5 \end{bmatrix} \tag{1.76}$$

$$\hat{\mathbf{A}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1.0016 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.5\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.50205 & -0.5 \\ 0 & 0 & 0 & -0.5 & 2.5 \end{bmatrix} \tag{1.77}$$

Using the formulation in Figure 1.25 $\hat{\mathbf{L}}_1$ and $\hat{\mathbf{U}}_1$ are calculated to be:

$$\hat{\mathbf{L}}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.78}$$

$$\hat{\mathbf{U}}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.79}$$

2. Factorizing the 2nd column of matrix (1.68):

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & 1 & 0 & 0 & 0 \\ \times & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1.0016 \\ -0.5\times10^{-6} \\ 0 \\ 0 \end{bmatrix}$$

Figure 1.27 Analysis of 2nd column of matrix (1.68)

The launch of maximum reach from row 2 and 3 of the right-hand side was not able to find any non-zero element below the diagonal of the second column and hence it was not able to add any fill-in. Hence, the non-zero elements of $\hat{\mathbf{L}}_2$ and $\hat{\mathbf{U}}_2$ are:

$$\hat{\mathbf{L}}_2 = \begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ 0 & \times & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.80}$$

$$\hat{\mathbf{U}}_2 = \begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.81}$$

In order to find the numerical values of $\hat{\mathbf{L}}_2$ and $\hat{\mathbf{U}}_2$ the system shown in (1.82) is solved

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.0016 \\ -0.5\times10^{-6} \\ 0 \\ 0 \end{bmatrix} \tag{1.82}$$

The above system of equations in (1.82) is only solved for x_2 and x_3. This solution results in $x_2 = 1.001575$ and $x_3 = -0.5\times10^{-6}$. Applying partial pivoting on the results of (1.82) it can be seen that element x_2 is larger than all other elements in $\mathbf{x}$, hence the existing pivot is valid.

Using the formulation shown in Figure 1.25, $\hat{\mathbf{L}}_2$ and $\hat{\mathbf{U}}_2$ are calculated to be:

$$\hat{\mathbf{L}}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.83}$$

$$\hat{\mathbf{U}}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.001575 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.84}$$

3. Factorizing the 3rd column of matrix (1.68):

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Figure 1.28 Analysis of 3rd column of matrix (1.68)

In the above analysis, the launch of the maximum reach from row 3 wasn't able to find any non-zero elements below the diagonal element of the 3rd column. Hence, the non-zero elements of $\hat{\mathbf{L}}_3$ and $\hat{\mathbf{U}}_3$ are:

$$\hat{\mathbf{L}}_3 = \begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.85}$$

$$\hat{\mathbf{U}}_3 = \begin{bmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 \\ 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.86}$$

In order to find the numerical values of $\hat{\mathbf{L}}_3$ and $\hat{\mathbf{U}}_3$ the system shown in (1.87) is solved.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{1.87}$$

The above system of equations in (1.87) is only solved for x_1, x_2 and x_3. This solution results in $x_3 = 1$. Applying partial pivoting on the results of (1.87) it can be seen that element x_3 is larger than all other elements in $\mathbf{x}$ hence the existing pivot is valid.

Using the formulation shown in Figure 1.25 $\hat{\mathbf{L}}_3$ and $\hat{\mathbf{U}}_3$ are calculated to be:

$$\hat{\mathbf{L}}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.88}$$

$$\hat{\mathbf{U}}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.001575 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.89}$$

4. Factorizing the 4th column of matrix (1.68):

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.5021 \\ -0.5 \end{bmatrix}$$

Figure 1.29 Analysis of 4th column of matrix (1.68)

In the above analysis, the launch of the maximum reach from row 4 and 5 failed to find any non-zero elements below the diagonal elements and was not able to introduce any other non-zero elements into $\hat{\mathbf{L}}_4$. Hence, the non-zero elements of $\hat{\mathbf{L}}_4$ and $\hat{\mathbf{U}}_4$ are:

$$\hat{\mathbf{L}}_4 = \begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & \times & 1 \end{bmatrix} \tag{1.90}$$

$$\hat{\mathbf{U}}_4 = \begin{bmatrix} \times & \times & \times & 0 & 0 \\ 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.91}$$

In order to find the numerical values of $\hat{\mathbf{L}}_4$ and $\hat{\mathbf{U}}_4$ the system shown in (1.92) is solved.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.50205 \\ -0.5 \end{bmatrix} \tag{1.92}$$

The symbolic analysis of this column found that only elements x_4 and x_5 are non-zero,

hence the above system of equations in (1.92) is only solved for x_4 and x_5 only. This solution results in $x_4 = 1.0266$ and $x_5 = -1$. Applying partial pivoting on the results of (1.92) it can be seen that element x_4 is larger than all other elements in x hence the existing pivot is valid.

Using the formulation shown in Figure 1.25, $\hat{\mathbf{L}}_4$ and $\hat{\mathbf{U}}_4$ are calculated to be:

$$\hat{\mathbf{L}}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -0.9959034 & 1 \end{bmatrix} \tag{1.93}$$

$$\hat{\mathbf{U}}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.001575 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.50205673 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.94}$$

5. Factorizing the 5th column of matrix (1.68):

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & \times & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0.5 \\ 0.5 \end{bmatrix}$$

Figure 1.30 Analysis of 5th column of matrix (1.68)

The above symbolic analysis was able to determine that rows 4 and 5 of column 5 of $\hat{\mathbf{L}}_5$ and $\hat{\mathbf{U}}_5$ will be non-zero entries. Hence, the non-zero elements of $\hat{\mathbf{L}}_5$ and $\hat{\mathbf{U}}_5$ are

$$\hat{\mathbf{L}}_5 = \begin{bmatrix} \times & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & \times & \times \end{bmatrix} \tag{1.95}$$

$$\hat{\mathbf{U}}_5 = \begin{bmatrix} \times & \times & \times & 0 & 0 \\ 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix} \tag{1.96}$$

In order to find the numerical values of $\hat{\mathbf{L}}_5$ and $\hat{\mathbf{U}}_5$ the system shown in (1.97) is solved.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -0.9959034 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0.5 \\ 2.5 \end{bmatrix} \tag{1.97}$$

Using the symbolic analysis above it was found that only elements x_4 and x_5 are non-zero; hence, the above system of equations in (1.97) is solved for these two elements only. This solution results in the following $\mathbf{x}$ vector.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0.5 \\ 2.0020483 \end{bmatrix} \tag{1.98}$$

Applying partial pivoting on the results of (1.98) it can be seen that element x_5 is larger than all other elements in $\mathbf{x}$, hence the existing pivot is valid.

Using the formulation shown in figure 4, $\hat{\mathbf{L}}_5$ and $\hat{\mathbf{U}}_5$ are calculated to be:

$$\hat{\mathbf{L}}_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -0.9959034 & 1 \end{bmatrix} \tag{1.99}$$

$$\hat{\mathbf{U}}_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.001575 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.50205673 & -0.5 \\ 0 & 0 & 0 & 0 & 2.0020483 \end{bmatrix} \tag{1.100}$$

In order to verify the above factorization, one can multiply $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ to obtain $\hat{\mathbf{A}}$.

$$\hat{\mathbf{L}}\hat{\mathbf{U}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1.0016 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.5\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.50205 & -0.5 \\ 0 & 0 & 0 & -0.5 & 2.5 \end{bmatrix} \tag{1.101}$$

The second step of the numerical analysis stage is the solution step that performs forward and backward substitution in order to obtain the results of vector $\hat{\mathbf{x}}$ [1]. This step is summarized below for the above example:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1.0016 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.5\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.50205 & -0.5 \\ 0 & 0 & 0 & -0.5 & 2.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 187785.9 \\ 3.471719 \\ -2.64183 \\ 0.002821 \\ 0 \end{bmatrix} \tag{1.102}$$

$$\hat{\mathbf{L}}\hat{\mathbf{U}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 187785.9 \\ 3.471719 \\ -2.64183 \\ 0.002821 \\ 0 \end{bmatrix} \tag{1.103}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5\times10^{-6} & 1 & 0 & 0 & 0 \\ 0.5\times10^{-6} & -0.4992223\times10^{-6} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -0.9959034 & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 187785.9 \\ 3.471719 \\ -2.64183 \\ 0.002821 \\ 0 \end{bmatrix} \quad (1.104)$$

Where

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.001575 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.50205673 & -0.5 \\ 0 & 0 & 0 & 0 & 2.0020483 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \quad (1.105)$$

The solution of equation (1.105) results in:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 1.8779\times10^{5} \\ 3.5656 \\ -2.7357 \\ 0.0028 \\ 0.0028 \end{bmatrix} \quad (1.106)$$

Substituting equation (1.106) in (1.105) yields to the following equation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.001575 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.50205673 & -0.5 \\ 0 & 0 & 0 & 0 & 2.0020483 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1.8779\times10^{5} \\ 3.5656 \\ -2.7357 \\ 0.0028 \\ 0.0028 \end{bmatrix} \quad (1.107)$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1.87785\times10^{5} \\ 3.56000 \\ -2.7357 \\ 0.0070 \\ 0.0014 \end{bmatrix} \quad (1.108)$$

KLU uses a scaling algorithm to scale all elements of $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ factors based on a predefined scale. This scaling topology helps reduces the size of numbers used throughout the calculation and increases accuracy. However, throughout this research project it was found that the scaling algorithm did not add much advantages for the selected applications, but it rather increased computation time and code complexity with no valid reason.

1.6.3 EMTP-MDO solver

From the above two sections on SuperLU and KLU, it can be noticed that these two solvers have two types of matrix permutations, namely fill-in reduction permutation and structural permutation. The first type of permutation is specialized in reducing the fill-in elements in $\mathbf{L}$ and $\mathbf{U}$ factors, hence reduces the computation time for factorization and substitution steps. The other type of permutation is specialized in permuting the structure of $\mathbf{A}$ in order to allow for some degree of parallelization. For example, in the case of BTF permutation, in context of this thesis this permutation decouples all strongly connected regions of the matrix forming multiple submatrices that are fully independent of each other. This approach allows numerical steps to work on different submatrices in parallel and assign each or a group of submatrices to a specific processor to reduce computation time [48].

Unlike SuperLU and KLU, EMTP-MDO [4] has only one permutation that is the fill-in reduction permutation. This permutation is based on the minimum degree ordering technique. The minimum degree ordering is a generic technique works on reducing the fill-in of $\mathbf{L}$ and $\mathbf{U}$ by re-ordering the matrix rows and columns based on different nodes connectivity. Various minimum degree algorithms exist in the literature such as basic minimum degree (which is used in EMTP-MDO), approximate minimum degree (AMD) [49][50] and column approximate degree (COLAMD) [51][52]. It was proven in [1] that AMD gives the best performance for circuits matrices. AMD finds a permutation vector $\mathbf{P}$ to reduce the fill-ins in Cholesky factorization and apply it on the matrix $\mathbf{A}$ as follow: $\mathbf{PAP}^T$. AMD assumes no numerical pivoting within its scope and all its ordering is purely symbolic. COLAMD on the other hands produces a column permutation vector $\mathbf{Q}$ to reduce the fill-in of $\mathbf{L}$ and $\mathbf{U}$ and it applies it on matrix $\mathbf{A}$ as follow: $\mathbf{Q}^{\mathrm{T}}\mathbf{AQ}$.

The basic minimum degree ordering is based on selecting a node with minimum number of connected edges and factorizes the column or row that corresponds to that node [29]. This

technique performs a symbolic elimination on the non-zero structure of the system. During this stage, a pivot element is chosen from those un-eliminated diagonal entries. The symbolic elimination results in a permutation array that is used to permute the system main matrix $\mathbf{A}$ into the pivoting order found during this stage. The permuted $\mathbf{A}$ matrix is solved in three steps, symbolic factorization, numerical factorization and forward-backward substitution.

During the symbolic factorization, the nonzero structure of the rows of $\mathbf{L}$ and $\mathbf{U}$ factors is determined based on the structure of the permuted matrix $\mathbf{A}$. Once the nonzero pattern of $\mathbf{L}$ and $\mathbf{U}$ is found, the numerical values of all $\mathbf{L}$ and $\mathbf{U}$ coefficients are calculated. The last step in the solution stage is to perform a backward and forward substitution to find the solution of $\mathbf{x}$ based on the right hand side $\mathbf{b}$ and using the $\mathbf{L}$ and $\mathbf{U}$ factors generated in the first two steps.

Therefore, the application of parallel computation with the EMTP-MDO solver is not feasible due to the fact that the minimum degree ordering technique does not permute the matrix into any kind of block diagonal format but rather reduces the fill-in of $\mathbf{L}$ and $\mathbf{U}$ factors. Even if a structural permutation technique is applied to obtain the BTF form of $\mathbf{A}$, the numerical solver has to be re-coded or changed to adapt to data structure needed by the added permutation.

The following example illustrates how minimum degree ordering technique works to find the best permutation topology that contributes to a maximum reduction of fill-ins. The system of equations shown in (1.62) is used herein.

In order to find the minimum degree ordering of the matrix shown in (1.62) one can use an elimination graph that is basically an undirected graph of the matrix $\mathbf{A}$, that is

$$G = (V, E) \tag{1.109}$$

Where, G is the undirected graph of the matrix shown in (1.62), V represents graph's G nodes and E is the edges between different nodes in G. The elimination graph as can be seen in Figure 1.31 has n vertices, where n is the size of matrix $\mathbf{A}$, and each vertex represents a column/row. The elimination graph can be established by adding a connection between any vertex with all other vertices that are adjacent to it. For example, vertex 1 is adjacent to vertices 4 and 5 through elements $\mathbf{A}(0,3)$ and $\mathbf{A}(0,4)$.

The minimum degree ordering technique starts by eliminating the node with minimum weight, which is the node with the least number of connections or the minimum number of adjacent nodes. In case of multiple nodes having similar weight, the selected node is chosen randomly. At start, the node weight is evaluated as shown in Table 1.2. This table shows the weight of nodes in the elimination graph shown in Figure 1.31. Since this is a small case, all nodes order varies between one and two and the elimination process is very simple and straightforward. The first node to be eliminated will be node 3 and the graph becomes as shown in Figure 1.32. The nodes are evaluated again and the node with the minimum weight is eliminated and the remaining nodes order is reevaluated once again. This process in repeated until the graph's last node is eliminated. Figure 1.33 to Figure 1.35 show the changes in elimination graph. In [49] a more complex example is given where the weight of the nodes changes during the elimination process and the tracking and storing of all new established edges becomes challenging. Larger graphs use a modified and simpler way of handling the nodes elimination process. This approach is based on creating Quotien graphs of the matrix as shown in [49].

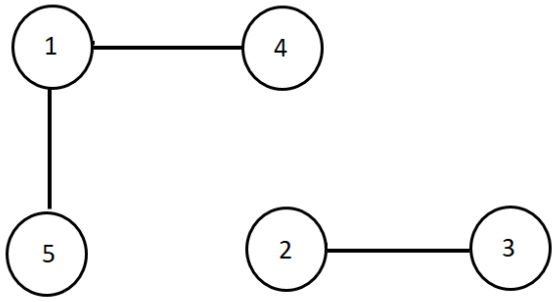

Figure 1.31 Matrix **A** elimination graph

Table 1.2 Elimination graph nodes weight

Node	**weight**
1	2
2	1
3	1
4	1
5	1

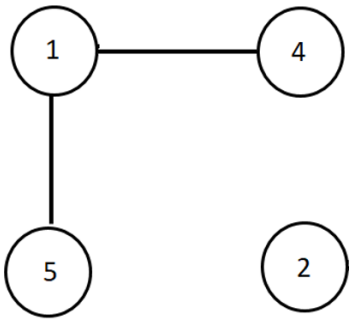

Figure 1.32 1st elimination step of matrix **A** graph

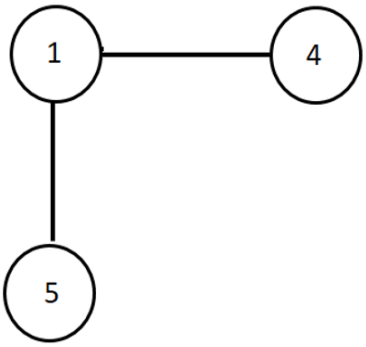

Figure 1.33 2nd elimination step of matrix **A** graph

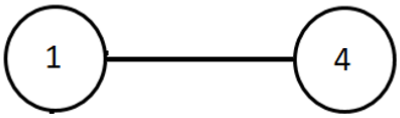

Figure 1.34 3rd elimination step of matrix **A** graph

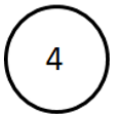

Figure 1.35 4th elimination step of matrix **A** graph

1.6.4 Threading

Different parallel algorithms have been proposed in order to apply parallel computation on EMT simulations in different real-time simulation tools [16]-[18] and in off-line applications [5][15][53]. Most of these proposed algorithms are based on some user intervention and/or user defined network partitioning as in [19][27] where the user has to define the location where the network can be partitioned, or using external packages to link all subnetworks as in [15]. Such partitioning technique requires the user to have full knowledge of the system in order to make an informative decision on the best partitioning location that guarantees the highest computation gain. Such decisions become very complicated for large power systems. Other proposed parallel techniques are based on the use of graphical processing units GPU [54]. This approach efficiency decreases with the increased size of the power network being analyzed and makes it not suitable for handling practical problems. Problems arise when repetitive matrix factorizations are needed.

In this project, the threading implementation is meant to be automated and the program will be self-sufficient to determine the feasibility of threading and assigning the location of threading and task distribution. The threading part will be done on an already existed sparse solver (KLU) and it will be mainly based on two threading techniques, the OpenMP and C++11 threading.

1.6.4.1 OpenMP

OpenMP is a thread programming tool used in the implementation of parallel processing [55] in Windows computing environment. It is a high-level threading technique that requires the user to define different segments of the code where parallel processing is required using one of OpenMP pragma notations. The compiler will launch, control, synchronize and terminate threads without much extra effort to be made by the user. Using the OpenMP implementation requires minimum changes to the sequential code as opposed to other threading techniques.

In KLU the symbolic analysis is done in a pure sequential fashion. However, when it comes to numerical analysis parallelizing the factorization and the forward-backward substitution is essential to convert the KLU code to a parallel solver given that the network that is being solved can be solved in parallel or has at least one delay-based line in it. Parallelizing the factorization process was done by surrounding the factorization loop that factorizes all blocks of BTF matrix with a pragma bracket that will guarantee a parallel execution of that loop, and in a similar manner the forward-backward substitution can be parallelized. The assignment of each bock to a specific thread requires defining a special mapping that is given to OpenMP before starting the parallel segment. In addition, distinguishing between thread specific variables and threads shared variables is critical to avoid any overlap between different threads and to avoid any kind of race conditions during OpenMP threads synchronization.

In the OpenMP parallel version of sparse solver, each thread has a set of private variables that are exclusive for each thread and can be accessed only by the thread that owns them. However, there are a set of variables mainly used for statistical purposes that are common between all threads such as the number of non-zero elements in L and U.

Given that the KLU algorithm was written in C, the link between the Fortran code based EMT simulation package and the KLU solver was done using the ICO_C_BINDING standard [55]. This standard allows for interchange calls from Fortran to C and vice versa as will be seen in Chapter 2.

1.6.4.2 C++11 Threading

Unlike OpenMP, using C++ multithreading is a low-level implementation which requires the user to manage all threads from the moment threads start until the moment threads finish. If a Fortran

based EMT simulation package is used, the Fortran-C++ threading lifecycle is divided into 3 phases: initialization, execution and finalization.

During the initialization phase, the number of threads required and the matrices **A** and **b** arrays are passed to the initialization function (init()) where the symbolic analysis of BTF and the launching of all worker threads (threads that do factorization and triangular solving for an assigned block) take place. In order to avoid the overhead of creating threads every time the solver is being called, a thread pool is created by storing the thread handles in a global vector for later use. Once the threads are created the initialization function is blocked until all worker threads indicate that they have started up and ready to execute iterations.

The signaling mechanism between the initialization function and the worker threads is implemented via an atomic integer that gets incremented by one whenever a starting thread is initialized and ready, after which the worker threads move into the BLOCKED state waiting for a signal to run the numerical analysis code. The C++ main thread performs a busy-wait until all threads are ready, then it returns control to FORTRAN. Illustration of this phase is shown in Figure 1.36.

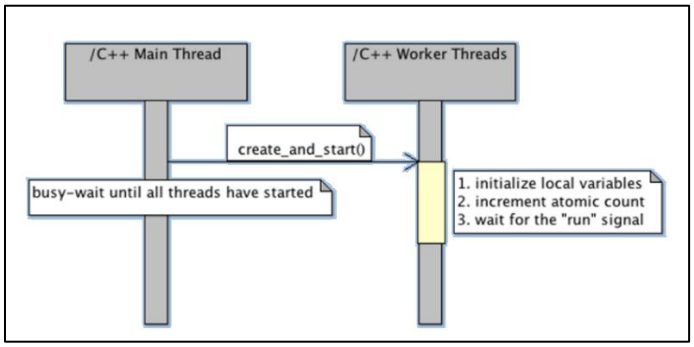

Figure 1.36 Parallel implementation initialization phase

During the execution phase, the C++ main thread signals to all worker threads to start executing the iteration code, and then waits for all threads to finish running numerical analysis on their assigned blocks. The start signaling is implemented via a condition variable and a lock. Once the worker threads are notified of the signal (by the operating system), they transition from the BLOCKED state into the RUNNING state.

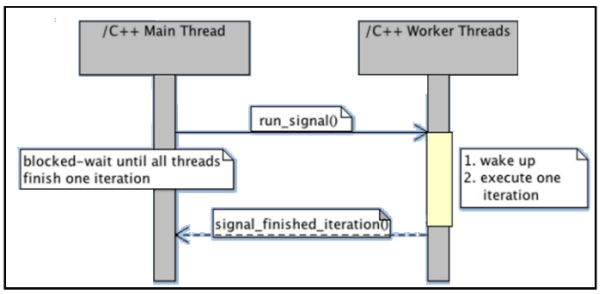

Figure 1.37 Parallel implementation Execution

Worker threads execute the numerical analysis process, and signal to the main thread that they've finished via a semaphore. After which, the worker threads go back again to the BLOCKED state waiting for another start signal. Once all worker threads are finished, the main thread returns control to EMT package to prepare for the next time step / iteration.

The last phase of the process is the finalization phase. It is a phase required to terminate worker threads and to release any resources allocated by the C++ subsystem. It starts by invoking a termination function (finish()) from FORTRAN into C++ main thread. The C++ main thread then sets an exit flag and awakes the worker threads. The worker threads check the flag and exist.

This approach was implemented and integrated with EMTP, and different test scenarios were used to validate its performance; However, the EMT simulation acceleration obtained with this approach was not considerable and the complexity that involved in implementing this parallel design was another reason that let to drop it as a viable parallel implementation option of KLU.

CHAPTER 2 IMPLEMENTATION OF SPARSE SOLVER PACKAGE FOR EMT SIMULATION

The objective of this chapter is to deliver the implementation of a new sparse matrix solver in an EMT-type tool for the solution of electrical network (power) equations. The implementation will be performed into EMTP [1] using an available code interface.

In addition to the power network equations, EMTP uses a separate solver for its control system equations [3]. The control system part is not considered in this thesis since its solution procedures fall into another category [56].

Before proceeding, it is important to recall that the network equations in EMTP are solved using the MANA formulation. Also, it is recalled that EMTP uses a fully iterative solver. At each time-point EMTP solves a set of equations similar to (1.54). It is recalled here for convenience:

$$\mathbf{A}_t \mathbf{x}_t = \mathbf{b}_t \tag{2.1}$$

where the vector $\mathbf{b}_t$ contains independent sources and history terms resulting from device model discretization, the matrix $\mathbf{A}_t$ is actually the Jacobian matrix [3]. At each time point the above system is solved using the LU decomposition of $\mathbf{A}_t$. If a nonlinear function changes its operating segment or an ideal switch changes its position, it becomes necessary to update $\mathbf{A}_t$ and consequently its factorization. This process is essential for maintaining an accurate solution for network but is also creates significant extra computational load.

After each time-point solution of (2.1), it is necessary to use the solution vector $\mathbf{x}_t$ for updating all model history terms preparing the solution for the next time-point. Analysis has demonstrated that when accounting for all solution procedure, the main computation burden is the iterative solution in network equation (2.1). Improving its performance through the usage of a better sparse matrix solver and through parallelization, is the main research objective of this thesis.

As a first step, this chapter presents the selection of a new sparse matrix solver. The second step is the parallelization of the solution process for gaining more computational performance.

The new sparse matrix solver is named *Sparse Matrix Package for EMTs (SMPEMT).*

2.1 Selecting a Sparse Solver

In the previous chapter three sparse solver packages have been introduced, namely SuperLU, KLU and EMTP-MDO existing sparse solver (MDO). The three solvers were briefly introduced to explain the underlying programming techniques. The objective here is to conduct numerical tests for actual systems. A variant of the Hydro-Quebec grid is used to perform tests with EMTP.

The size of the Hydro-Quebec $\mathbf{A}$ matrix is 41797x41797 with 99% sparsity and 169369 non-zero entries. The simulation interval was chosen to be 1 s with a time-step of 50 μs. The network contains nonlinearities and the average number of iterations per time-point is equal to 2.07.

The computation time of solving equation (2.1) for different solvers are presented in Table 2.1.

Table 2.1 Solver comparison timings (s), EMTP solution, Single-Core

	MDO-EMTP	**KLU Solver**	**SuperLU solver**
Time Domain solution	1048	1216	5340
Number of KLU-FF	-	68532	-
Number of KLU-RF	-	57543	-

As seen from the above table, the MDO-EMTP solver is apparently the fastest among the three selected packages running on a single CPU core. This phenomenon is due to the heavy computation involved in the numerical analysis of both KLU solver and SuperLU solver. However, after studying the algorithms of various packages, it was found that the KLU package has significant potential of improvement for EMT-type solution. In addition to the fact that other circuit-based simulation packages demonstrated the potential of the KLU method [1][57].

The most useful features with the KLU package are:

1. The existence of BTF partitioning technique in KLU that is implemented as part of the solver package.
2. The data structure used in storing $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ matrices.
3. The existing ordering techniques can be replaced easily with a user defined alternative.
4. The code structure and code documentation.
5. The separation of different tasks in different C functions.
6. The minimization of a potential stack overflow run time error during BTF permutation calculation. This is mainly due to the fact that the stack used in all recursive calls in the package are allocated on the heap and have more memory backup compare to other stack

memory given by the compiler.

7. The lower fill-in produced during factorization compared to the other two packages.
8. KLU performance is proven to be better that other solvers with matrices with high sparsity degree.
9. The use of an efficient left looking factorization technique that reduces floating-point operations during numerical factorization.
10. The existence of re-factoring technique (KLU-RF) that can significantly speed-up the re-factorization process due to a time-domain varying $\mathbf{A}$ matrix.

Figure 2.1 shows a test case that was used to compare the ordering results of produced by KLU and EMTP-MDO solvers. This case represents Reluctance network based transformer model. The case consists of one block due to the fact that it does not include any transmission line, and it has many nonlinear devices such as non-linear resisters. From the Figure 2.2, Figure 2.3 and Figure 2.4 it can be seen that KLU solver was able to produce an ordering that results in less fill-in compare to EMTP-MDO and this will result in less factorization and solution time for $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$.

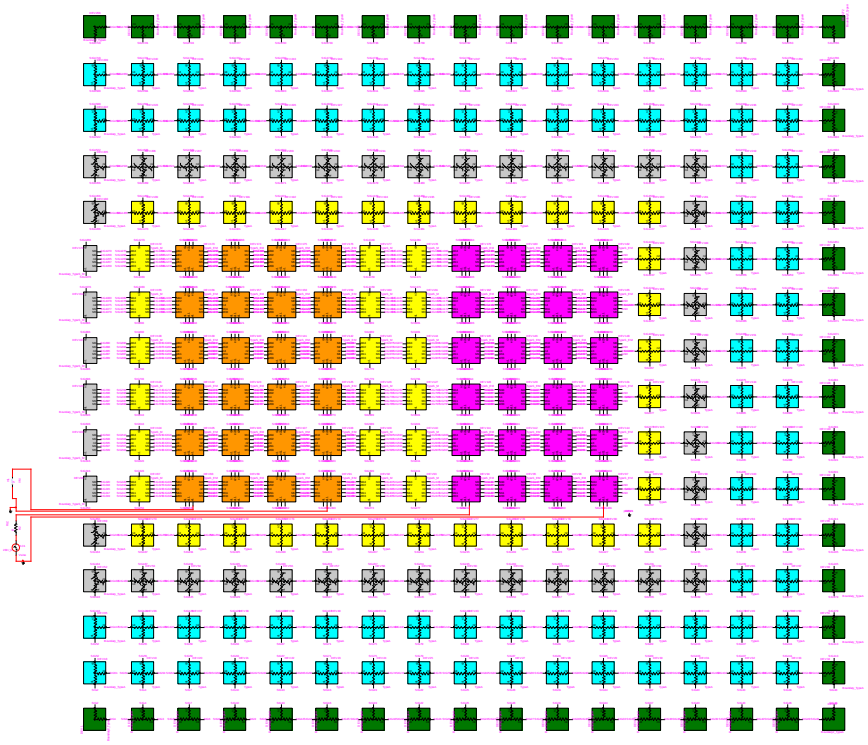

Figure 2.1 Top view of Reluctance based transformer model case (Contributed by EDF)

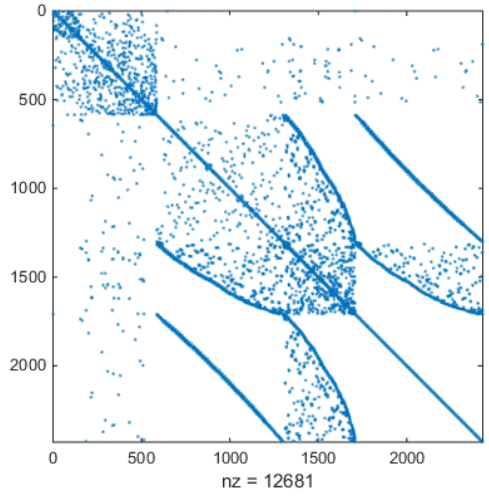

Figure 2.2 Reluctance based transformer model case matrix sparsity pattern

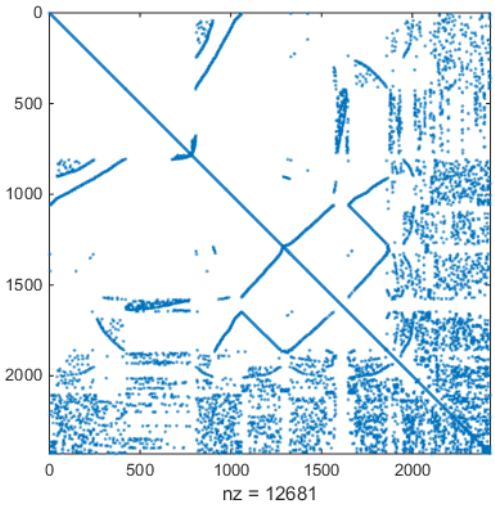

Figure 2.3 Reluctance based transformer model case EMTP permutation for Matrix **A**

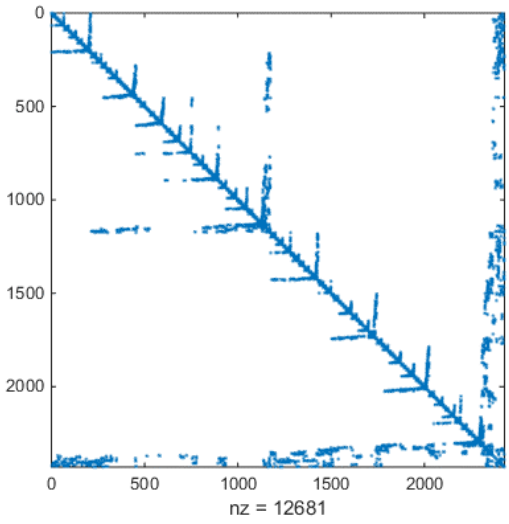

Figure 2.4 Reluctance based transformer model case KLU permutation of matrix **A**

Table 2.2 Shows the sparse matrix solution time for 1 second simulation of the case shown in Figure 2.1. It can be seen from the table that the difference between the two solvers are huge and the effect of fill-in reduction and an efficient ordering is important in computation time optimization.

Table 2.2 Reluctance based transformer model case $\mathbf{Ax} = \mathbf{b}$ solution time

Solver	**Time (second)**
EMTP-MDO	5560
KLU	75

2.2 KLU Interface

Since EMTP computational engine is written in FORTRAN, and the KLU package is coded in C, it is necessary to establish an interface between both programming languages for allowing calls into the KLU solver. This work is also applicable to other EMT-type simulation tools [5] that are written in Fortran.

In order to establish such an interface, the ISO_C_BINDING module is used to provide Fortran with access to different C types and functions. KLU has also three user defined types namely KLU_common, KLU_symbolic and KLU_numeric. The KLU_common consists of different tuning parameters that are used in defining how the solver runs and the type of ordering package used in the symbolic analysis, KLU_symbolic consists of variables related to the symbolic analysis of KLU and other techniques such as the column ordering permutation vector $\mathbf{P}_c$ and row permutation vector $\mathbf{P}_R$ and KLU_numeric contains all variables related to numerical factorization and solution. Other than user defined types, ISO_C_BINDING provides an interface between Fortran and conventional C types such as *int, double, float* and all other types including pointer types. Figure 2.5 shows different FORTRAN types with their corresponding C variable types; for example, the type *int* in C matches the type *INTEGER* in FORTRAN. In order to map the two variables in an ISO_C_BINDING interface, the Name constant types shown in Figure 2.5 is used. Figure 2.6 and Figure 2.7 give an example of how this mapping is done in defining ISO_C_BINDING interface for a user defined type and a function.

The syntax of declaring a user defined type using ISO_C_BINDING is shown in Figure 2.6. This declaration will allow the use of KLU types (i.e KLU_symbolic) in different FORTRAN modules.

Fortran Type	Named constant	C type
INTEGER	C_INT	int
INTEGER	C_SHORT	short int
INTEGER	C_LONG	long int
INTEGER	C_LONG_LONG	long long int
INTEGER	C_SIGNED_CHAR	signed char/unsigned char
INTEGER	C_SIZE_T	size_t
INTEGER	C_INT8_T	int8_t
INTEGER	C_INT16_T	int16_t
INTEGER	C_INT32_T	int32_t
INTEGER	C_INT64_T	int64_t
INTEGER	C_INT128_T	int128_t
INTEGER	C_INT_LEAST8_T	int_least8_t
INTEGER	C_INT_LEAST16_T	int_least16_t
INTEGER	C_INT_LEAST32_T	int_least32_t
INTEGER	C_INT_LEAST64_T	int_least64_t
INTEGER	C_INT_LEAST128_T	int_least128_t
INTEGER	C_INT_FAST8_T	int_fast8_t
INTEGER	C_INT_FAST16_T	int_fast16_t
INTEGER	C_INT_FAST32_T	int_fast32_t
INTEGER	C_INT_FAST64_T	int_fast64_t
INTEGER	C_INT_FAST128_T	int_fast128_t
INTEGER	C_INTMAX_T	intmax_t
INTEGER	C_INTPTR_T	intptr_t
INTEGER	C_PTRDIFF_T	ptrdiff_t
REAL	C_FLOAT	float
REAL	C_DOUBLE	double
REAL	C_LONG_DOUBLE	long double
REAL	C_FLOAT128	__float128
COMPLEX	C_FLOAT_COMPLEX	float _Complex
COMPLEX	C_DOUBLE_COMPLEX	double _Complex
COMPLEX	C_LONG_DOUBLE_COMPLEX	long double Complex
REAL	C_FLOAT128_COMPLEX	__float128 _Complex
LOGICAL	C_BOOL	_Bool
CHARACTER	C_CHAR	char

Figure 2.5 ISO_C_BINDING types declaration

```
type, Bind(C)  :: klu_symbolic
    real(kind=C_DOUBLE) ::   symmetry
    real(kind=C_DOUBLE) ::   est_flops
    real(kind=C_DOUBLE) ::   lnz, unz
    TYPE(C_PTR)         ::   LLnz
    TYPE(C_PTR)         ::   block_flops
    integer(kind=C_INT) ::   n
    integer(kind=C_INT) ::   nz
    TYPE(C_PTR)         ::   P
    TYPE(C_PTR)         ::   Q
    TYPE(C_PTR)         ::   R
    TYPE(C_PTR)         ::   RB
    integer(kind=C_INT) ::   nzoff
    integer(kind=C_INT) ::   nblocks
    integer(kind=C_INT) ::   maxblock
    integer(kind=C_INT) ::   ordering
    integer(kind=C_INT) ::   do_btf
    integer(kind=C_INT) ::   TimeStep_Call
    integer(kind=C_INT) ::   structural_rank
   integer(kind=C_INT)  ::   maxLnz
end type  klu_symbolic
```

Figure 2.6 KLU symbolic declaration using ISO_C_BINDING

The declaration of functions using ISO_C_BINDING is different than variables. It consists of adding a special section in the interface file that encapsulates all function declarations and their argument types. Figure 2.7 shows the syntax of ISO_C_BINDING function declaration, this declaration lists the function name, its arguments and it defines the name of the function in the C based code. The interface then imports the type mapping of all arguments used in the function and includes a list of arguments with their types. This standard is used in Figure 2.7 to build the interface for KLU_SOLVER_ANALYZE.

```
interface
       subroutine FUNCTION_NAME(argumets1, argumets2) bind(C, name="FUNCTION_NAME")
                       import C_INT, C_FLOAT     ! Imports all types of arguments
                       INTEGER(C_INT), VALUE   :: argumets1 ! Argument1 of type integer
                       INTEGER(C_FLOAT), VALUE :: argumets2 ! Argument1 of type float
       end subroutine FUNCTION_NAME

end interface
```

Figure 2.7 ISO_C_BINDING functions declaration syntax

```
interface
        function KLU_SOLVER_ANALYZE(n,Ap,Ai)  bind (C, name="KLU_SOLVER_ANALYZE")
                        import C_INT, C_PTR
                        INTEGER(kind=C_INT)        :: KLU_SOLVER_ANALYZE
                        INTEGER(Kind=C_INT), VALUE :: n
                        type(C_PTR), VALUE :: Ap
                        type(C_PTR), VALUE :: Ai
        end function KLU_SOLVER_ANALYZE
end interface
```

Figure 2.8 ISO_C_BINDING declaration of KLU_ANALYZE function

2.3 Pivot validity test

In order to improve the performance of the KLU solver, it was necessary to make modification in its code. As explained above, the KLU-RF technique of the KLU solver assumes that the non-zero pattern of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ calculated in the previous KLU-FF and the previous pivoting order are still valid. Therefore, by making such an assumption the symbolic analysis during the numerical factorization and the computation of the partial pivoting order can be skipped. The KLU-RF function updates the numerical values of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ to reflect any changes in the matrix $\hat{\mathbf{A}}_i$. However, KLU-RF technique does not involve any pivot validation, and it blindly uses the old non-zero pattern and the old pivoting order without any verification. This practice increases the risk of introducing numerical instability and producing inaccurate results.

The first added feature to KLU in this thesis is called the “pivot validity test”. It deals mainly with how KLU decides on whether conducting a KLU-FF on a certain block $\hat{\mathbf{A}}_i$ or KLU-RF is required. The pivot validity testing is an added feature that allows the KLU solver to be able to make an informative decision on whether a KLU-FF or KLU-RF is needed. Pivot validity testing criterion is based on verifying that the element stored at the location of each column’s pivot is greater than all other elements belonging to the same column by at least the user defined tolerance. The use of tolerance avoids calculating a new pivoting order if the new candidate pivot is slightly greater than the previously calculated one. Equation (2.2) shows the pivot validity testing verification criterion. This test is performed on every column of block $\hat{\mathbf{A}}_i$. In case any column of $\hat{\mathbf{A}}_i$ fails to satisfy this criterion, block $\hat{\mathbf{A}}_i$ is deemed ineligible for KLU-RF and a KLU-FF is needed.

$$\varepsilon_p a_{new} > a_{old} \tag{2.2}$$

Where, ε_p is the pivot testing criteria, a_{new} is the new pivot element candidate and a_{old} is the old pivot element. The pivot tolerance plays a major rule in controlling the acceleration gain of SMPEMT. This is due to the fact that this tolerance ratio determines the number of times KLU-FF is executed as opposed to KLU-RF. The higher ε_p results in increasing number of iterations use KLU-FF compare to lower ε_p.

Figure 2.9 shows a flow chart presentation of the sequence of pivot validity testing implemented in KLU during an EMT type simulation process.

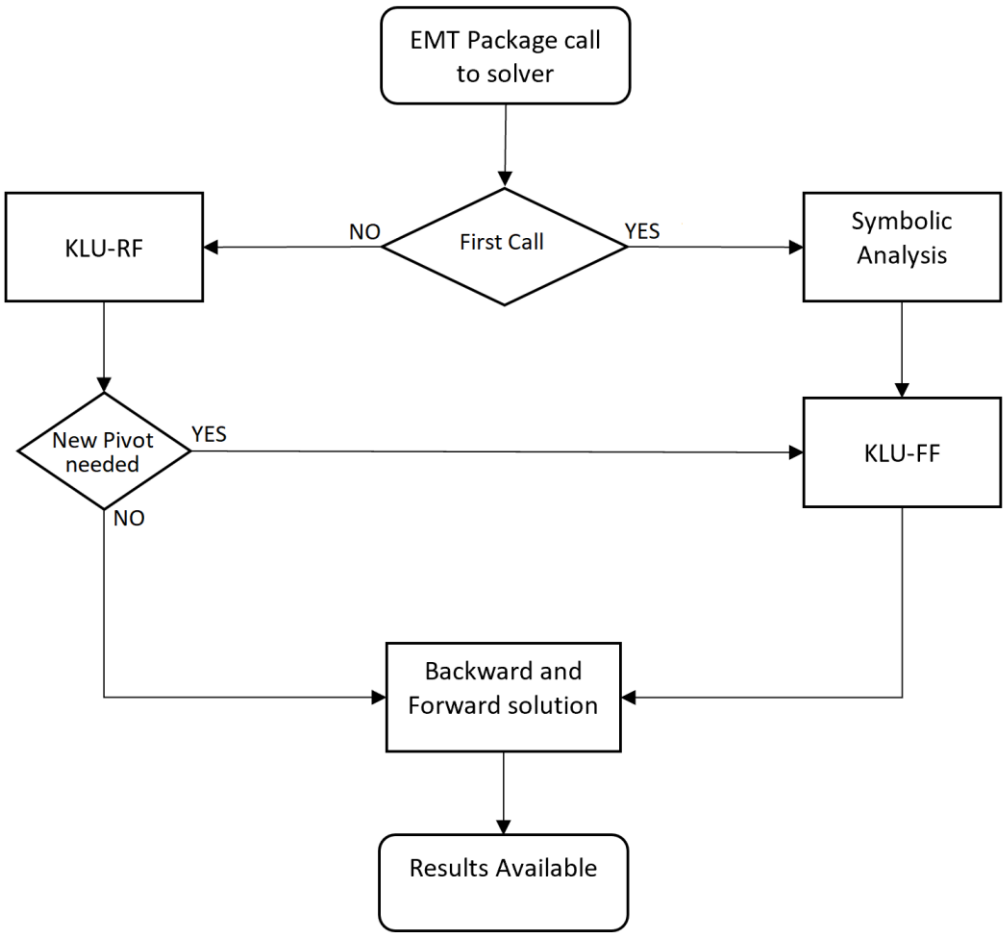

Figure 2.9 Pivot validity test flow chart

From the above figure, it can be seen that a KLU-FF is essentially needed at the beginning of simulation to calculate the non-zero patterns of $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ and finding the first pivoting order. Once the first solution of (2.1) is completed, KLU-RF becomes the default factoring algorithm used during KLU numerical stage. If an invalid pivot is spotted, the KLU-RF function is terminated for the block $\hat{\mathbf{A}}_i$ and a KLU-FF will start to calculate a new $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ non-zero pattern and pivot order.

It is important to highlight the fact that this feature allows to have a sort of hybrid factorization technique during the same time step solution. Given that BTF blocks are independent of each other, the fact that one block failed the pivot validity test does not necessary mean that all other blocks will fail the test. There could be a scenario where some blocks are updating their $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ numerical elements using KLU-RF technique and other blocks are calculating $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ from scratch using KLU-FF.

2.4 Partial factorization

A second feature added in this thesis to the KLU solver is called “Partial Factorization”. To reduce the computational cost of KLU-RF for $\hat{\mathbf{A}}_i$ even further, it is possible to apply partial KLU-RF. In a given system of equation (2.1), it is possible to determine the cells that are occupied by NMs and TVMs. Those dynamic cells may change between solution time-points and during iterations at a given time-point. These changes require the KLU-RF of $\hat{\mathbf{A}}_i$. A mapping can be derived to determine the BTF block number that contains each dynamic cell and the column number within BTF blocks that contains these cells.

$$f = \mathbf{A}_c \rightarrow \hat{\mathbf{A}}_i \tag{2.3}$$

$$\hat{\mathbf{A}}_c(i) = \mathbf{A}_c(\mathbf{P}_{c\_inv}(i)) \tag{2.4}$$

Let f in equation (2.3) be a mapping between $\mathbf{A}$ columns indices and $\hat{\mathbf{A}}$ column indices. This mapping is based on the column permutation vector $\mathbf{P}_c$ found during KLU symbolic analysis. The

conversion of column indices into BTF indices requires the calculation of the inverse column permutation vector $\mathbf{P}_{c\_inv}$.

$$\mathrm{M}:\mathbf{A}\rightarrow i \tag{2.5}$$

A similar mapping can be drawn between each matrix cell and the BTF blocks they belong in $\hat{\mathbf{A}}$. Let (2.5) be the mapping between matrix $\mathbf{A}$ nonzero elements and the BTF blocks i they belong to in $\hat{\mathbf{A}}$. Figure 2.10 shows the mapping procedure between these two sets. The mapping involved two nested loops that go over all matrix cells (the outer loop) and all BTF blocks (the inner loop). The outer loop runs from 1 to the total number of non-zeros (nnz) and passes column index of each cell to the inner loop. The inner loop runs from 1 to the number of blocks (nblocks) looking for the block the cell belongs to. In Figure 2.10, vector $\mathbf{R}$ represents block boundaries vector where $\mathbf{R}(i)$ is the starting row of block i and $\mathbf{R}(i+1)$ is the starting row number of block $i+1$, and vector $\mathbf{R}_{BTF}$ has the BTF block number of each non-zero element in $\mathbf{A}$.

Cells to BTF blocks mapping

```
for(i = 1; i <= nnz; i++)
{
        for (j = 1; j < nblocks; j++)
        {
                if(( Pc[Ac[i]] > R[j]) && (Ac[i] < R[j + 1]))
                {
                        RBTF[i] = j;
                }
        }
}
```

Figure 2.10 Cells to BTF blocks mapping

The matrix $\hat{\mathbf{A}}_i$ is reordered using AMD and can be written as

$$\hat{\mathbf{A}}_i=\begin{bmatrix}\mathbf{P}_{cc} & \mathbf{P}_{cd}\\ \mathbf{P}_{dc} & \mathbf{P}_{dd}\end{bmatrix} \tag{2.6}$$

Where, the subscripts c and d mean constant and dynamic respectively. The c columns do not have any dynamic parts, but the d columns contain at least one dynamic cell and may have zero or more dynamic cells in the following columns.

In the left-looking algorithm, the columns of $\hat{\mathbf{L}}_i$ are derived one-by-one by solving for each column of $\hat{\mathbf{A}}_i$. If, for example, the lower matrix decomposition of $\hat{\mathbf{A}}_i$ is stopped at its first dynamic column then

$$\hat{\mathbf{L}}_i^p = \begin{bmatrix} \mathbf{L_{cc}} & \mathbf{0} \\ \mathbf{P_{dc}} & \mathbf{I_{dd}} \end{bmatrix} \tag{2.7}$$

Where $\hat{\mathbf{L}}_i^p$ is a partially computed lower-triangular matrix, $\mathbf{L}_{cc}$ is a lower-triangular matrix, $\mathbf{P}_{dc}$ and $\mathbf{L}_{cc}$ contain the columns of the static part of $\hat{\mathbf{L}}_i^p$ and $\mathbf{I}_{dd}$ is the identity matrix. Once $\mathbf{P}_{dd}$ is determined (status of time-varying devices or iterative Norton equivalent) at a given solution time-point, the calculation process is continued until the replacement of $\mathbf{I}_{dd}$ to obtain $\mathbf{L}_i^{'}$ from $\hat{\mathbf{L}}_i^p$. The upper-triangular matrix $\mathbf{U}_i^{'}$ is calculated within the calculation process of $\mathbf{L}_i^{'}$. For (2.7), the partial upper matrix factorization is available up to the constant columns

$$\mathbf{U}_i^p = \begin{bmatrix} \mathbf{U_{cc}} \\ \mathbf{0} \end{bmatrix} \tag{2.8}$$

In the above approach it is not necessary to restart the partial KLU-RF process for the complete set of columns of $\mathbf{P}_{dd}$. Better efficiency can be gained, if partial KLU-RF is applied by restarting from the first left modified column j_{md} in $\mathbf{P}_{dd}$. As before, since KLU is a left-looking solver, all unchanged columns to the left of j_{md} can maintain the previous contributions to the $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ factors. In addition, given the fact that not all the elements in the right hand side vector $\mathbf{b}$ are dynamic, the forward substitution can start from the top changed element of $\mathbf{b}$ and the results of the skipped part can be retrieved from the previous iteration.

It is also possible to apply a permutation technique that forces $\mathbf{P}_{dd}$ to contain only NMs and TVMs (similar idea in [34]). But such an approach interferes with the AMD ordering and creates extra fill-ins which hinder the performance gains. It was tested and was not retained for this thesis.

The following example shows the application of partial factorization feature on the simple electric circuit shown in Figure 2.11.

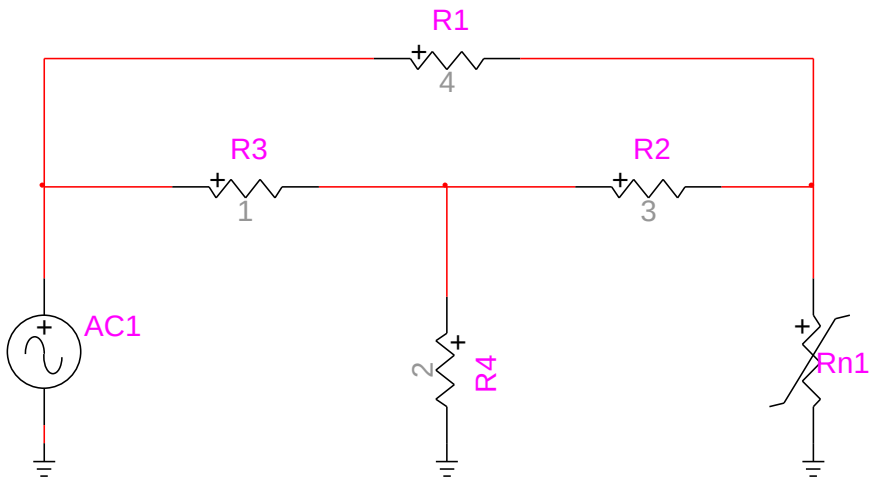

Figure 2.11 Sample circuit for demonstrating partial factorization

The BTF permuted MANA matrix for the circuit shown in the above figure is shown in equation (2.9). At the start of simulation, the nonlinear resistor Rn1 is equal to 1 Ohm (initial linear slope position) and the contribution of this resistance in the $\hat{\mathbf{A}}$ matrix appears at the diagonal element $\hat{\mathbf{A}}(3,3)$.

$$\begin{bmatrix} 1.2500 & -1.0000 & -0.25000 \\ -1.000 & 1.83333 & -0.33333 \\ -0.2500 & -0.33333 & 1.583333 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} I_{S1} \\ 0 \\ 0 \end{bmatrix} \tag{2.9}$$

The KLU-FF of system (2.9) is performed in equations (2.10) to (2.19). The following steps demonstrate a summarized KLU-FF process (for detailed procedure of KLU-FF refer to section 1.6.2.2).

1. KLU 1st column factorization:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1.2500 \\ -1.000 \\ -0.2500 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1.2500 \\ -1.000 \\ -0.2500 \end{bmatrix} \tag{2.10}$$

$$\hat{\mathbf{L}}_1 = \begin{bmatrix} 1 & 0 & 0 \\ -0.8000 & 1 & 0 \\ -0.2000 & 0 & 1 \end{bmatrix} \tag{2.11}$$

$$\hat{\mathbf{U}}_1 = \begin{bmatrix} 1.2500 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

2. KLU 2nd column factorization:

$$\begin{bmatrix} 1.0000 & 0 & 0 \\ -0.8000 & 1.0000 & 0 \\ -0.2000 & 0 & 1.0000 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.0000 \\ 1.83333 \\ -0.3333 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.0000 \\ 1.03300 \\ -0.53300 \end{bmatrix} \tag{2.13}$$

$$\hat{\mathbf{L}}_2 = \begin{bmatrix} 1 & 0 & 0 \\ -0.8000 & 1 & 0 \\ -0.2000 & -0.5161 & 1 \end{bmatrix} \tag{2.14}$$

$$\hat{\mathbf{U}}_2 = \begin{bmatrix} 1.2500 & -1.000 & 0 \\ 0 & 1.0333 & 0 \\ 0 & 0 & 1.000 \end{bmatrix} \tag{2.15}$$

3. KLU 3rd column factorization:

$$\begin{bmatrix} 1.0 & 0 & 0 \\ -0.8000 & 1.0 & 0 \\ -0.2000 & -0.5161 & 1.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.2500 \\ 0.33333 \\ 1.58333 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.2500 \\ 0.13333 \\ 1.60214 \end{bmatrix} \tag{2.16}$$

$$\hat{\mathbf{L}}_3 = \begin{bmatrix} 1.0 & 0 & 0 \\ -0.8000 & 1.0 & 0 \\ -0.2000 & -0.5161 & 1.0 \end{bmatrix} \tag{2.17}$$

$$\hat{\mathbf{U}}_3 = \begin{bmatrix} 1.25000 & -1.000 & -0.250 \\ 0 & 1.0333 & 0.1333 \\ 0 & 0 & 1.6021 \end{bmatrix} \tag{2.18}$$

The fully factorized system of (2.9) is shown in (2.19)

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.8 & 1 & 0 \\ -0.2 & -0.5161 & 1 \end{bmatrix} \begin{bmatrix} 1.25 & -1 & -0.25 \\ 0 & 1.0333 & -0.533 \\ 0 & 0 & 1.257 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} I_{S1} \\ 0 \\ 0 \end{bmatrix} \tag{2.19}$$

Once the system in (2.19) is solved and the simulation moves to the next time step, the nonlinear resistor in Figure 2.11 may change to another value. In this example and for the sake of illustrating

the partial KLU-RF concept, it is assumed that Rn1 value changes from 1 ohm to 2 Ohms. The MANA matrix in (2.9) changes and it becomes as shown in (2.20).

$$\begin{bmatrix} 1.2500 & -1.000 & -0.2500 \\ -1.000 & 1.8333 & -0.3333 \\ -0.2500 & -0.33 & 1.08333 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} I_{S1} \\ 0 \\ 0 \end{bmatrix} \tag{2.20}$$

A comparison between (2.9) and (2.20) shows that the change in Rn1 only affects element $\hat{\mathbf{A}}(3,3)$ and hence column and row 3 only. The partial KLU-RF feature can detect the first left change column *(FLCC)* in $\hat{\mathbf{A}}_i$ and starts the factorization process from that changed column. In this example, the first left change column is column number 3. In order to factorize this column successfully, the lower and upper matrices resulted from the factorization of (2.9) up to the second column (shown in equations (2.14) and (2.15) are retrieved. The factorization of (2.20) can be achieved by only factoring the third column with the new element at $\hat{\mathbf{A}}(3,3)$.

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.8000 & 1 & 0 \\ -0.2000 & -0.5161 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.2500 \\ 0.33333 \\ 1.08333 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.2500 \\ 0.13333 \\ 1.10213 \end{bmatrix} \tag{2.21}$$

$$\hat{\mathbf{L}} = \begin{bmatrix} 1.0 & 0 & 0 \\ -0.8000 & 1.0 & 0 \\ -0.2000 & -0.5161 & 1.0 \end{bmatrix} \tag{2.22}$$

$$\hat{\mathbf{U}} = \begin{bmatrix} 1.2500 & -1.000 & -0.2500 \\ 0 & 1.0333 & -0.5333 \\ 0 & 0 & 1.10213 \end{bmatrix} \tag{2.23}$$

$$\begin{bmatrix} 1.0 & 0 & 0 \\ -0.8000 & 1.0 & 0 \\ -0.2000 & -0.5161 & 1.0 \end{bmatrix} \begin{bmatrix} 1.2500 & -1.000 & -0.2500 \\ 0 & 1.0333 & -0.5333 \\ 0 & 0 & 1.10213 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} I_{S1} \\ 0 \\ 0 \end{bmatrix} \tag{2.24}$$

From the above example, it can be seen that the partial factorization process allows to save computing time since it avoids the lengthy operations for recalculating the full $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ matrices from scratch. The computational impact will depend effectively on the locations of the nonlinear component columns. The Impact is maximized when all nonlinear component cells are located at the far-right part of $\hat{\mathbf{A}}_i$. It is also obvious that the impact is more important for very large-scale systems with nonlinear components. It is worth mentioning that in the above example the diagonal

pivots assumed at the beginning of KLU-FF continue to be valid throughout the 3 columns factorization. Applying partial KLU-RF involved continued validation of pivot for each column being factorized, and if at any point an invalid pivot is found the partial KLU-RF process is halted and a KLU-FF is performed on that particular block.

2.5 Parallel KLU Implementation

The second approach of achieving faster EMT simulation is by applying parallel computation on the enhanced version of KLU presented above. In this project the OpenMP multithreading technique is applied on different parts of KLU such as KLU-FF, KLU-RF and forward-backward substitution.

Throughout this thesis, two different parallel sparse solver techniques were implemented and tested. The two approaches are presented in this section.

2.5.1 Shared memory Model

The shared memory design of OpenMP is mainly based on keeping the matrix $\mathbf{A}$ received by the simulation package (i.e EMTP) as one matrix that is allocated on one sequential segment of the memory and using this matrix in the solution of (1.4). All BTF blocks in this model are concatenated in one matrix and accessing these blocks requires the knowledge of the starting and ending column/row of each block.

In KLU, the symbolic analysis is done in a pure sequential fashion due to the fact that it is done only once at the beginning of simulation ($t = 0$). However, when it comes to numerical analysis, parallelizing the solution of different blocks is essential to convert the KLU code to a parallel solver, given that the network that is being solved can be solved in parallel because it has at least one delay-based line model in it. KLU-FF process can be done in parallel allowing the Full-factorization of different blocks to be done simultaneously. similarly, KLU-RF and the backward-forward substitution steps can each be done in parallel as well.

In the shared memory model, parallelizing the factoring process (KLU-FF and KLU-RF) was done by surrounding the factorization loop, that loops over BTF blocks, with a pragma bracket that will guarantee a parallel execution of that loop. In a similar manner the backward/forward substitution

step can be parallelized. Using shared memory model requires the distinguishing between thread specific variables and threads shared variables and the proper distribution of blocks on different threads. This concept it crucial to avoid any overlap between different threads, and to avoid any kind of race conditions during OpenMP thread synchronization.

Figure 2.12 shows a flow chart of the shared memory OpenMP design. From the flow chart it can be seen that the three parallel regions in this OpenMP model are launched and joined locally within their SMPEMT functions. These three regions are defined by two black bold horizontal lines that represent the launch and join points of threads. For example, in KLU-FF function, OpenMP launches threads at the beginning of KLU-FF loop and joins them when the last block is fully factored. The same concept applies for the KLU-RF and backward and forward substitution functions. The KLU-FF and KLU-RF can run on the same thread and three parallel regions are using threads that are launched and kept for further usage in an OpenMP thread pool.This process of launching and joining threads at different locations within the solver increases threading overhead and introduces further delays in the computation speed with the increased number of threads.

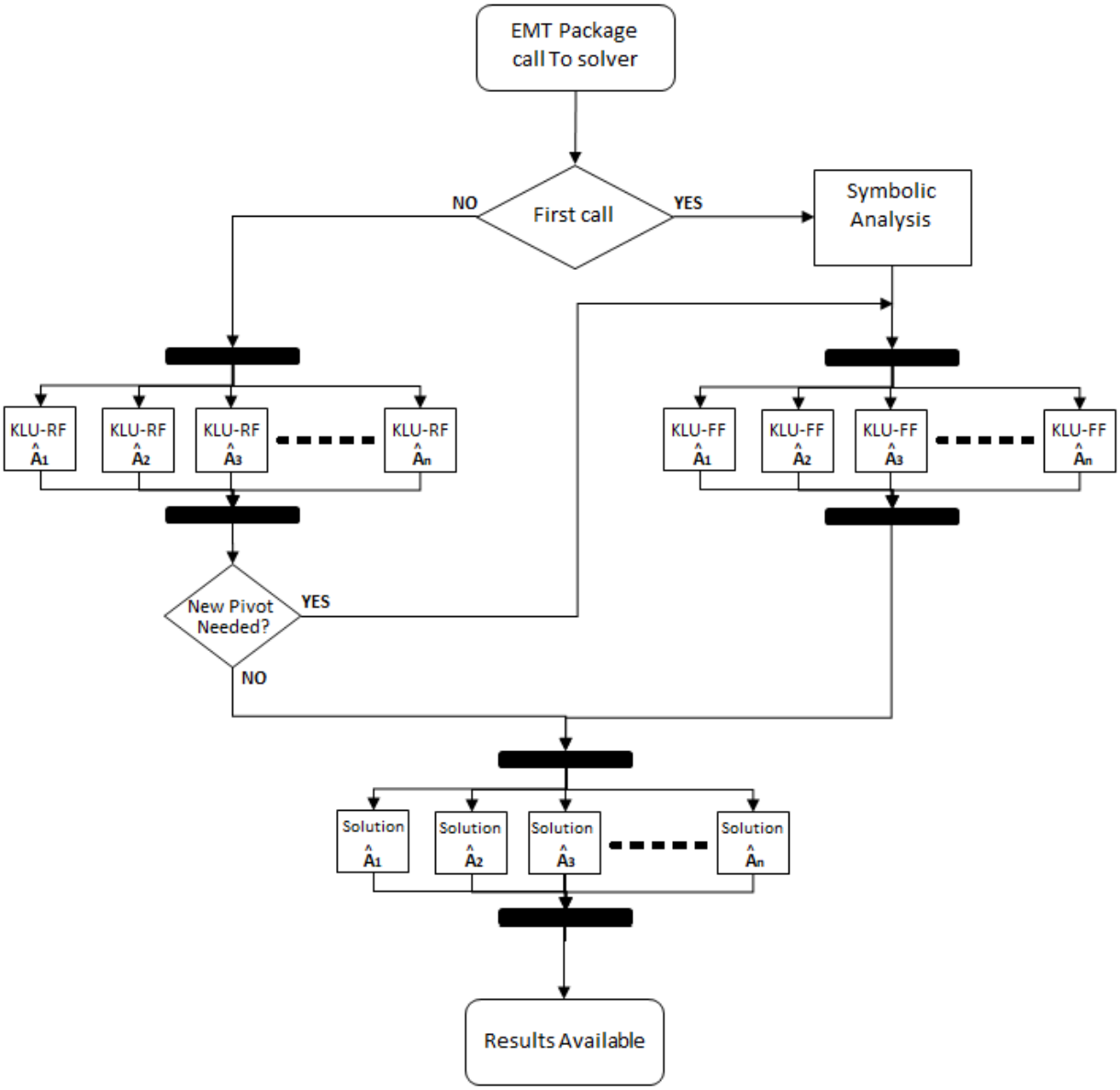

Figure 2.12 Shared memory OpenMP model

2.5.2 Distributed Memory Model

The second design of OpenMP implementation is based on the distributed memory concept. This model uses matrix **A** to create another set of matrices that are fully independent in terms of equations and memory storage. These new matrices are created based on the BTF permutation found during the symbolic analysis of **A**. A new data type is created to fully represent the new matrices both symbolically and numerically. Figure 2.13 shows the user defined type *KLU_unit*

used to represent different sub-matrices. If the user of SMPEMT solver requested the launch of 4 threads, and the size of the matrix $\mathbf{A}$ and the circuit being solved support this number of threads, four instances of KLU_unit will be created with each one of them representing part of $\mathbf{A}$.

```
/* ------------------------------------------------------------------------ */
/* klu_units: sets default parameters for each thread */
/* ------------------------------------------------------------------------ */
typedef struct
{
      klu_common klu_unit_common;
      klu_numeric *klu_unit_numeric;
      klu_symbolic *klu_unit_symbolic;
      int n;
      int nnz;
      double *klu_unit_Ax;
      int *klu_unit_Ai;
      int *klu_unit_Ap;
      double *klu_unit_X;
      double *klu_unit_B;
      int *klu_unit_work;
      int *row_reverse_mapping;
      ij_cells *changed_cells;
      int *A_EMTP_cell_number;
      int Changed_cells_count;
      int Call_counter;
      int klu_unit_flag;
} klu_unit;
```

Figure 2.13 KLU_unit type declaration

In Figure 2.13, klu_unit_common is a variable of type KLU_Common that stores KLU control parameters of the submatrix represented by KLU_UNIT, klu_unit_symbolic is a variable of type KLU_Symbolic that stores the symbolic parameters of the submatrix represented by KLU_UNIT (such as permutation vectors and different statistics variables), klu_unit_numeric is a variable of type KLU_Numeric that stores the numerical quantities of the submatrix represented by KLU_Unit (such as the matrices $\hat{\mathbf{L}}_i$, $\hat{\mathbf{U}}_i$ and the solution vector $\hat{\mathbf{x}}$). The variables n, nnz, klu_unit_Ax, klu_unit_Ai, klu_unit_Ap, klu_unit_X and klu_unit_B are a representation of the submatrix being represented by KLU_UNIT. All other variables in Figure 2.13 are used to provide two ways mapping between KLU_UNIT submatrix and the matrix $\mathbf{A}$.

Solving the system of equation in (1.54) using the distributed memory algorithm requires running symbolic analysis on the matrix $\mathbf{A}$. This symbolic analysis will find the BTF permutation of $\mathbf{A}$ and determine the load balancing to achieve best parallelization possible (see section 2.6). A new

function called KLU_submatrix_creation was created to form the new matrices and allocate all necessary memory required to store KLU_unit elements. The creation of KLU_unit instances is done in parallel in order to make sure that each thread uses its own memory and hence its own cache line to store and manipulate data. This practice allows to minimize thread conflicts and enhances the ability of each thread to fetch data faster and more efficiently.

The distributed memory algorithm is shown below:

1. First call to SMPEMT:
    a. Perform symbolic analysis on $\mathbf{A}$.
    b. Create new matrices $\hat{\mathbf{A}}_i$ by concatenating BTF blocks of $\mathbf{A}$.
    c. Launch a parallel SMPEMT call for all KLU_unit objects
    d. Go to step 2
2. First KLU_unit call to SMPEMT:
    a. Perform symbolic analysis
    b. Perform KLU-FF to find $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$.
    c. Backward and Forward substitution to find $\hat{\mathbf{x}}_i$.
    d. Go to step 4.
3. Not First KLU_unit call to SMPEMT:
    a. Perform KLU-RF using existing nonzero pattern and pivoting with pivot validity testing
        i. Invalid pivot found: go to step 2.b
    b. Go to step 4.
4. Copy the values in $\hat{\mathbf{x}}_i$ back to $\mathbf{x}$.

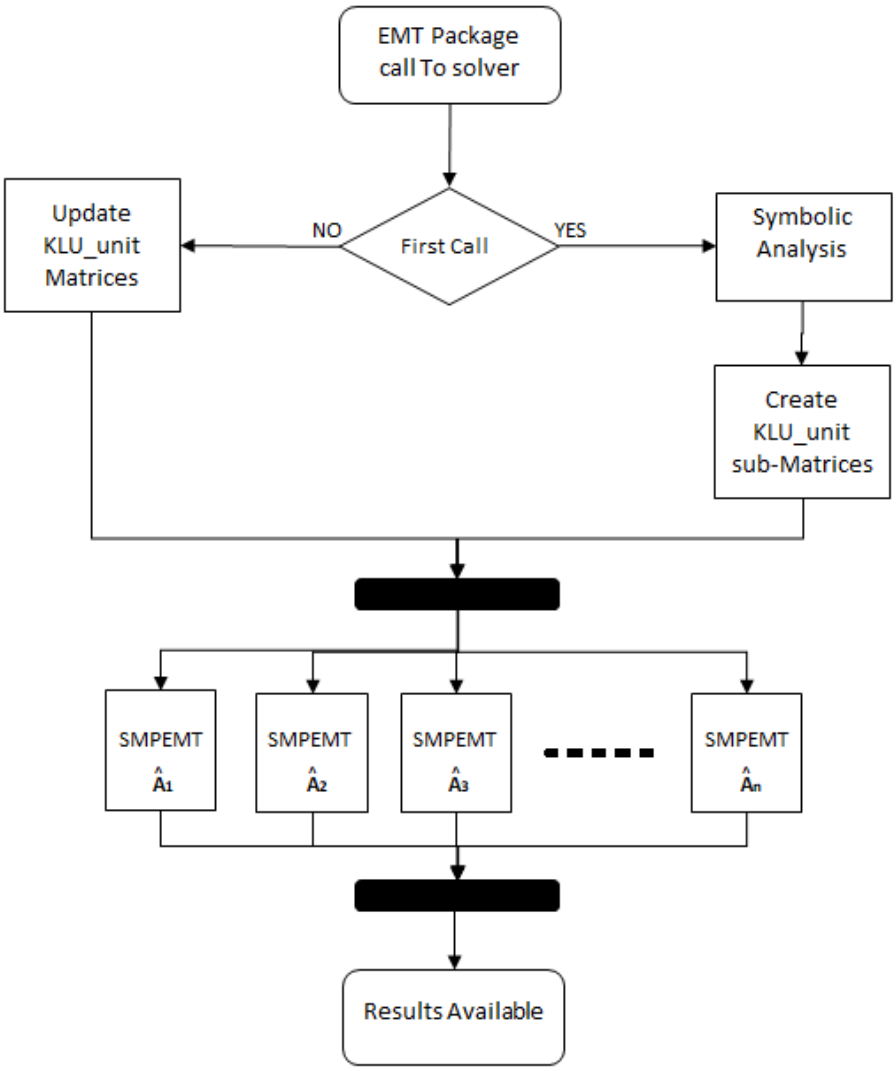

Figure 2.14 Distributed model OpenMP design

In this thesis, two solvers will be used that are based on SMPEMT, namely SMPEMT1 and SMPEMT2. The first solver (SMPEMT1) has only the pivot validity test implemented in it; whereas SMPEMT2 has all features discussed above (pivot validity test and partial factorization) implemented. This practice allows for better understanding of the effect of each feature on different test benchmarks and gives more insight of the advantages and disadvantages of all added feature.

2.6 Load balancing

Parallelization is applied to blocks found from BTF permutation, that is for each $\hat{\mathbf{A}}_i$ matrix. Since there is a limited number of CPU cores and the computing gains are limited by the largest network blocks, it is necessary to apply a balancing technique for the given number of cores. An algorithm that is based on different approaches has been implemented.

In the first approach a pre-programmed method allows to estimate the number of floating-point operations for the solution of each block ($NFPO_i$). The following formula is used for a matrix block of $n \times n$:

$$NFPO_i = \sum_{j=1}^{n} \left[\sum_{m=1}^{j-1} 2Llen(m) + 3Llen(j) + 2Ulen(j) \right] + n \tag{2.25}$$

Where *j* and *m* are the indices of $\hat{\mathbf{A}}_i$ columns, *Llen* and *Ulen* are the counts of non-zeros in $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$ respectively. This formula accounts for the LU-factorization based on the initial $\hat{\mathbf{L}}_i \hat{\mathbf{U}}_i$ nonzero patterns. It also accounts for the backward-forward substitution step. Equation (2.25) was derived by considering the operations of KLU solvers factorization line by line and accounting for any step to produce accurate and efficient load balancing.

The above equation consists of two nested summations, the outer summation of (2.25) goes through all the block's columns while the inner summation calculates the floating points operations needed to KLU-RF column j and solving it. The main tasks equation (2.25) accounts for are calculating the solution for a sparse lower triangular system used to find $\hat{\mathbf{L}}_i$ and $\hat{\mathbf{U}}_i$, finding the numerical elements of $\hat{\mathbf{L}}_i$ and backward and forward solution. As can be seen in section 1.6.2, the results of solving a sparse lower triangular system for column j requires the use of nonzero elements in the columns prior to j ($m < j$) in $\hat{\mathbf{L}}_i$ and that accounts for the term $\sum_{m=1}^{j-1} 2Llen(m)$ in equation (2.25); whereas the calculation of $\hat{\mathbf{L}}_i$ numerical values contributes by $Llen(j)$ floating points operations as seen in the last line of Figure 1.25 pseudocode. Finally, the calculation of backward and forward substitution costs $2Llen(j) + 2Ulen(j) + n$ floating points operations.

In the second approach, the number of non-zeros in each block (NNZ_i) is available from its nonzero pattern. However, for all the test cases presented in this paper, using NNZ_i was less

efficient than using $NFPO_i$.

The blocks are assigned to cores using the number of available cores (N_C) and the factor $k_d = NFPO / N_C$, where $NFPO$ is the total number for the entire system of equations. Since the number of blocks *N* could be higher than N_C , it is necessary to verify the limiting k_d for each assignment. If a given core is assigned a block with $NFPO_i$ less than k_d then it can contain additional blocks until k_d is reached or exceeded. This is basically a packing procedure for populating available cores.

If a block's $NFPO_i$ falls below a minimum size, then it must be packed into an assigned core since threading for such a block can become inefficient. The same is applicable using NNZ_i instead of $NFPO_i$.

CHAPTER 3 TESTING AND RESULTS

In this chapter, different cases with different sizes and topologies are tested and validated. The new implemented solvers SMPEMT1 and SMPEMT2 are used in addition to the solver already exists in EMTP (EMTP-MDO). The new solvers are tested with single thread and multithread in order to validate the performance under all circumstances and scenarios.

When it comes to EMT simulation, speed is not the most important factor to look at. The accuracy of simulation results must be fully maintained in the new implemented solver under both single threaded and multithreaded execution. The accuracy of SMPEMT1/2 was verified for all benchmarks used in this chapter by calculating the difference error percentage between SMPEMT1/2 and EMTP-MDO waveforms. The percentage error has been calculated between the two sets of results using equation (3.1).

$$e\% = \frac{\left\| \tilde{f} - f \right\|^2}{\left\| f \right\|^2} \tag{3.1}$$

where:

$e\%$: percentage error between SMPEMT1/2 and EMTP-MDO

$\tilde{f}$: results vector produced by SMPEMT1/2 solver

f : results vector produced by EMTP-MDO solver

In addition to the above quantitative measure, few signals of each test case were used to compare the results of both solvers visually. These signals produced by both solvers were plotted and overlapped to visually realize any differences along the simulation period.

SMPEMT1 and SMPEMT2 solver ability to provide simulation acceleration and flexibility to different EMT cases can clearly be seen herein. In the following few sections further validations of the new proposed solver is given with emphasis of the main advantages and the few limitations the solver has.

3.1 SMPEMT testing and validation

The modified KLU solver named SMPEMT was tested on a wide range of cases and benchmarks. The aim was to test the developed new solvers on realistic power grid cases. In addition, different scenarios were considered to stress numerical limitations and examine solver stability and accuracy. These scenarios involve faults, large numbers of nonlinear models and the use of wind generators with power electronics converters. In addition, the distributed memory design of OpenMP was used in all cases and has been validated.

In order to draw a clear conclusion about each test case and fully understand each scenario the following is given for each benchmark:

- A brief description of the case.
- A listing of the benchmark main components.
- A plot of the network sparsity pattern before and after BTF permutation.
- Simulation timing results for EMTP, KLU, SMPEMT1 and SMPEMT2.
- Simulation acceleration plot (both in seconds and acceleration gain).
- Results description and discussion.

All tests were run on a machine that has the specifications listed in Table 3.1.

Table 3.1 Testing platform

Test Platform: HP DL360	
Processor Model	Intel Xeon CPU E5-2650 v4
CPU frequency	2.20 GHz
Number of physical processors	12 / cluster
Number of logical processors	24 / cluster
Memory	32.0 GB
Windows	10

3.1.1 Hydro-Quebec Full network (HQ-L)

This test case is an upgraded version of the one presented in [6]. It is based on the actual Hydro-Quebec grid (HQ-grid). A top view of the test case is presented in Figure 3.1.

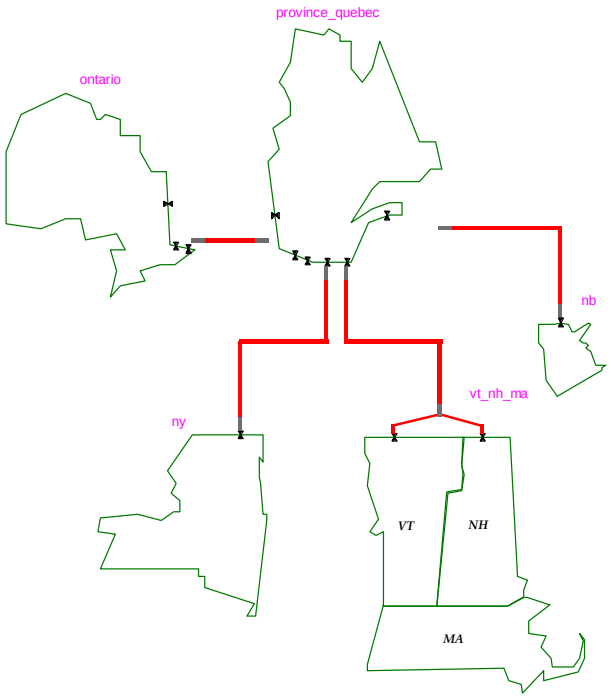

Figure 3.1 Hydro-Quebec case top view

The summary of the case main components is:

- RLC branches: 27530;
- PI/RL coupled branches, 3-phase: 860
- CP Lines/Cable: 1354 phases
- Ideal transformer units (for 3-phase transformers): 6294
- Ideal switches: 3663
- Zinc-Oxide Arresters: 174;
- Nonlinear inductances (transformer magnetization): 4452
- Synchronous generators (with excitors and governors): 349;

- Loads: 4452

HQ-L Simulation data:

- Simulation time: 1 s
- Simulation time step: 50 μs
- Pivot tolerance ε_p : 0.01
- Average number of iterations per time step: 2.07
- Total number of iterations: 41400
- Matrix $\mathbf{A}$ size: 41797×41797
- Number of nonzero elements (nnz) in $\mathbf{A}$: 169369
- Sparsity percentage: 99%
- Total number of BTF of Blocks: 217
- Biggest block size: 2898×2898
- Smallest block size: 3×3

The sparsity pattern of HQ-L network matrix $\mathbf{A}$ is shown in Figure 3.2, and the BTF version of the matrix is shown in Figure 3.3.

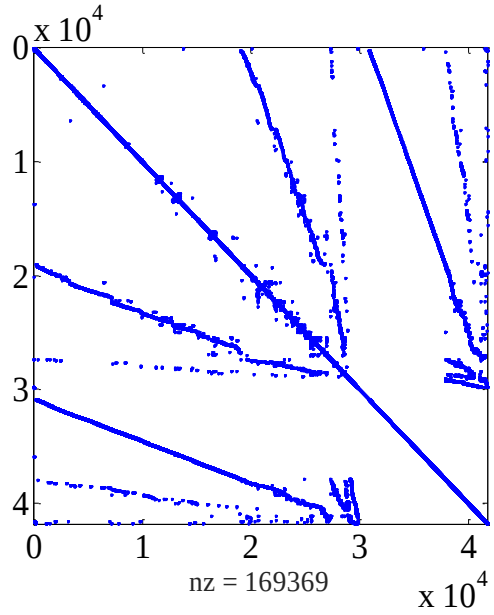

Figure 3.2 HQ-L matrix **A** before BTF

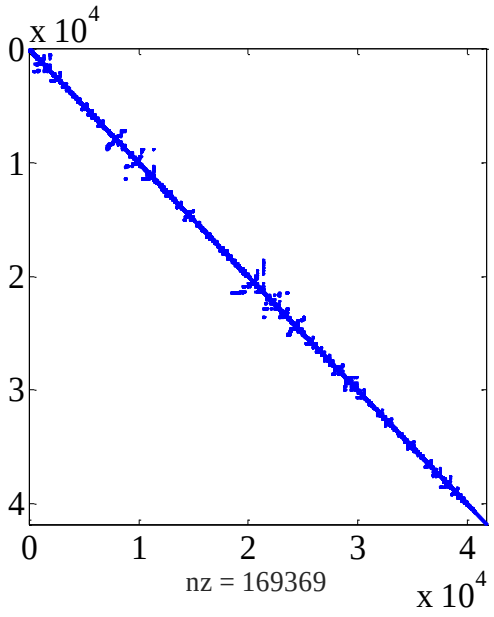

Figure 3.3 HQ-L matrix **A** after BTF

shows the solution of equation (1.4) timing using different solvers and different number of threads. It is apparent that the KLU method alone does not have performance gains as seen in section 2.1. This is due to many reasons including the heavy computation operations used in KLU factorization process and the fact that KLU-FF is applied to all blocks without the improvements of SMPEMT1

and SMPEMT2. For this case, SMPEMT2 and SMPEMT1 gave very close timings since some dynamic elements can be found in the far-left segment of the matrices $\hat{\mathbf{A}}_i$.

Table 3.2 HQ-grid sparse matrix solution timings for 1s simulation and $\Delta t = 50\,\mu s$

	Number of cores								
Solver	**1**	**2**	**4**	**8**	**12**	**13**	**14**	**15**	**16**
EMTP	1048								
KLU	1216								
SMPEMT1	296	133	82	47	34	36	37	38.5	39
SMPEMT2	285	126	77	43	31	32	32.5	32.5	34

The computational gain against existing EMTP solver is 1048/31=33.8 with 12 cores. The gain over the standard single-core KLU solver is 1216/31=39 with 12 cores.

Performance plots are presented in Figure 3.4. The maximum gain over the single core SMPEMT2 version is 9.2 and there are no significant gains after 12 cores. This is mainly due to the limitation imposed by the largest block, increased memory exchange and thread management costs which increase with the number of threads. The overall computation time including the solution of equation (2.1), the control solution, steady-state solution and updating matrix $\mathbf{A}$ and vector $\mathbf{b}$ drops from 1976 seconds (when using EMTP solver) to 404 seconds (when using SMPEMT2, parallel control solver and 16 threads topology).

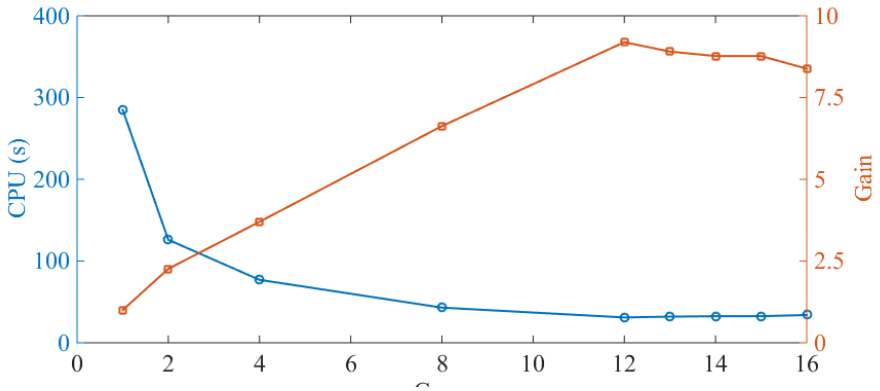

Figure 3.4 SMPEMT HQ-L Grid simulation time and gain

In order to validate the results of SMPEMT, three signals were selected to determine the accuracy of the solution. The first selected signal is the voltage (phase A) drop across line L7016 located in the province of Quebec and it was chosen in particular due to its distance proximity to a fault that

is located between L7016 and L 7046A transmission lines. The fault event in this test case is a (3-phase-to-ground) fault that is triggered at t = 0.5s as can be seen in Figure 3.5.

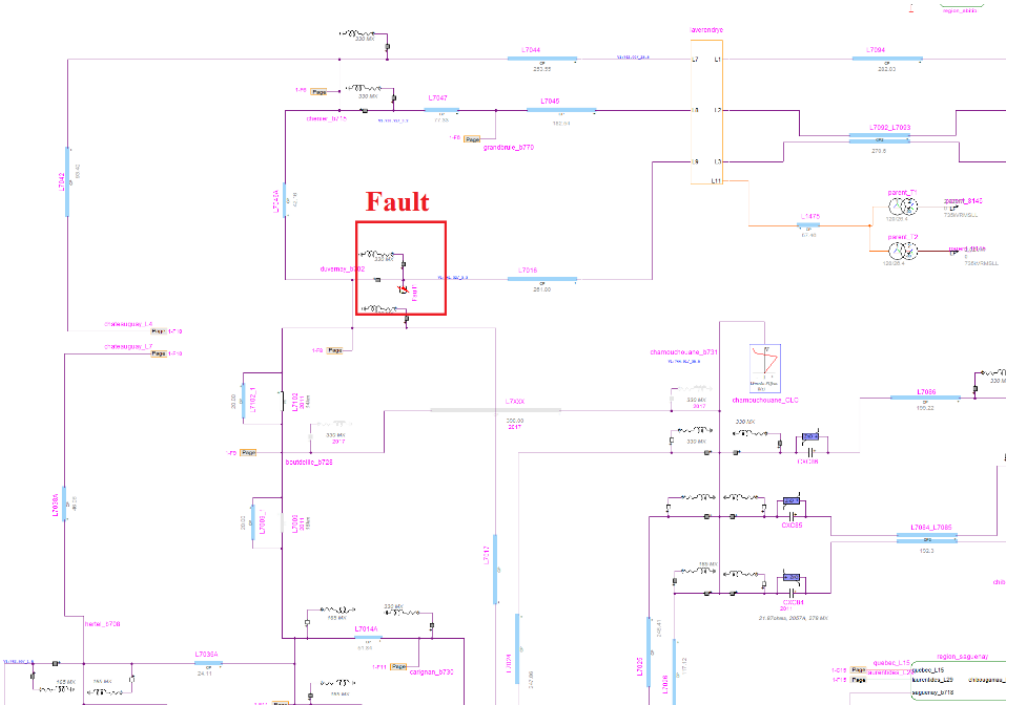

Figure 3.5 HQ-L Grid fault location

The second signal is the real power of synchronous machine Generator Mercier_A1 located in the province of Quebec in the Laurentides region, and the third signal is the real power of the synchronous machine Hydrocanyon_A located in the province of Quebec in the Quebec City region. The comparison of the three signals are shown in Figure 3.6 to Figure 3.8. In these figures, the red waveform represents the result of EMTP-MDO solver, and the blue is SMPEMT solver. It can be seen from the figures that both results are matching and a complete overlap between the two curves is achieved (including during the fault effect period). The error percentage between both set of waveforms are found to be 2.67×10^{-9}, 2.93×10^{-10} and 1.57×10^{-9} for the three signals respectively.

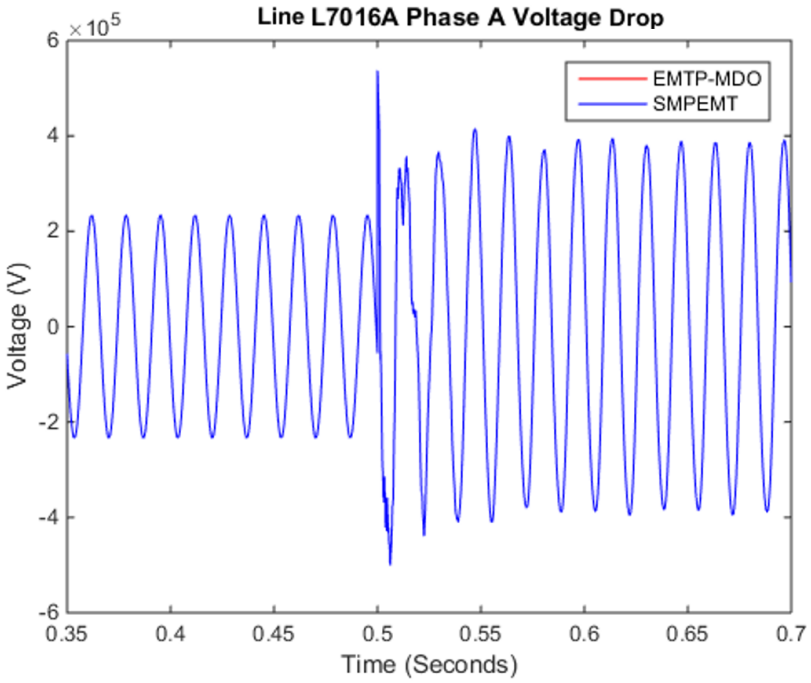

Figure 3.6 HQ-L grid line L7016 voltage drop - phase A

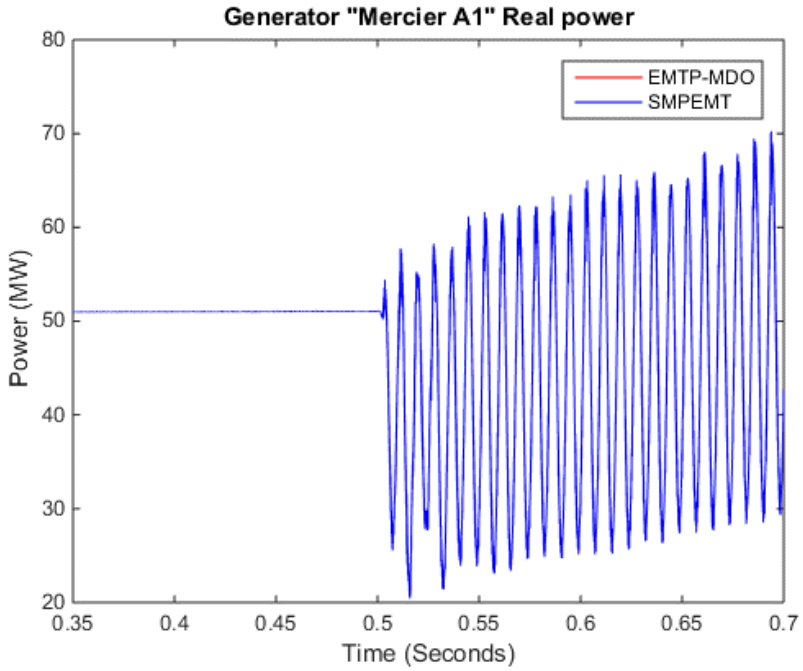

Figure 3.7 HQ-L grid Generator Mercier_A1 real power

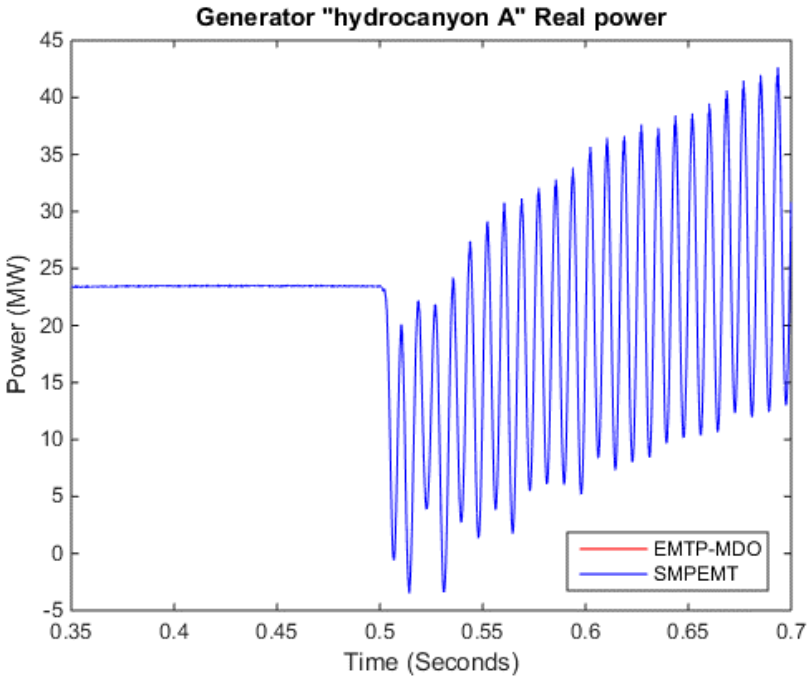

Figure 3.8 HQ-L grid Generator Hydrocanyon_A real power

3.1.2 T0-Grid

This case is a realistic 400 kV, 50 Hz network. It is designed with high integration of renewable sources to stress numerical accuracy and stability. It includes 72 synchronous generators modeled with their exciters and governors. There is a total of 10 wind parks with aggregated wind generators. These generators of DFIG type are simulated with their controllers that are based on reactive power control mode. The DFIG converters are given two modeling options: Detailed model (DM) and average value model (AVM) [58]. The DM includes nonlinear IGBT models which require iterations and significantly increases computational burden. In the AVM controlled sources are used to represent average converter behavior and sufficient accuracy can be achieved when studying grid performance issues. The details of this benchmark are listed in [59].

The top view of T0-Grid is shown in Figure 3.9, where the green boxes represent sub-transmission networks at 154 kV with wind generation, and the yellow boxes represent only sub-transmission networks with no wind turbines. In addition to the above, the network has the following contents:

- RLC branches: 2319; PI/RL coupled branches, 3-phase: 595

- CP Lines/Cable: 174 phases
- Ideal transformer units (for 3-phase transformers): 6294
- Controlled switches (converter switches): 190
- Ideal switches: 254
- Nonlinear resistances (used for IGBT models): 270
- Nonlinear inductances (transformer magnetization): 564
- Loads: 1029

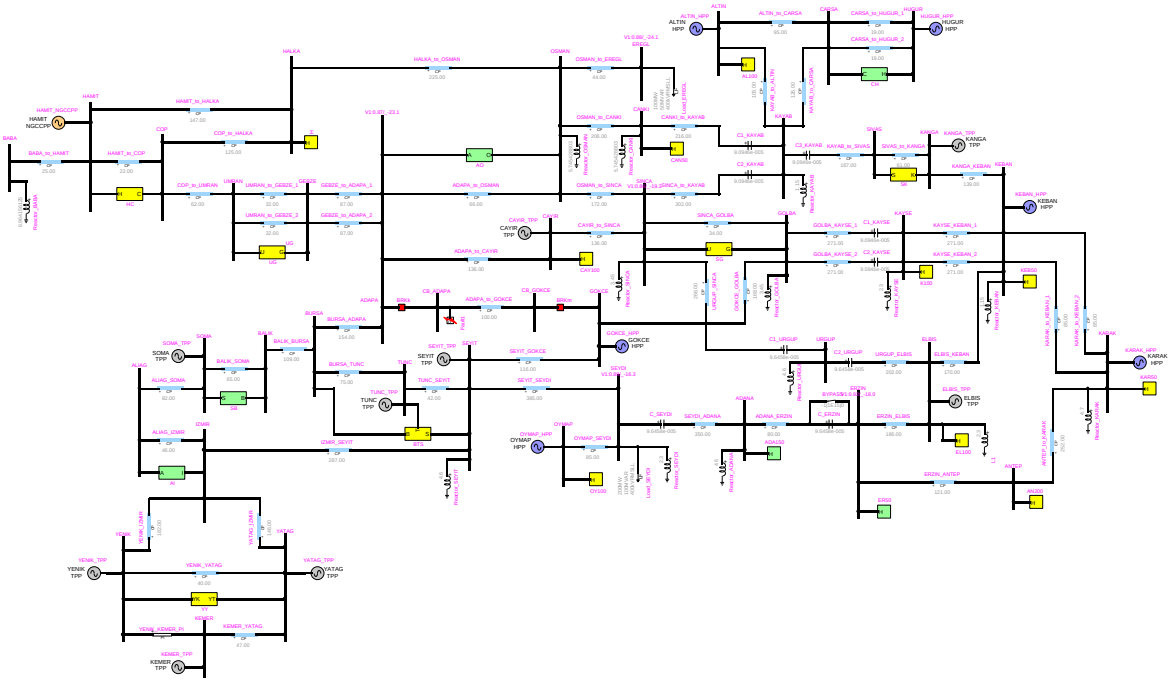

Figure 3.9 T0-Grid top view

T0-Grid Simulation data

- Simulation time: 1 s
- Simulation time step: 10 µs
- Pivot tolerance ε_p : 0.01
- Average number of iterations per time step: 6.23

- Total number of iterations: 799174
- Matrix $\mathbf{A}$ size: 4703×4703
- Number of nonzero elements (nnz) in $\mathbf{A}$: 25117
- Sparsity percentage: 99%
- Total number of BTF of Blocks (nblocks): 28
- Biggest block size: 573×573
- Smallest block size: 3×3

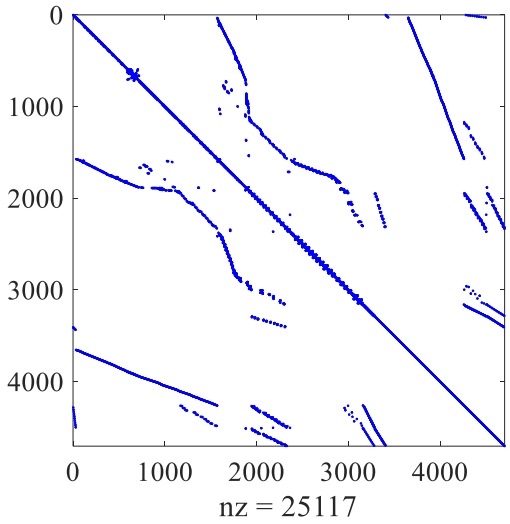

Figure 3.10 T0-Grid matrix $\mathbf{A}$ before BTF

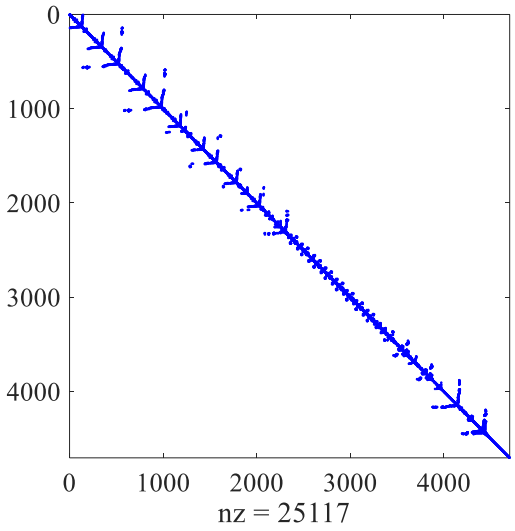

Figure 3.11 T0-Grid matrix $\mathbf{A}$ after BTF

Table 3.3 T0-DM sparse matrix solution timings for 1s simulation and $\Delta t = 50\,\mu s$

	Number of cores								
Solver	**1**	**2**	**4**	**8**	**12**	**13**	**14**	**15**	**16**
EMTP	1241								
KLU	2120								
SMPEMT1	720	380	229	151	157	157	161	161	165
SMPEMT2	675	360	210	141	99	99	101	105	112

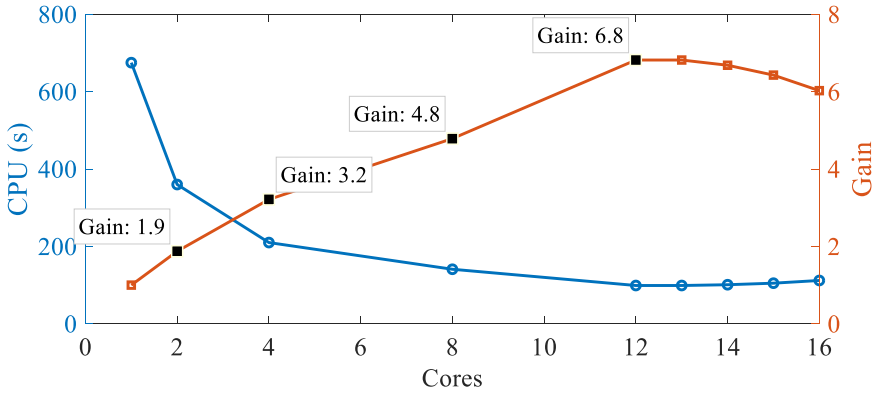

Figure 3.12 SMPEMT T0-Grid simulation time and gain for DM model

shows the solution of equation (1.4) timing using different solvers and different number of threads. A gain of 1241/151=8.2 is recorded over EMTP when SMPEMT1 is used with 8 threads and no

further gain is noticed with the increase number of threads. This is mainly due to the largest block that imposes limitation on further distribution of computation loads on additional threads, and acts as the bottle nick that takes the biggest computation time and forces all other threads to perform a busy wait while its computation is being finalized. However, a gain of 1241/99=12.5 was recorded when SMPEMT2 is used. This difference between the two solvers (SMPEMT1 and 2) is mainly due to the usage of partial factorization and the location of the first left dynamic column (FLDC). The gain of SMPEMT1 with 8 threads is 720/151=4.7 compared to SMPEMT1 with 1 thread, while SMPEMT2 achieved 675/99=6.8 with 12 threads compared to 1 thread.

The overall computation time including the solution of equation (2.1), the control solution, steady-state solution and updating matrix $\mathbf{A}$ and vector $\mathbf{b}$ drops from 2943 seconds (when using EMTP solver) to 578 seconds (when using SMPEMT2, parallel control solver and 16 threads topology).

The studied event in this test case is a (phase-a-to-ground) fault on the transmission line ADAPA_to_GOKCE connected between the lines ADAPA and GOKCE as can be seen in Figure 3.13.

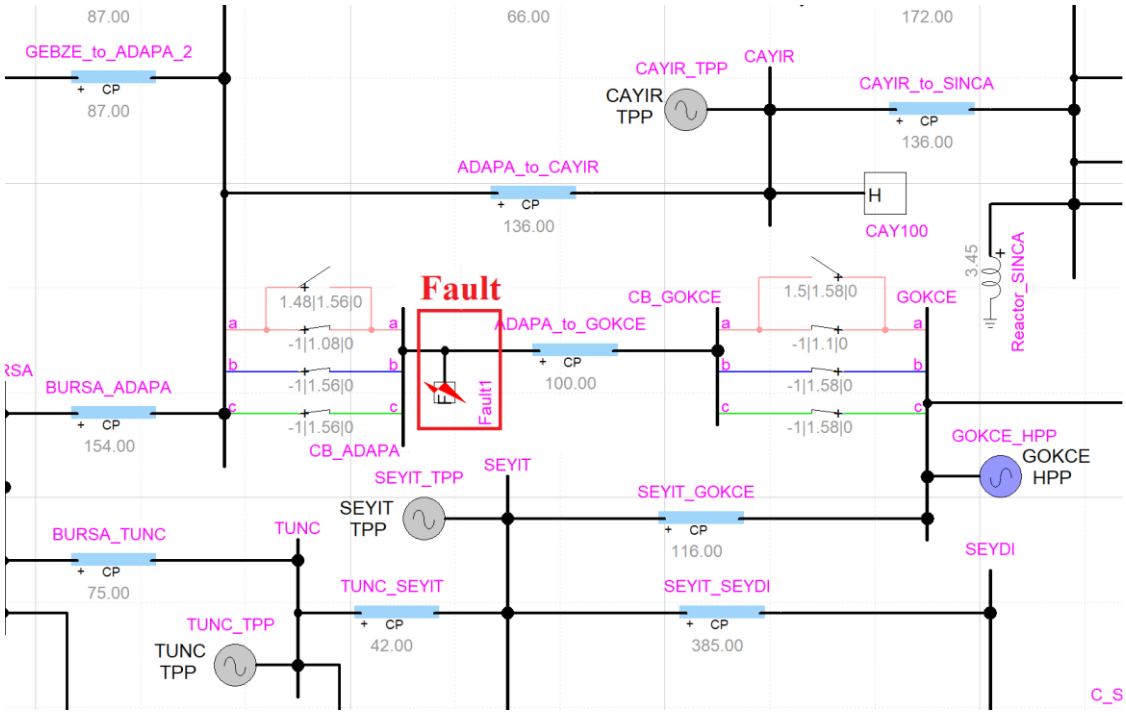

Figure 3.13 T0-DM Grid fault location

The fault occurs at 1 s, the phase-a breaker on the left of the line receives the opening signal at 1.08 s and the one on the right at 1.1 s. The phase-a breaker on the left recloses at 1.48 s and the one on the right at 1.5 s. The reclosing is unsuccessful and all breakers (all left and right phases) receive the opening signal at 1.56 s to isolate the line. Figure 3.14 shows two waveforms of phase A voltage drop across line ADAPA_to_GOKCE calculated by EMTP-MDO and SMPEMT solvers. Figure 3.15 and Figure 3.16 show real power comparison of two synchronous machines located close to the fault. Calculation of the error percentage between the EMTP-MDO and SMPEMT solvers at $t = 1.01$ second is found to be 3.8×10^{-10}, 2.67×10^{-8} and 1.37×10^{-10} for the three signals respectively.

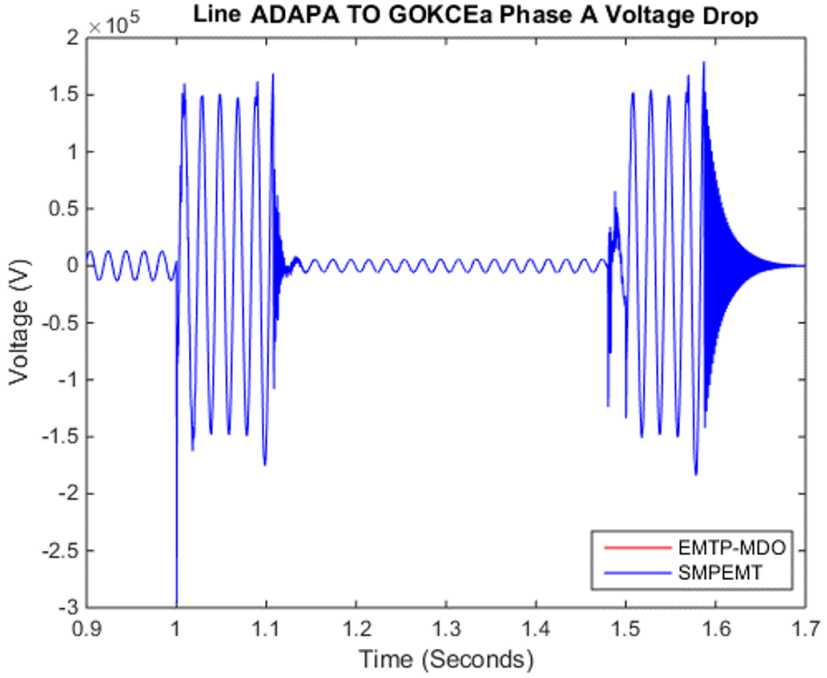

Figure 3.14 Line ADAPA TO GOKCE voltage drop - phase A

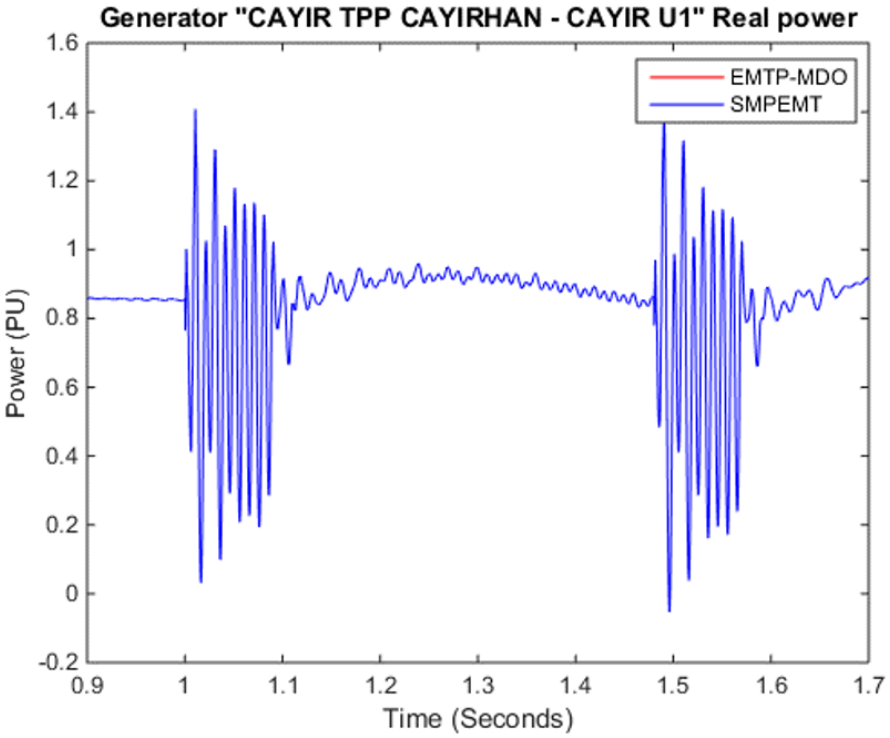

Figure 3.15 Generator CAYIR TPP CAYIRHAN U1 real power

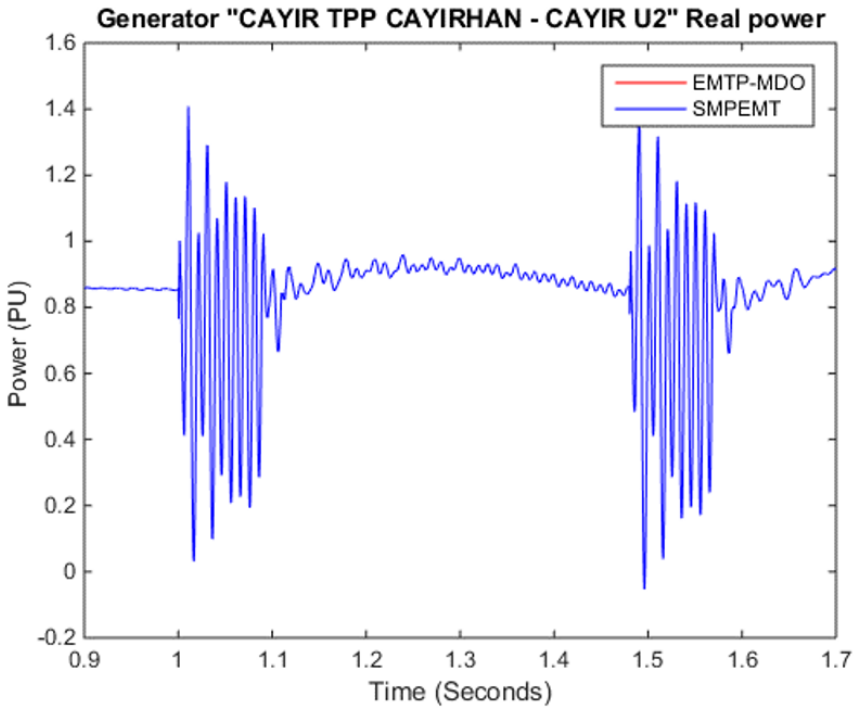

Figure 3.16 Generator CAYIR TPP CAYIRHAN U2 real power

3.1.3 T1-AVM Grid

The T1-Grid is another version of the Turkish grid that uses the average model converters. This case simulates the effect of a fault inserted between buses CAYER and ADAPA. This case shown in Figure 3.17 uses wind turbine as part of its generations and includes the following main components:

- RLC branches: 594
- PI/RL coupled branches, 3-phase: 6
- CP Lines/Cable: 58
- Ideal transformer units (for 3-phase transformers): 141
- Ideal switches: 213
- Synchronous generators (with AVRs and governors): 33
- Loads: 105

A top view of T1-Grid is shown in Figure 3.17, and the exact location of the fault can be seen in the same figure. Figure 3.18 and Figure 3.19 show matrix $\mathbf{A}$ sparsity pattern before and after BTF permutation.

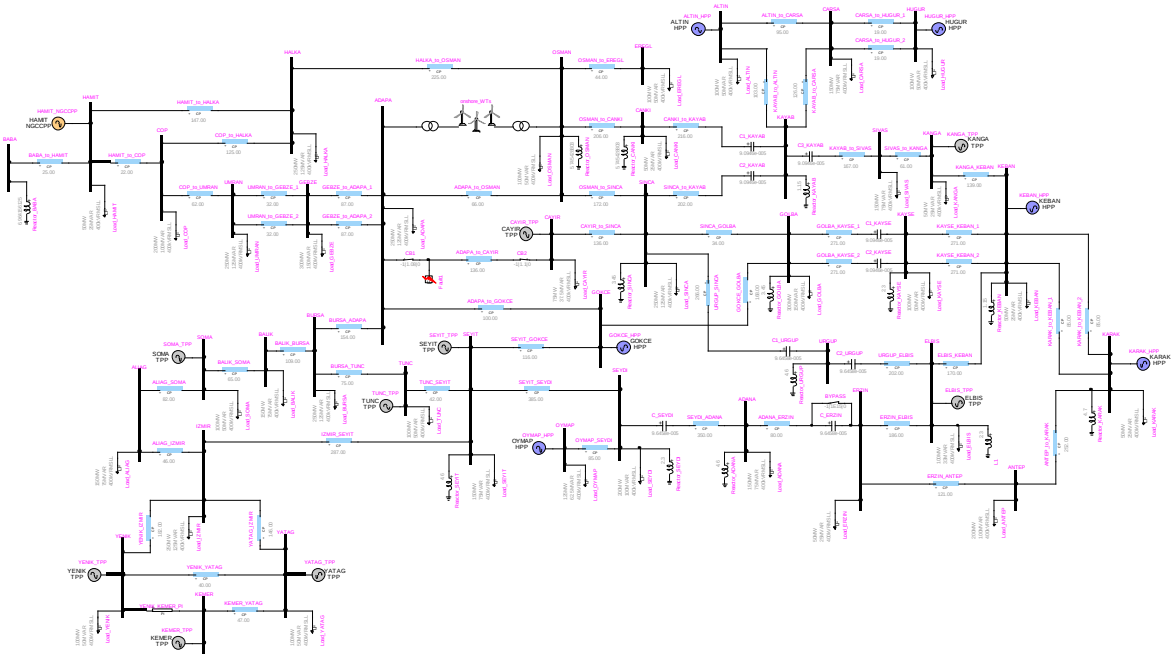

Figure 3.17 T1-AVM Grid top view

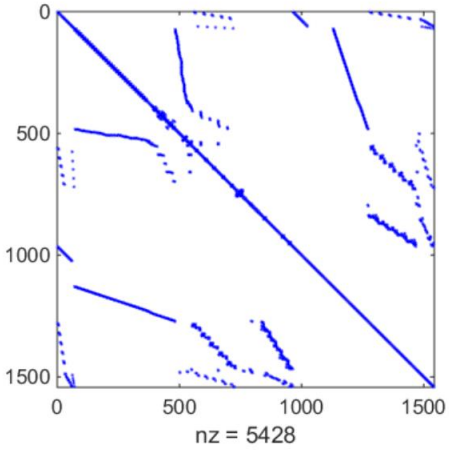

Figure 3.18 T1-AVM Grid matrix **A** before BTF permutation.

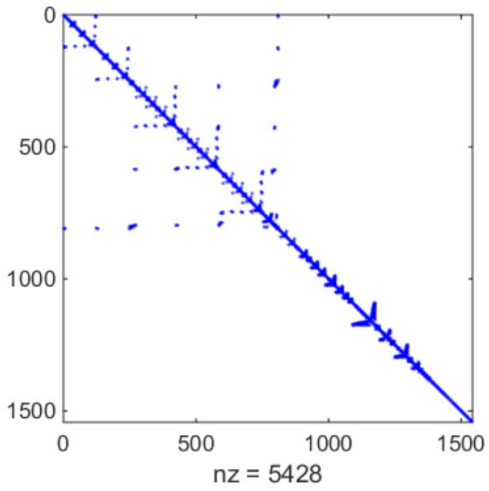

Figure 3.19 T1-AVM Grid matrix $\mathbf{A}$ after BTF permutation

T1-Grid Simulation data:

- Simulation time: 10 second
- Simulation time step: 50 μs
- Pivot tolerance ε_p : 0.01
- Average number of iterations per time step: 3.01
- Total number of iterations: 604919
- Matrix $\mathbf{A}$ size: 1542 ×1542
- Number of nonzero elements (nnz) in $\mathbf{A}$: 5428
- Sparsity percentage: 99%
- Total number of BTF of Blocks (nblocks): 45
- Biggest block size: 811×811
- Smallest block size: 3×3

The BTF of matrix $\mathbf{A}$ shows that a limiting block exists in this case. This block is the first block seen in Figure 3.19. The size of the limiting block is 811 and it limits the parallelization of the case

beyond two threads since it does not have any delay-based lines in it and can't be divided using BTF permutation. shows the solution of equation (1.4) timing using different solvers and different number of threads.

Table 3.4 T1-Grid sparse matrix solution timings for 1s simulation and $\Delta t = 50\,\mu s$

	Number of cores								
Solver	**1**	**2**	**4**	**8**	**12**	**13**	**14**	**15**	**16**
EMTP	48								
KLU	53								
SMPEMT1	19.5	10	13.1	14	15.4	17	19	20	20
SMPEMT2	17	8.5	11.2	13	14.5	16	18	18.5	19

In this case the difference between SMPEMT1 and 2 is minor due to the fact that the biggest block's FLDC is located at the 6th column in the BTF format and that limits the ability of partial KLU-RF to decrease the computation time of the block factorization. The best gain is achieved with SMPEMT2 (48/10 = 4.8). The overall computation time including the solution of equation (2.1), the control solution, steady-state solution and updating matrix $\mathbf{A}$ and vector $\mathbf{b}$ drops from 68 seconds (when using EMTP solver) to 21 seconds (when using SMPEMT2, parallel control solver and 8 threads topology).

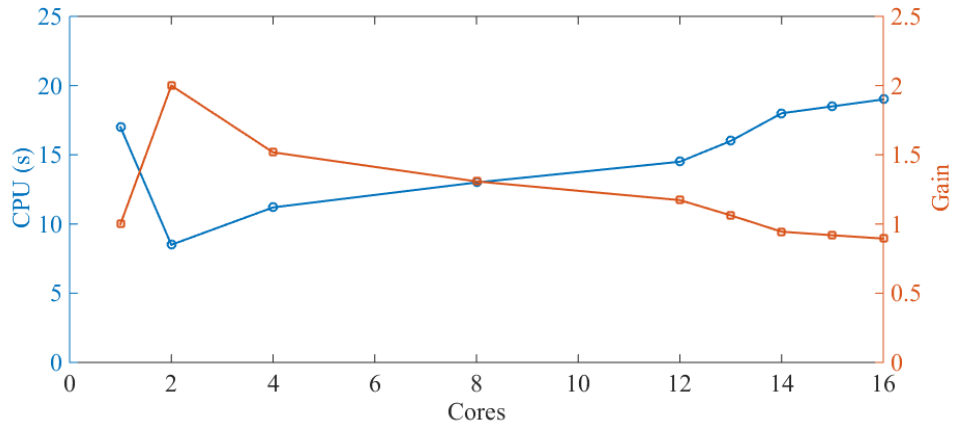

Figure 3.20 SMPEMT T1-Grid simulation time and gain for AVM model

The studied event in this test case is a (3-phase-to-ground) fault on the transmission line ADAPA_to_ CAYIR connected between the lines ADAPA and CAYIR as seen in.

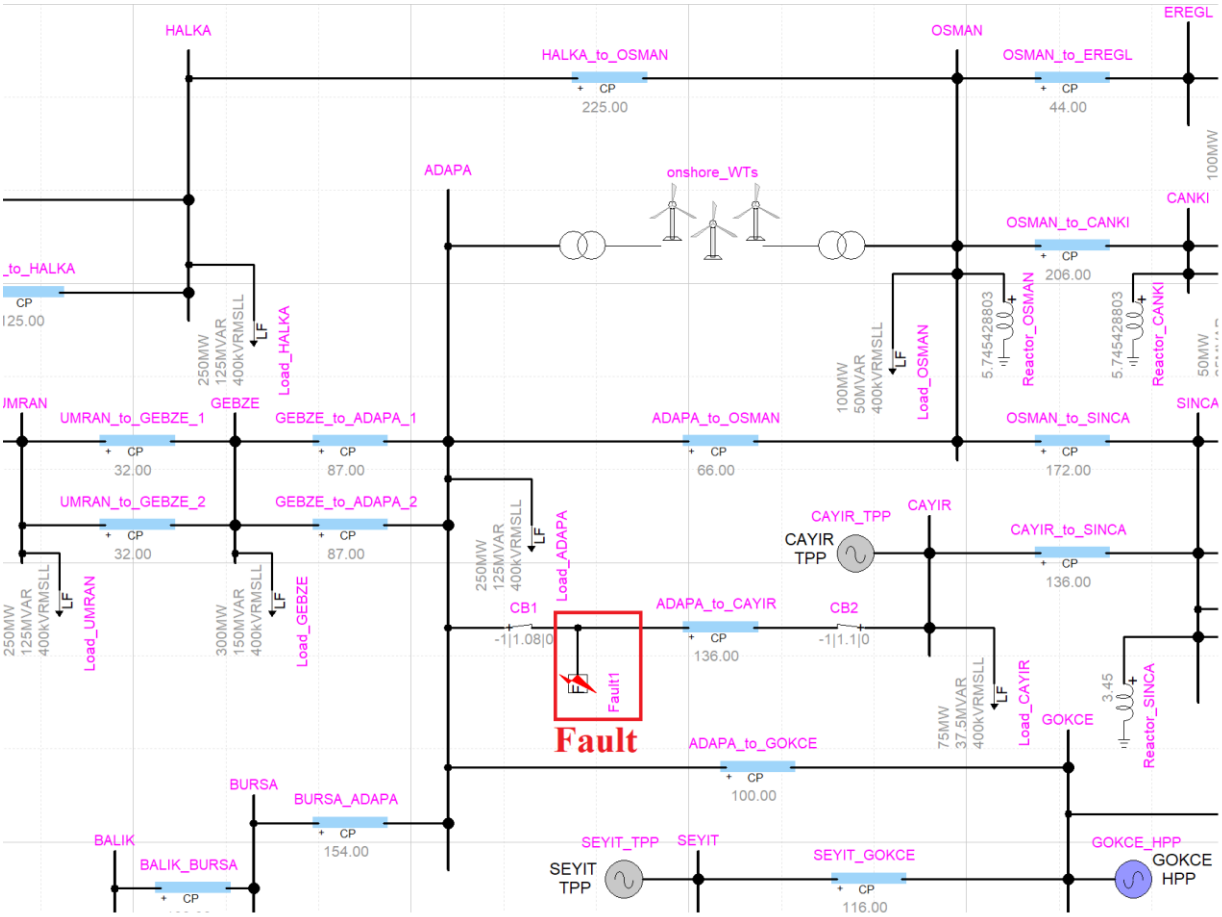

Figure 3.21 T1-AVM fault location

The fault occurs at 1 s, the breaker on the left of the line receives the opening signal at 1.08 s and the one on the right at 1.1 s. Figure 3.22 and Figure 3.23 show the voltage drop across line ADAPA_TO_CAYIR and the real power of SM CAYIR TPP CAYIRHAN U2 respectively. Both figures contain two waveforms calculated by EMTP-MDO and SMPEMT and both curves overlap with difference seen throughout the faults span period.

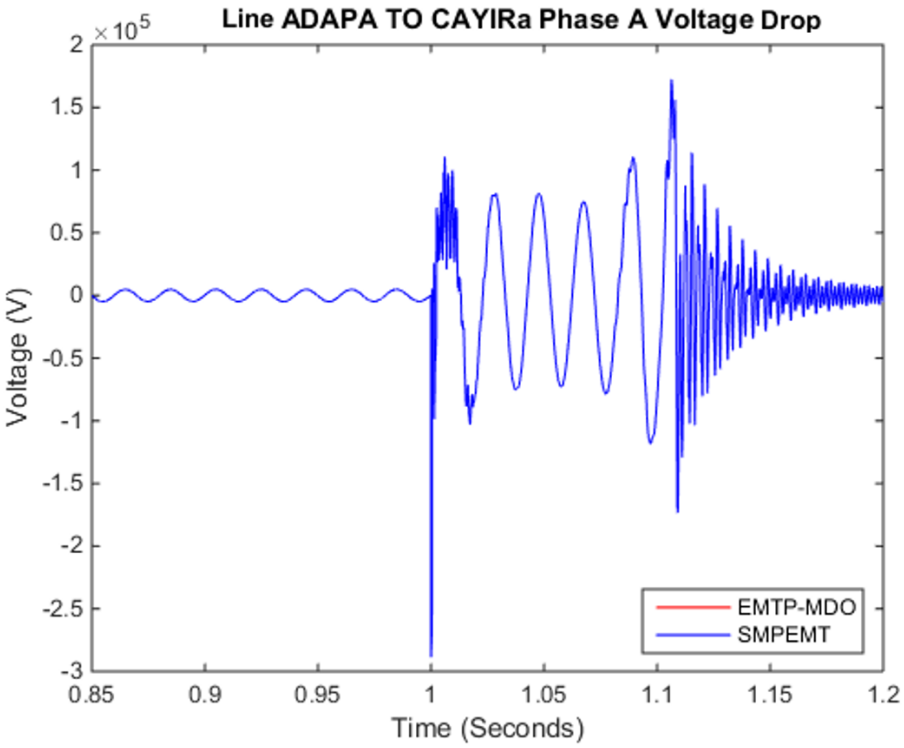

Figure 3.22 T1-Grid line ADAPA TO CAYIR voltage drop - phase A

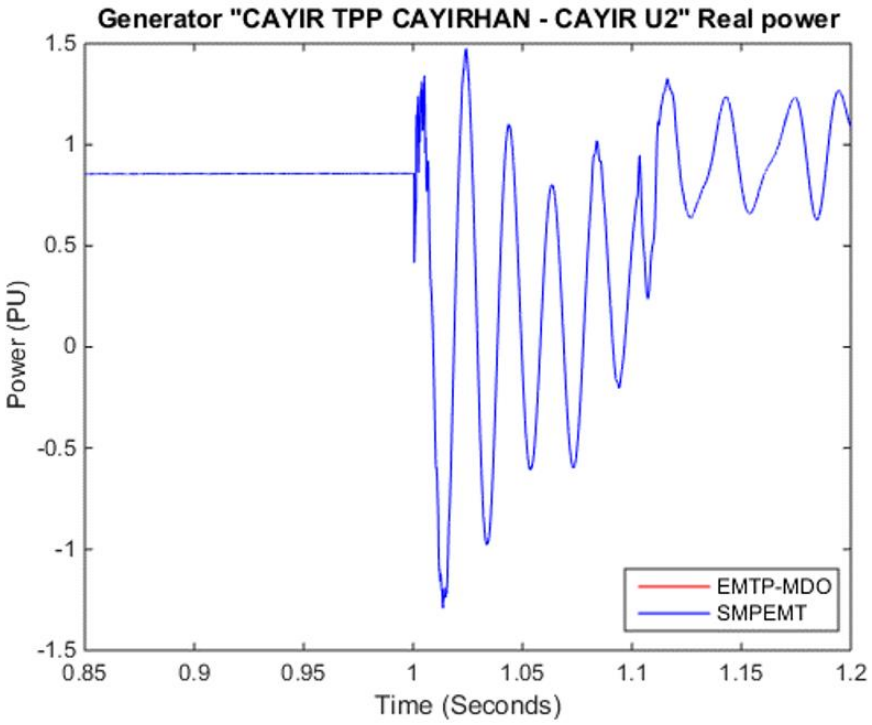

Figure 3.23 Generator CAYIR TPP CAYIRHAN U2 real power

3.1.4 T2-AVM Grid

T2-Grid is a modified version of the Turkish grid discussed in the previous section. Three offshore wind turbine farms were added to the HVDC_ALIGA and HVDC_IZMIR buses. The location of the fault is kept between buses CAYER and ADAPA. The main components of the case are the following:

- RLC branches: 900
- PI/RL coupled branches, 3-phase: 9
- CP Lines/Cable: 62
- Ideal transformer units (for 3-phase transformers): 168
- Ideal switches: 410
- Synchronous generators: 28
- Loads: 105

T2-Grid Simulation data:

- Simulation time: 10 second
- Simulation time step: 50 μs
- Pivot tolerance ε_p : 0.01
- Average number of iterations per time step: 3.04
- Total number of iterations: 610783
- Matrix $\mathbf{A}$ size: 2425ˣ2425
- Number of nonzero elements (nnz) in $\mathbf{A}$: 8347
- Sparsity percentage: 99%
- Total number of BTF of Blocks (nblocks): 58
- Biggest block size: 811×811

- Smallest block size: 3×3

Figure 3.24 shows the top view of the case that provides an illustration of the faults position and all offshore wind farms locations. Figure 3.25 and Figure 3.26 shows the matrix **A** nonzero pattern before and after BTF permutation. Unlike T1-Grid discussed in section 3.1.3, the biggest block (size = 811) consist of almost 30% of the case size and that will loosen the parallelization limitation seen in benchmark T1. However, the biggest block will still impose limitation on parallelization beyond 4 threads.

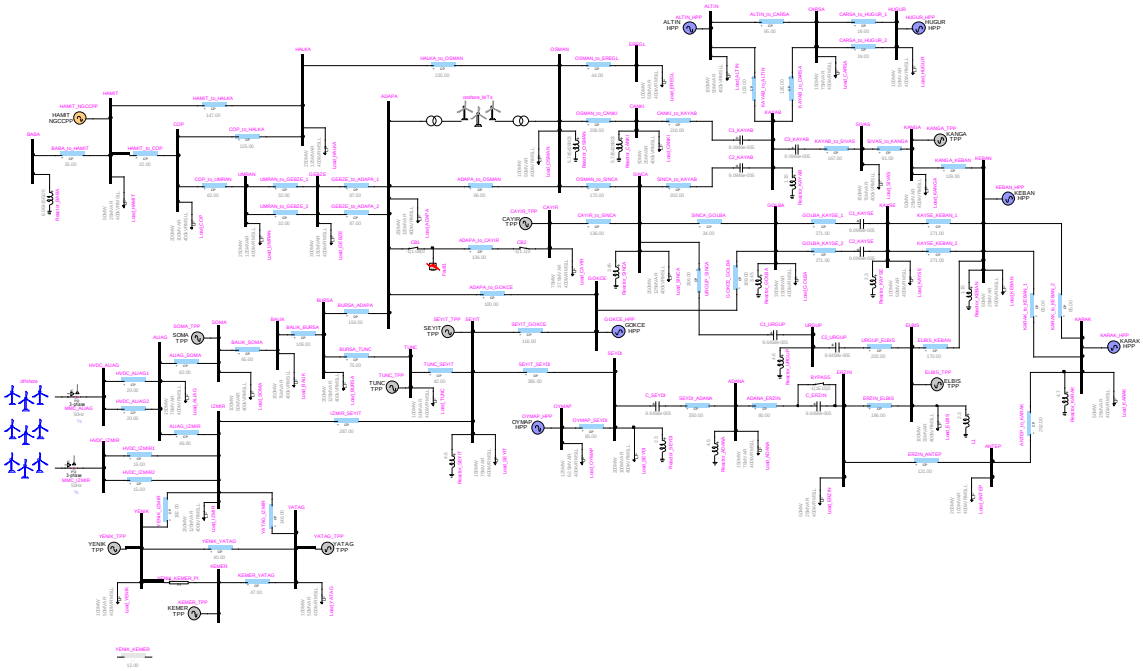

Figure 3.24 T2-AVM Grid top view

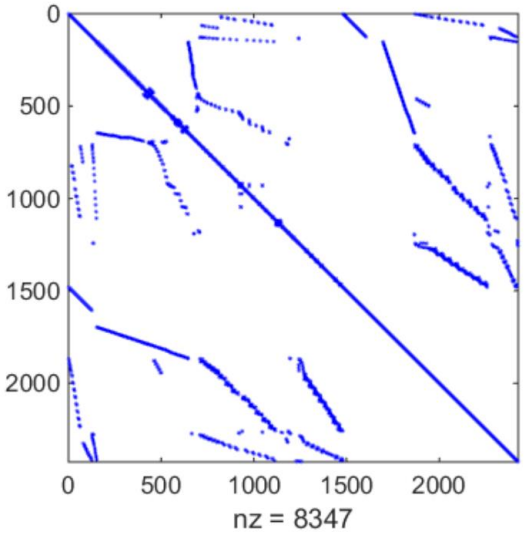

Figure 3.25 T2-AVM Grid matrix **A** before BTF permutation

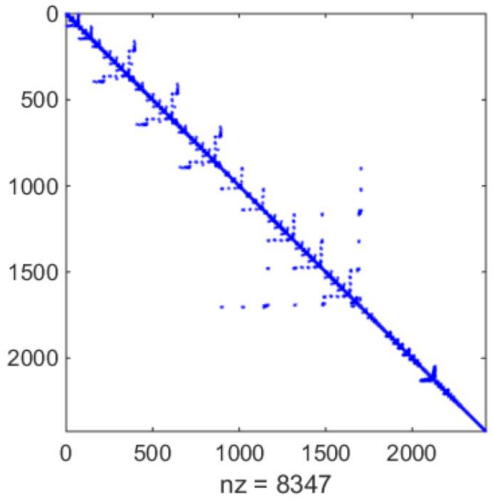

Figure 3.26 T2-AVM Grid matrix **A** after BTF permutation

shows the solution of equation (1.4) timing using different solvers and different number of threads. It can be notices from the results that SMPEMT1/2 performance accelerates with the increase number of threads up to 3 threads and after that the performance starts to deteriorate. Like other cases, this phenomenon is due to the limiting block (biggest block = 811) that does not have any CP line in. The overall computation time including the solution of equation (2.1), the control solution, steady-state solution and updating matrix **A** and vector **b** drops from 164 seconds (when

using EMTP solver) to 29 seconds (when using SMPEMT2, parallel control solver and 8 threads topology).

Table 3.5 T2-Grid sparse matrix solution timings for 1s simulation and $\Delta t = 50\,\mu s$

	Number of cores								
Solver	**1**	**2**	**4**	**8**	**12**	**13**	**14**	**15**	**16**
EMTP	64								
KLU	72								
SMPEMT1	33	15	7.8	11	11.9	12.5	13.2	14	14
SMPEMT2	31	14	7.5	10	11.2	12	13	14	14

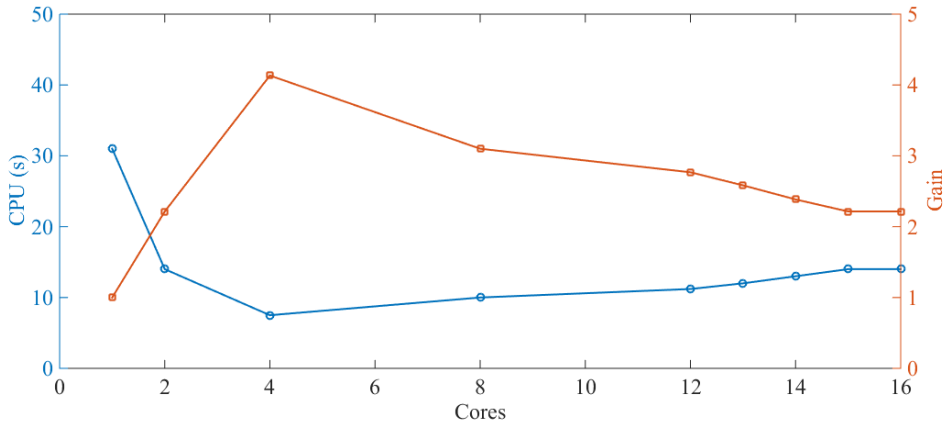

Figure 3.27 SMPEMT T2-Grid simulation time and gain for AVM model

The studied event in this test case is similar to the one seen in the previous section and the fault scenario remains the same. However; three set of offshore wind turbines are added to the case and that will add more numerical stress on the solvers. Figure 3.28 and Figure 3.29 show the comparison between EMTP-MDO and SMPEMT using phase A voltage drop across line ADAPA_TO_CAYIR and SM CAYIR TPP CAYIRAN U2 real power. Both figures show complete overlap between the two solvers results and no difference can be seen visually.

Calculation of the error percentage between the EMTP-MDO and SMPEMT solvers at $t = 1.01$ second is found to be 5.3×10^{-9} and 8.2×10^{-10} for both signals respectively.

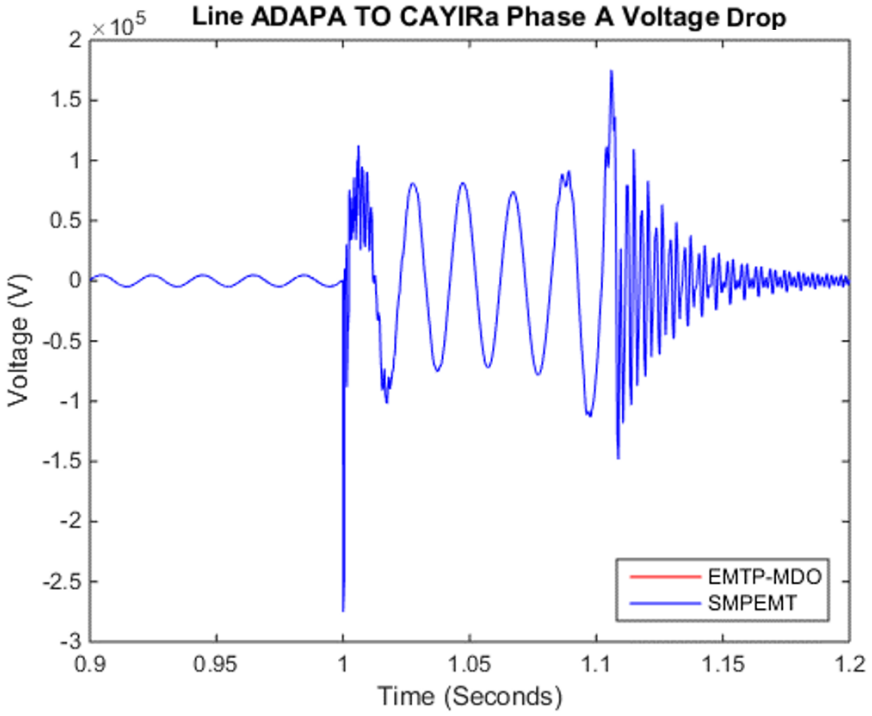

Figure 3.28 Line ADAPA_TO_CAYIR voltage drop - Phase A

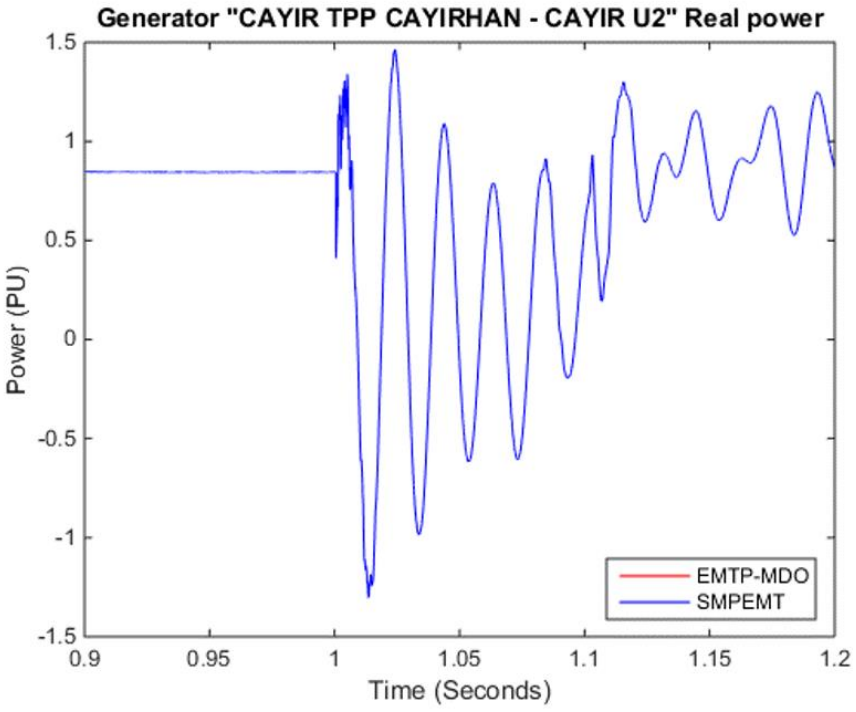

Figure 3.29 SM CAYIR TPP CAYIRAN U2 real power

From the above figure it can be seen that both solvers produce similar result and both curves perfectly overlap each other.

3.1.5 IEEE14

This benchmark represents a simplified version of the IEEE 14 bus system [60]. This case has 14 buses, 5 generators and 11 loads. The case does not have any nonlinear instances and hence the number of iterations is 0. The case was simulated for 1 second with a 50 μs time step.

The BTF version of matrix $\mathbf{A}$ has only one block since the case has no CP lines and can't be decoupled in time domain. The size of the only block in BTF form is the size of the case overall $\mathbf{A}$ matrix. Figure 3.30 and Figure 3.31 show the sparsity pattern of matrix $\mathbf{A}$ before and after BTF permutation. shows the solution of equation (1.4) timing using different solvers with one thread only.

IEEE14-Grid Simulation data:

- Simulation time: 1 second
- Simulation time step: 50 μs
- Pivot tolerance ε_p : 0.01
- Average number of iterations per time step: 0 (linear case)
- Total number of iterations: 20000
- Matrix $\mathbf{A}$ size: 99×99
- Number of nonzero elements (nnz) in $\mathbf{A}$: 711
- Sparsity percentage: 92.7%
- Total number of BTF of Blocks (nblocks): 1

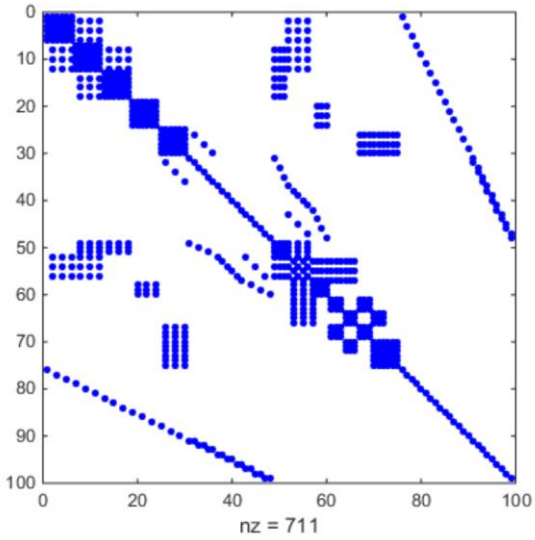

Figure 3.30 IEEE14-Grid matrix **A** before BTF permutation

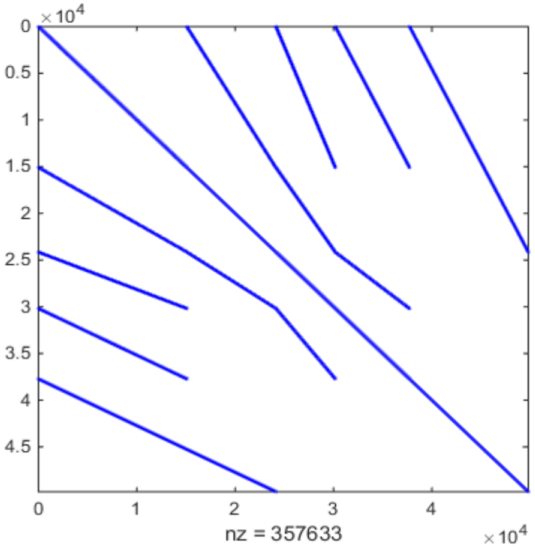

Figure 3.31 IEEE14-Grid matrix **A** after BTF permutation

Table 3.6 IEEE14 sparse matrix solution timings for T=1s and Δt =50 μs

	Number of cores				
Solver	**1**	**2**	**4**	**8**	**12**
EMTP	0.91				
KLU	1.10	N/A	N/A	N/A	N/A
SMPEMT1	0.77	N/A	N/A	N/A	N/A
SMPEMT2	0.77	N/A	N/A	N/A	N/A

Since the BTF format has only one block, all tests conducted for this case were done using one thread only. No major gain is seen when SMPEMT1 and 2 since factorization of the **A** matrix is done once due to the lack of nonlinear elements in the case and the pivot validity testing feature didn't have any impact of the gain seen in SMPEMT1/2 timings, but rather it is all due to the partial forward substitution that was explained in 2.4.

Figure 3.32 shows comparison between two signals calculated by EMTP-MDO and SMPEMT solvers. The two signals represent phase A voltage drop across transmission line (PI15). From the figure, it can be seen that both signals are completely matching and no difference can be notices throughout the waveforms in the figure.

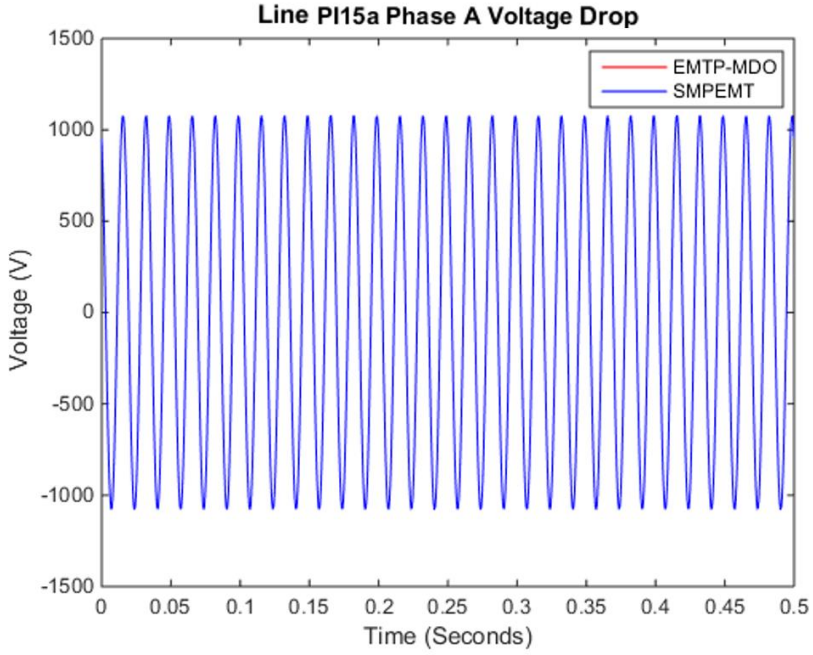

Figure 3.32 Line PI15 voltage drop - phase A

3.1.6 IEEE7000

The IEEE7000 benchmark is built by repeating the IEEE14 case 500 times in order to get a case with 7000 buses. The different IEEE14 cases were linked by CP lines at buses 13 and 14. Using CP lines between different IEEE14 allows to have 500 blocks in the BTF format where each block represents an IEEE14 case. Figure 3.33 and Figure 3.34 show the network **A** before and after BTF permutation.

The IEEE7000 case was simulated for 1 second with time step $\Delta t = 50\mu s$ and similar to IEEE14 benchmark, this case does not have any iterations due to the absence of any nonlinear objects and hence the rate of iterations per time step is 0. shows the solution of equation (1.4) timing using different solvers and different number of threads.

IEEE7000-Grid Simulation data:

- Simulation time: 1 second
- Simulation time step: 50 μs
- Pivot tolerance ε_p : 0.01
- Average number of iterations per time step: 0
- Total number of iterations: 20000
- Matrix $\mathbf{A}$ size: 49698×49698
- Number of nonzero elements (nnz) in $\mathbf{A}$: 357633
- Sparsity percentage: 99.9%
- Total number of BTF of Blocks (nblocks): 500

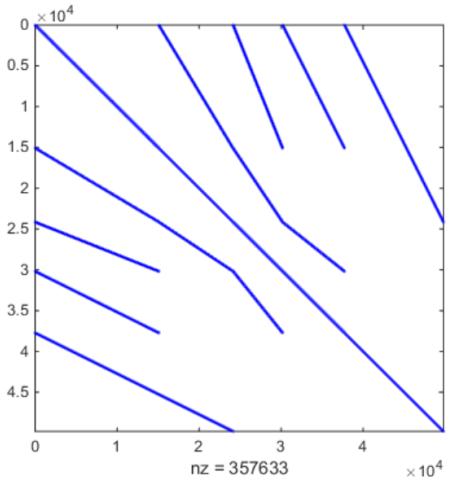

Figure 3.33 IEEE7000-Grid matrix $\mathbf{A}$ before BTF permutation

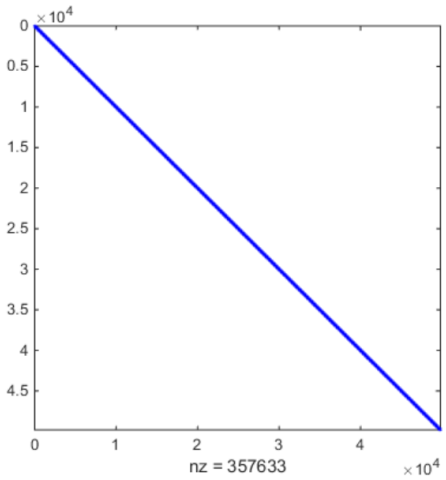

Figure 3.34 IEEE7000-Grid matrix **A** after BTF permutation

Table 3.7 IEEE7000 sparse matrix solution timings for 1s simulation and Δt =50 µs

	Number of cores								
Solver	**1**	**2**	**4**	**8**	**12**	**13**	**14**	**15**	**16**
EMTP	1074								
KLU	1135								
SMPEMT1	222	125	75	42	25	21	18.4	16	15
SMPEMT2	222	125	75	42	25	21	18.4	16	15

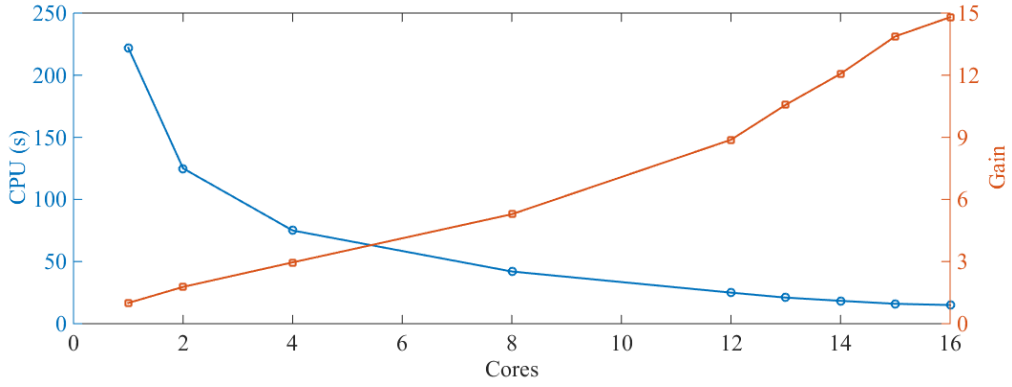

Figure 3.35 SMPEMT IEEE7000-Grid simulation time and gain

The same case was simulated in [19] where the case was partitioned using a boarder block diagonal scheme based on the use of PI section transmission lines. An approximation of this test results is shown in .

Table 3.8 T2-Grid sparse matrix solution timings with BBD (s)

Number of cores	**1**	**10**	**20**	**30**	**40**	**50**	**60**
Simulation time	5000	550	250	200	200	210	220

From the above two tables, it can be seen that the KLU based approach implemented herein is more efficient and faster than what is proposed in [19], and the timing obtained with 20 threads in was achieved and overcome with only one thread of SMPEMT2 as seen in . Although the machine used to obtain the results in and [19] have different processors type, the other specifications are very close and this difference can't justify the different in results.

3.1.7 IEEE39

The IEEE39 benchmark represents a part of New England 345-KV grid. It consists of 10 synchronous generators, 39 buses, 12 transformers, and 19 loads. The case has a total of 34 transmission lines with 24 modeled as CP lines and the rest as PI section type of lines. A simplified version of the case was modeled using EMTP with the following list presents a summary of the case main components:

- Synchronous machine: 10
- Ideal Transformer units: 90
- RLC: 337
- Ideal switch: 123
- L nonlinear: 87
- PQ load centers: 57
- AC current source: 57
- PI/RL lines: 10
- CP lines/cable: 24

IEEE39-Grid Simulation data:

- Simulation time: 10 second
- Simulation time step: 20 μs

- Pivot tolerance ε_p : 0.01
- Average number of iterations per time step: 1.18
- Total number of iterations: 711930
- Matrix $\mathbf{A}$ size: 486ˣ486
- Number of nonzero elements (nnz) in $\mathbf{A}$: 1662
- Sparsity percentage: 99.2%
- Total number of BTF of Blocks (nblocks): 57
- Biggest block size: 60×60
- Smallest block size: 3×3

Figure 3.36 shows the case top view that provides general understanding about the case layout and elements distributions. Figure 3.37 and Figure 3.38 show the case $\mathbf{A}$ matrix before and after BTF permutation. This case is a relatively small case and applying parallel computation on it shows to what extend parallelization can accelerate the performance of the solution before the overhead weight of synchronizing thread, launching and joining threads takes over. shows the solution of equation (1.4) timing using different solvers and different number of threads.

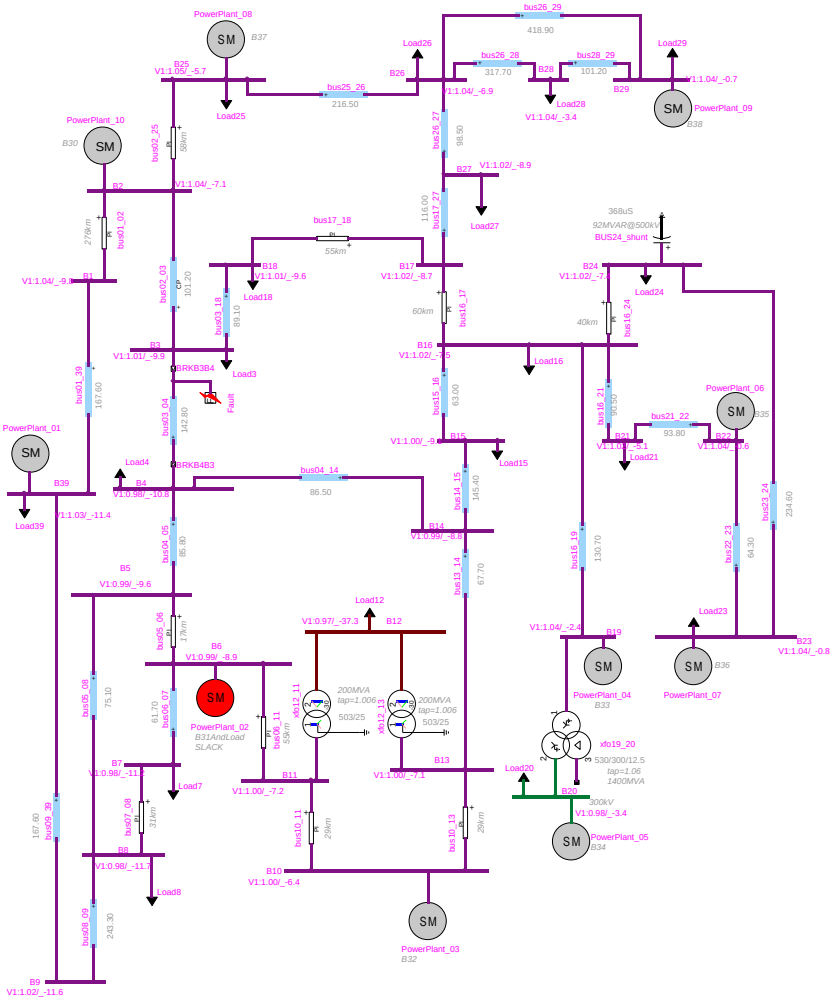

Figure 3.36 IEEE39-Grid top view

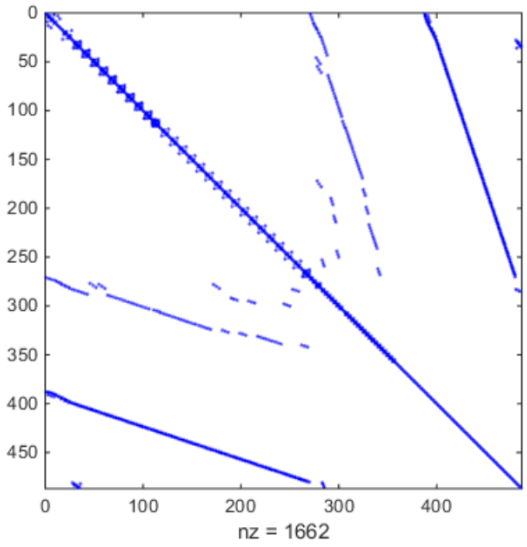

Figure 3.37 IEEE39-Grid matrix **A** before BTF permutation

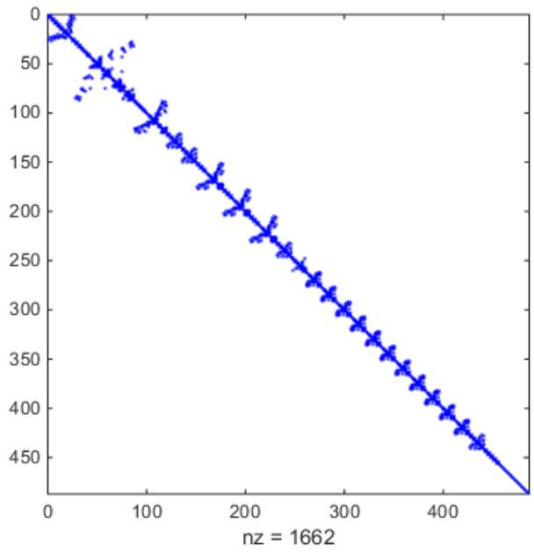

Figure 3.38 IEEE39-Grid matrix **A** after BTF permutation

Table 3.9 IEEE39- Grid sparse matrix solution timings for T=1s and Δt =50 μs

	Number of cores								
Solver	**1**	**2**	**4**	**8**	**12**	**13**	**14**	**15**	**16**
EMTP	38								
KLU	43								
SMPEMT1	11.7	6.5	5	7.8	9.2	10.8	12	13	13.5
SMPEMT2	10.2	5.5	4	6.9	9	10.5	11.8	13	13.5

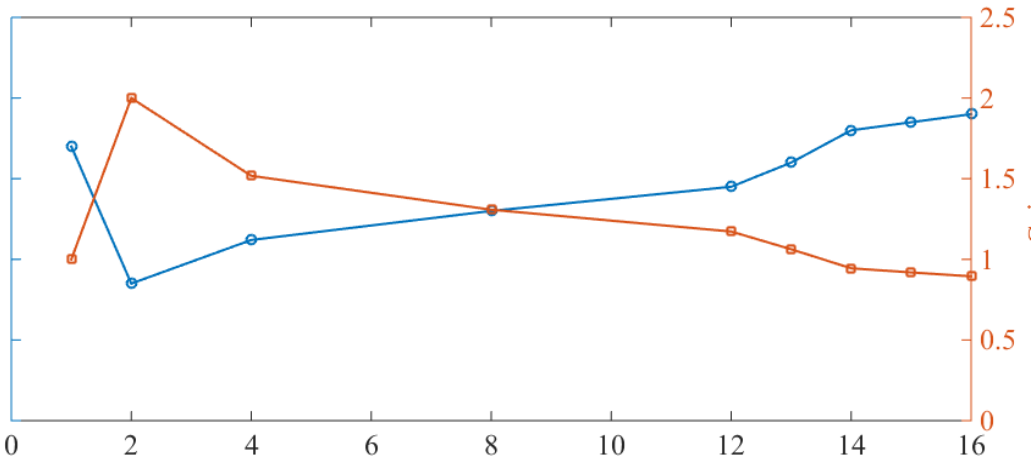

Figure 3.39 SMPEMT IEEE39-Grid simulation time and gain

The overall computation time including the solution of equation (2.1), the control solution, steady-state solution and updating matrix $\mathbf{A}$ and vector $\mathbf{b}$ drops from 52 seconds (when using EMTP solver) to 38 seconds (when using SMPEMT2, parallel control solver and 8 threads topology).

The studied event in this test case is a (3-phase-to-ground) fault inserted between on the transmission line bus03_04 as seen in Figure 3.40. The fault is triggered at $t = 0.2$ second and removed at $t = 0.3$ second.

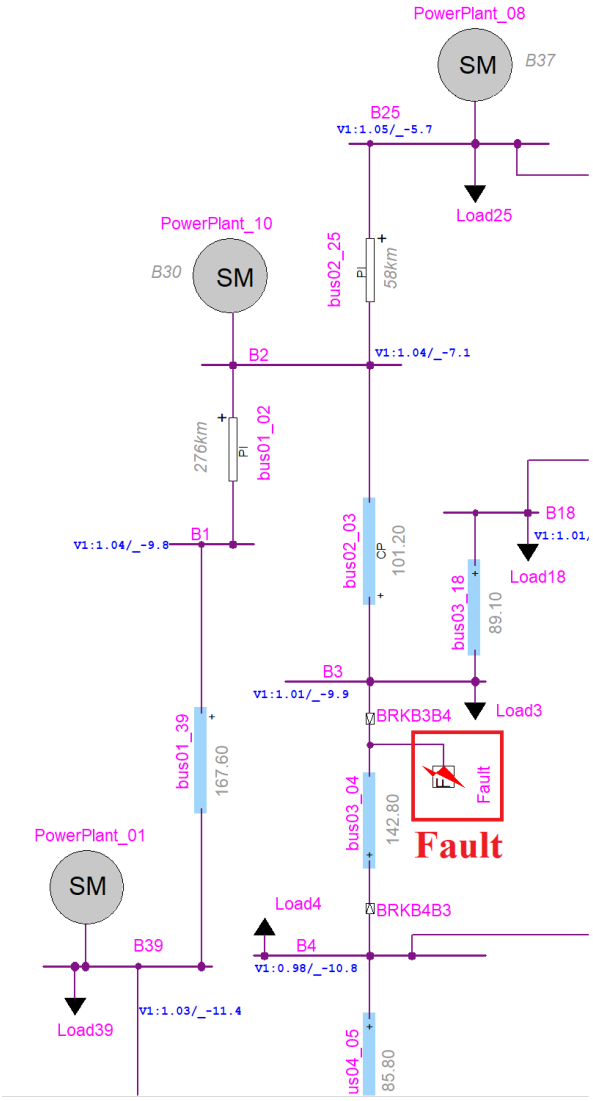

Figure 3.40 IEEE39 fault location

Figure 3.41 and Figure 3.42 below show a comparison of EMTP-MDO results and SMPEMT result. Both figures show very similar results for both solvers and no difference can be seen during the fault effect. The difference error percentage between the two solvers is found to be 7.1×10^{-9} and 1.5×10^{-11} for both signals respectively.

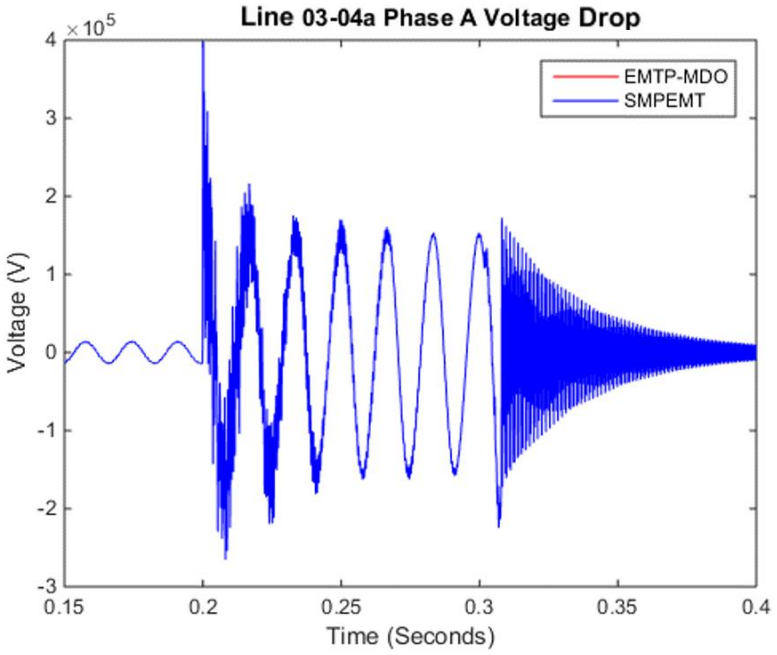

Figure 3.41 Line 03-04 voltage drop - phase A

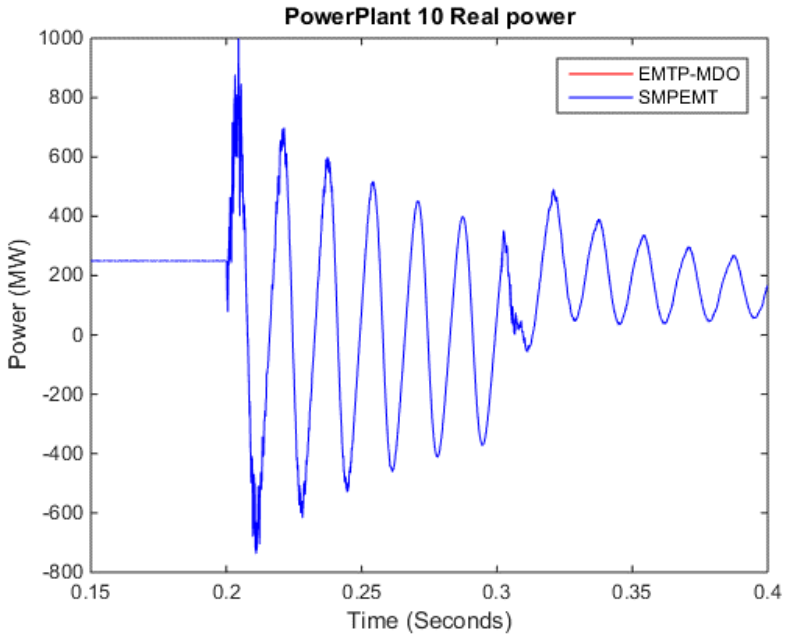

Figure 3.42 Power Plant 10 real power

3.1.8 IEEE118-GMD

This benchmark represents a modified version of the IEEE-118 that represents a portion of the American Electrical Power (AEP) system in the US Midwest [59]. This version included herein has comes with different upgrades and modifications to the original case. This modification includes

- Modifying transmission line and machine data according to the latest IEEE standards and publications and typical data from North America transmission grid.
- Adding extra features and data to allow EMT-type studies, these added features/data include transmission line data such as tower configuration, conductor data, per unit length positive sequence, zero-sequence line impedance data and line length data. In addition, different types of transmission lines have been incorporated with the case such as PI, CP and FD that will allow the user to use combination of transmission lines depending on the type of study and requirement.
- Updating machine data and adding machine controls such as excitors, governors, OEL and PSS.

The following list presents a summary of the IEEE-118 main components:

- 177 transmission lines (CP, PI and FD)
- 91 loads
- 9 Transformers
- 54 synchronous machines (SMs)
- 19 Synchronous generators (SGs)
- 35 Synchronous condensers (SCs)

IEEE118-Grid Simulation data:

- Simulation time: 400 second
- Simulation time step: 50 μs

- Pivot tolerance ε_p : 0.01
- Matrix $\mathbf{A}$ size: 8514×8514
- Number of nonzero elements (nnz) in $\mathbf{A}$: 27471
- Sparsity percentage: 99.96%
- Total number of BTF of Blocks (nblocks): 40
- Biggest block size: 1148×1148
- Smallest block size: 30×30

The case has also several voltage levels that vary between 345KV transmission, 138KV sub-transmission, 25V distribution and 20, 15, 10.5 KV generation. Figure 3.43 shows an overview of the IEEE-118 grid and the location of different components within the Network.

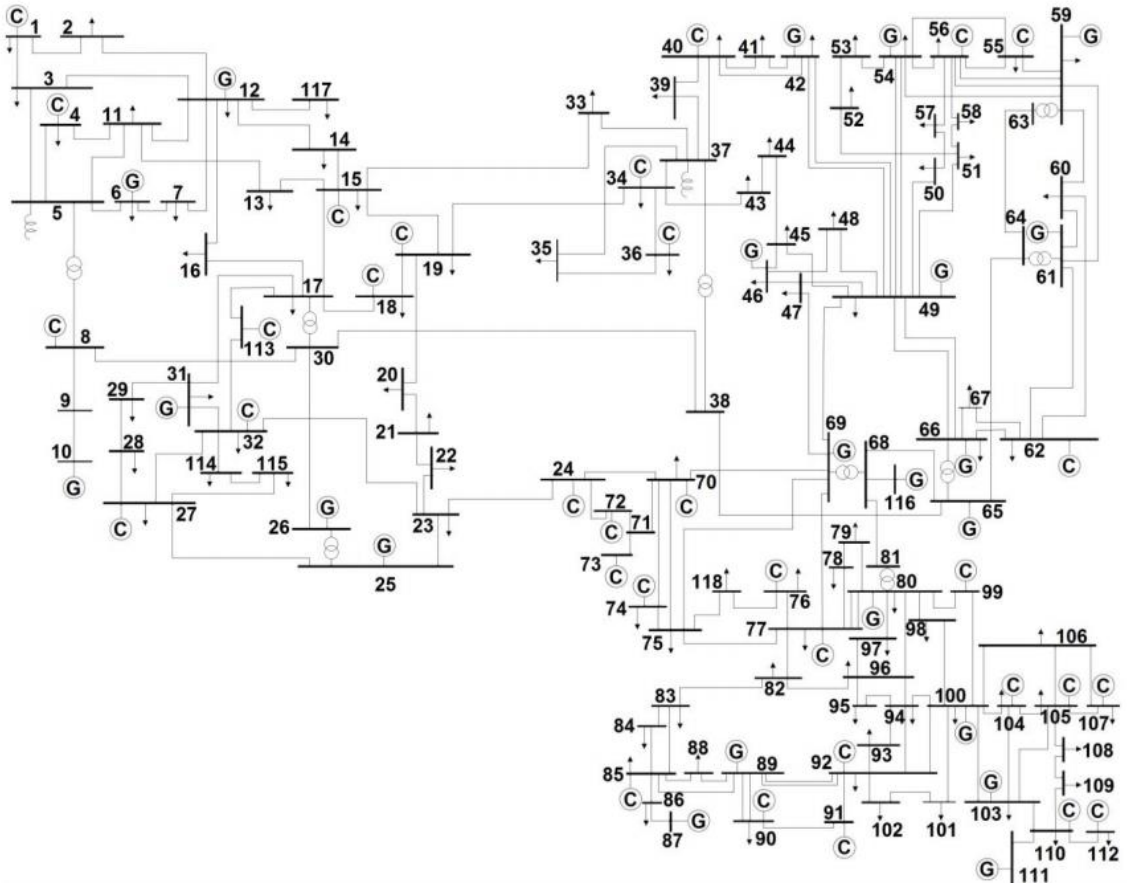

Figure 3.43 Single line diagram of IEEE-118 Grid

Figure 3.44 and Figure 3.45 show the case $\mathbf{A}$ matrix before and after BTF permutation. The case was simulated for 400 seconds with a $50\,\mu s$ time-step. The long simulation interval was selected due to the existence of different events along the first 400 seconds of simulation. shows the solution of equation (1.4) timing using different solvers and different number of threads.

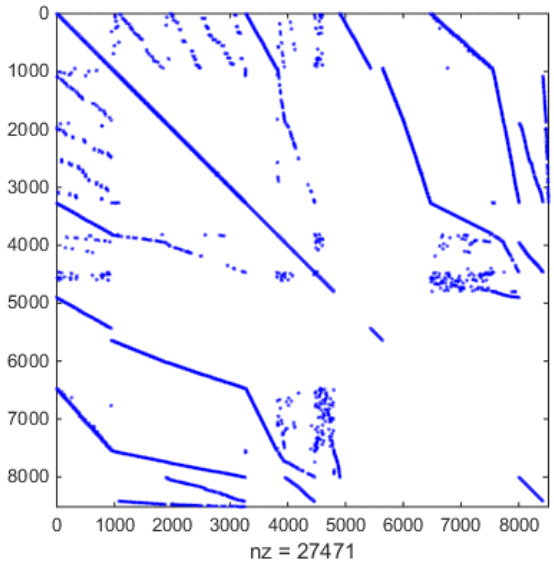

Figure 3.44 IEEE118-Grid matrix $\mathbf{A}$ before BTF permutation

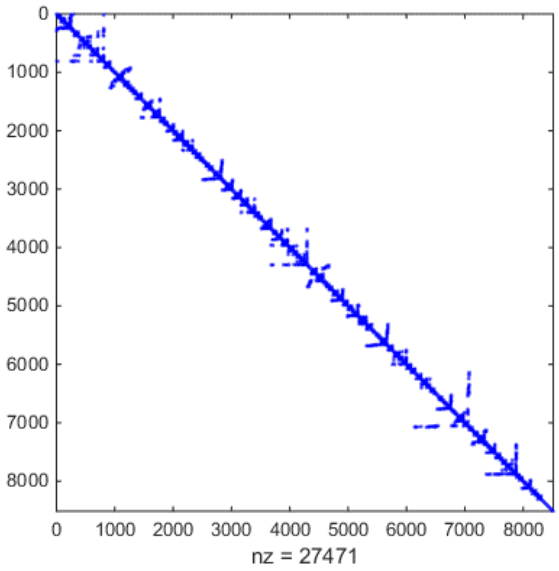

Figure 3.45 IEEE118-Grid matrix $\mathbf{A}$ after BTF permutation

Table 3.10 IEEE118- Grid sparse matrix solution timings for T=400s and $\Delta t = 50\,\mu s$

	Number of cores								
Solver	**1**	**2**	**4**	**8**	**12**	**13**	**14**	**15**	**16**
EMTP	39730								
KLU	43687								
SMPEMT1	16870	8698	4698	2267	2865	2883	2892	2892	2898
SMPEMT2	16794	8624	4653	2241	2843	2868	2867	2871	2873

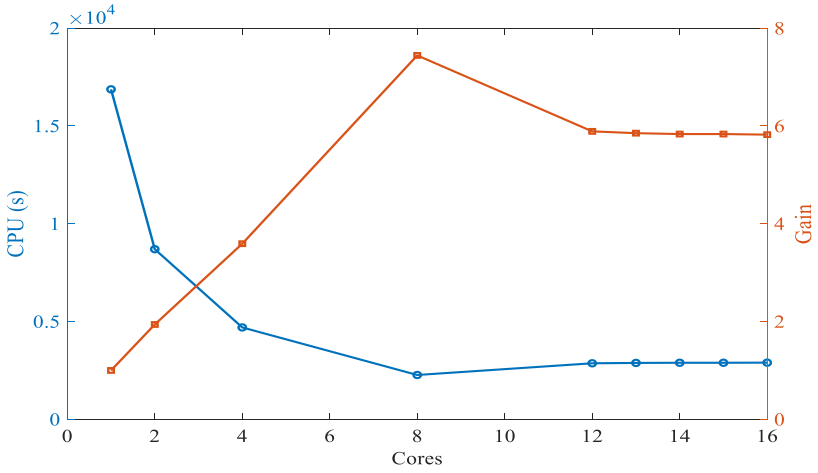

Figure 3.46 SMPEMT IEEE118-Grid simulation time and gain

The overall computation time including the solution of equation (2.1), the control solution, steady-state solution and updating matrix $\mathbf{A}$ and vector $\mathbf{b}$ drops from 168130 seconds (when using EMTP solver) to 67473 seconds (when using SMPEMT2, parallel control solver and 16 threads topology). The above simulation timing shows a nearly linear gain from thread 1 to thread 8 and the performance starts to deteriorate after the 8^{th} thread. This phenomenon is due to the existence of a limiting block of size 1148 that limits the gain to 8 threads and prevents any further acceleration.

3.2 Results analysis

The results presented in the previous chapter illustrate the gain that can be achieved by using SMPEMT solver. Depending on the case configuration, the gain may vary widely depending on different types of factors. These factors involve the following:

- The existence of CP lines in the case: The use of parallel computation in SMPEMT depends mainly of the ability to divide the network matrix into various independent blocks. The division process is based on the time domain decoupling effect of the constant parameter transmission lines. If no CP lines exist in the case, the parallel computation algorithm can't be used and the whole network matrix is solved on one thread. Although some of the features of SMPEMT may help accelerating the performance, the overall gain will not be that great compared to the gain obtained by parallelization the solution.
- The testing platform (hardware) used in the simulation: Although SMPEMT works on all machines with more than one processor (CPU), it is notices that the ultimate performance can be achieved with higher number of physical cores (avoiding hyper threading) and bigger cache line of the machine. These two factors allow threading to be more efficient by avoiding sequencing of parallel tasks and allowing different threads to handle bigger blocks and matrices.
- Network configuration: The satisfying of the first two factors does not guarantee good performance and a scaling gain without having a network configuration that is well designed with parallel solution in mind. In order to have an efficient parallelization with lasting effect at higher number of threads, the blocks of BTF matrix must be as small as possible to enable SMPEMT load balancing topology to distribute blocks evenly on different CPUs. Having a limiting block (bottle nick) will limit the gain and make the use of higher number of processors a burden. Figure 3.47 shows an example of a network that has a block that is almost one third of the overall size of the matrix. Such block limits the gain of SMPEMT to three threads only. Whereas, Figure 3.48 shows an example of a case that has a perfect distribution of elements across its blocks and the threading performance of this type of cases will be efficient and a high gain can be achieved with higher number of threads.

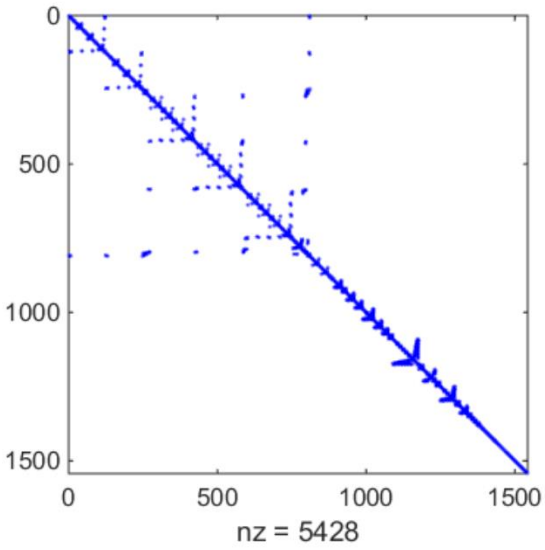

Figure 3.47 A network with a limiting block

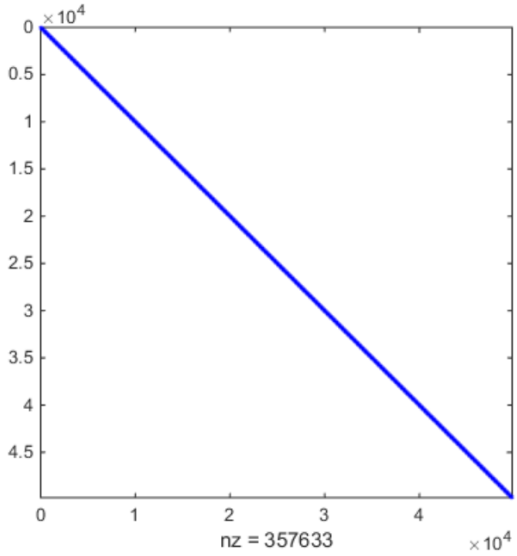

Figure 3.48 A network with a perfect distribution of blocks

The following table lists all cases used in SMPEMT validation with the maximum gain achieved for each case, the size of **A** matrix and the size of limiting block for each case.

Table 3.11 Testing cases performance summary

Case name	Matrix size	Limiting Block	Maximum gain
HQ case (Full)	41797	2898	33.8
T0-DM	4703	573	12.5
T1_AVM	1542	811	5.65
T2_AVM	2425	811	8.5
IEEE-14	99	99	1
IEEE-7000	49500	99	71.6
IEEE-39	486	60	9.5
IEEE-118 GMD	8514	1148	17.7

From the above table, it can be seen that most cases have hit a point where the gain is maximized, and no further gain can be achieved. Despite the fact that increasing the number of threads adds an overhead to the compiler and hinder the efficiency of threading, the limiting block is the main reason that limits the gain of further parallelization and forms a bottle nick to any possible improvement and acceleration.

- The size of blocks loaded on each thread: Adding a relatively large blocks on threads is crucial to overcome threading overhead. In order to make an efficient use of any extra thread launched, a sufficient amount of computation load need to be available to keep that new thread busy. Otherwise, this increase of number of threads will contribute in slowing down performance and makes overall gain deteriorate.

CHAPTER 4 CONCLUSION AND RECOMMENDATIONS

4.1 Thesis summary

The main objective of this thesis was to present an enhanced and a more efficient way of conducting EMT type simulations that is faster and less time consuming. The main trigger of this project was the long waiting time needed to simulate large scale power network that are realistic and involve nonlinear devices, power electronics and have some sort of renewable sources penetration. The main case used in this project was the Hydro-Quebec grid benchmark that represents a simulated version of the complete Hydro-Quebec network with its extensions in the Canadian provinces of Quebec, Ontario and New Brunswick, and New York, Vermont, Massachusetts, and New Hampshire states in the United State of America. Simulating this case using a traditional solver was consuming a lot of computation time with the MANA matrix solution acting as the bottle neck of this simulation time delay. Another trigger of this work is the urgency of attaining real time simulation (or as close to real time as possible) for realistic and existing power grids. This PhD project is a step forward in reaching the ultimate goal of having an automatic real time EMT simulation package that requires no intervention of the user and provides accurate and reliable simulation results.

4.1.1 Sparse matrix package for EMTs (SMPEMT)

The new way of enhancing EMT simulation is based on accelerating the solution of a network $\mathbf{Ax = b}$ system of equation and provide a customized sparse solver that is suitable for electromagnetic transient studies. The new sparse solver is called sparse matrix package for EMTs (SMPEMT) and it has been validated and tested using the EMT simulation package EMTP-RV that used an iterative technique to solve nonlinear equations and hence involves more computations that other packages. The development of this sparse solver involved two major steps namely: Finding an existing and fast sparse solver and applying parallel computation to the new solver.

4.1.1.1 Replacing the Sparse solver package

Throughout this PhD project, several sparse solvers were considered to select a fast and reliable solver package to act as the baseline that the work and improvement will be based on. The survey of literature narrowed down the search to three solvers: KLU, SuperLU and EMTP-MDO. The

three solvers were studied and tested against each other and EMTP-MDO were found to be relatively faster than the other two. However, studying KLU and its features makes adopting KLU as the based solver more appealing than EMTP-MDO. This is mainly due to the potential improvement that may applied on KLU and the continuous support the package has by its developers. Many features were added to KLU and contribute in boosting its performance, these features are the following:

- **Pivot validity testing:** This feature was added to avoid unnecessary factorization during the $\mathbf{Ax = b}$ solution. The feature assumes the previously calculated pivot order is valid unless proven otherwise. This was done by making the refactoring technique of KLU as the default topology of updating LU factors. A test criterion was added in the refactor function to test the validity of the used pivot and flag any faults if detected. The same tolerance used in determining the pivot element is KLU is used in testing the validity of the previously calculated pivot in refactor function.
- **Partial factor:** This added feature to KLU is capable of reducing the computation load of any case by providing a mapping between the changed elements of the matrix $\mathbf{A}$ and different BTF blocks. By creating this mapping, only blocks with changed elements are factorized and the other unchanged blocks will be only solved using backward and forward substitution. In addition, using the refactor technique allows the partial factor technique to start refactoring process from the first left changed column (FLCC).

4.1.1.2 Applying parallel computation

Since BTF blocks are completely independent of each other, factorization and solving of these blocks in parallel was done by using parallel computation techniques OpenMP. OpenMP allowed to integrate the concept of parallel computation with minimum change of the sparse solver code. A load balancing technique was also developed to guarantee that all thread's load are balanced and match the load other threads are loaded with.

In conclusion, this PhD work enhanced the speed of EMT type simulation with the implementation of the new SMPEMT without jeopardizing the accuracy and precision of the simulation. The proposed SMPEMT solver accounts for varying topologies and the accurate iterative solution of

nonlinear models. The SMPEMT sparse solver is applicable to any software tool for the computation of electromagnetic transients. Moreover, the proposed enhancements to the KLU solver are applicable to other power system computation tools.

The computational gains are demonstrated for practical and large networks. The demonstration benchmarks and results constitute another contribution of this project.

4.2 Future work

Investigation a new In-Block-Permutation

Since the dynamic elements of the matrix A can be provided to KLU beforehand, it is worth investigation creating a new way of ordering the BTF blocks internally to reduce the amount of calculation KLU needs to refactor blocks. This new in blocks permutation will focus on individual blocks and push dynamic column to the right of the bock and all constant columns to the left of the block. This type of permutation will affect the fill-in reduction permutation used now in KLU and hence the challenge of this idea arises. The new permutation will have to combine the consideration of maximizing the constant part of the block (located at the left side of the block) and minimizes the dynamic part, and at the same time keeping the fill-in levels of L and U without big increase. This idea of IBP is similar to some extend to what is proposed in [34].

The application of METIS on single BTF blocks

BTF permutation in SMPEMT is based on the existence of constant parameters transmission lines in the case under study. Each block represents a part of the network that is isolated from the other parts of the network due to the time domain decoupling effect of the CP line model. These blocks do not have any lines in the part of circuit they represent and that may limit the ability of obtaining an efficient parallelization. This effect was seen in many cases in chapter 3 and the biggest block of most cases acted as a limiting factor of the parallel process. Adding the concept of METIS into SMPEMT will allow the solver to break these limiting blocks into boarder block diagonal format and allow to increase the parallelization degree of the case being studied.

Loading balancing technique

The loading balancing technique developed in SMPEMT is an efficient algorithm that provide a relatively efficient load balancing. However, it does take into account the size of constant regions and dynamic region of each block, and the integration of METIS or other permutation techniques will make such ordering obsolete.

Improving threads loading

It is notices throughout this PhD project that the efficiency of threading is based on the amount of work (load) assigned to threads. The more computation load threads have the better the performance. In the current SMPEMT implementation only $\mathbf{Ax = b}$ solution is solved in parallel. In addition to this part, many parts of EMT solution process can be added to the threads and be done in parallel. These parts include the solution of the control system, update of models, update history and so on….

REFERENCES

[1] T. A. Davis, and E. P. Natarajan, ”Algorithm 907: KLU, a direct sparse solver for circuit simulations problems,” ACM Trans. Math. Softw., Vol. 37, pp 36:1-36:17, September 2010.

[2] J. Mahseredjian, J. L. Naredo, U. Karaagac, and J. A. Martinez, “EMTP Off-line Simulation Methods and Tools for Electromagnetic Transients in Power Systems: Overview and Challenges,” *IEEE Power Eng. Soc. Gen. Meeting*, Minneapolis, Minnesota, USA, Jul. 25–29, 2010

[3] J. Mahseredjian, U. Karaagac, S. Dennetiere, H. Saad, “Simulation of electromagnetic transients with EMTP-RV”, Book, A. Ametani (Editor), “Numerical Analysis of Power System Transients and Dynamics”, IET (The Institution of Engineering and Technology), 2015.

[4] S.C. Eisenstat, M.C. Gursky, M. H. Schultz, A. H. Sherman. “Yale Sparse matrix Package”. Research report #114, Department of computer science, Yale University, 1982.

[5] J. Mahseredjian, V. Dinavahi and J.A. Martinez “Simulation Tools for Electromagnetic Transients in Power Systems: Overview and Challenges”, *IEEE Transactions on Power Delivery*, Vol. 24, Issue 3, pp. 1657-1669, July 2009.

[6] L. Gérin-Lajoie and J. Mahseredjian: “Simulation of an extra large network in EMTP: from electromagnetic to electromechanical transients”, Proc. of International Conference on Power Systems Transients, IPST 2009 in Kyoto, Tokyo, June 2-6, 2009.

[7] U. Karaagac, J. Mahseredjian, L. Cai, H. Saad, “Offshore Wind Farm Modeling Accuracy and Efficiency in MMC-Based Multi-Terminal HVDC Connection”, IEEE Trans. on Power Delivery, Vol. 32, No. 2, pp. 617-627, 2017.

[8] H. Saad, J. Peralta, S. Dennetière, J. Mahseredjian, et al., “Dynamic Averaged and Simplified Models for MMC-based HVDC Transmission Systems”, IEEE Trans. on Power Delivery, Vol. 28, Issue 3, pp. 1723-1730, July 2013.

[9] U. N. Gnanarathna, A. M. Gole and R. P. Jayasinghe, “Efficient Modeling of Modular Multilevel HVDC Converters (MMC) on Electromagnetic Transient Simulation Programs,” IEEE Trans. on Power Delivery, vol. 26, no. 1, pp. 316-324, Jan. 2011.

[10] B. Gustavsen, "Passivity enforcement of rational models via modal perturbation," IEEE Transactions on Power Delivery, vol. 23, pp. 768-775, Apr 2008.

[11] U. D. Annakkage, N. K. C. Nair, Y. Liang, A. M. Gole, V. Dinavahi, B. Gustavsen, et al., "Dynamic System Equivalents: A Survey of Available Techniques," IEEE Transactions on Power Delivery, vol. 27, pp. 411-420, 2012.

[12] A. Benigni, A. Monti, and R. Dougal, "Latency-based approach to the simulation of large power electronics systems," IEEE Transactions on Power Electronics, vol. 29, no. 6, pp. 3201–3213, June 2014.

[13] J. M. Bahi, K. Rhofir, J.-C. Miellou, "Parallel solution of linear DAEs by multisplitting waveform relaxation methods," Elsevier, Linear Algebra and its Applications, Aug. 2001, pp. 181-196.

[14] Y. Zhang, A. M. Gole, W. Wu, B. Zhang, H. Sun, "Development and Analysis of Applicability of a Hybrid Transient Simulation Platform Combining TSA and EMT Elements", IEEE Transactions on Power Systems, vol. 28, no. 1, pp. 357-366, 2013.

[15] S. Montplaisir-Goncalves, J. Mahseredjian, O. Saad, X. Legrand, A. El-Akoum, "A Semaphore-based Parallelization of Networks for Electromagnetic Transients", International conference on power system transients, 2013, Vancouver, Canada.

[16] D. Paré, G. Turmel, J.-C. Soumagne, V. A. Do, S. Casoria, M. Bissonnette, B. Marcoux, D. McNabb, "Validation tests of the Hypersim digital real time simulator with a large AC-DC network", International conference on power system transients, 2003, New Orleans.

[17] S. Abourida, C. Dufour, J. Belanger, G. Murere, N. Lechevin, and B. Yu, "Real-time PC-based simulator of electric systems and drives", Proc. 17th IEEE APEC, Applied Power Electronics Conf. and Expo., Mar. 10–14, 2002, vol. 1, pp. 433–438.

[18] R. Kuffel, J. Giesbrecht, T. Maguire, R. P.Wierckx, and P. G. McLaren, "RTDS-A fully digital power system simulator operating in real-time", Proc. EMPD'95, 1995, vol. 2, pp. 498–503.

[19] S. Fan, H. Ding, A. Kariyawasam, A. M. Gole, "Parallel Electromagnetic Transients Simulation with Shared Memory Architecture Computers", IEEE Trans. on Power Delivery, Vol. 33, Issue 1, 2018, pp. 239-247.

[20] J. Mahseredjian, S. Dennetière, L. Dubé, B. Khodabakhchian and L. Gérin-Lajoie, "On a new approach for the simulation of transients in power systems". Electric Power Systems Research, Volume 77, Issue 11, September 2007, pp. 1514-1520.

[21] J. Mahseredjian and F. Alvarado, “Creating an electromagnetic transients program in MATLAB: MatEMTP,” IEEE Trans. on Power Delivery, vol. 12, no. 1, pp. 380-388, January 1997.

[22] J. Mahseredjian, “Simulation des transitoires électromagnétiques dans les réseaux électriques,” Édition ‘Les Techniques de l'Ingénieur’, February 10, 2008, Dossier D4130. 2008, 12 pages.

[23] A. Abusalah, O. Saad, J. Mahseredjian, U. Karaagac, L. Gerin-Lajoie, I. Kocar. “CPU Based Parallel Computation of Electromagnetic Transients For Large Scale Power Systems”. IPST - International Conference on Power Systems Transients 2017, Seoul, Republic of Korea.

[24] A. Abusalah, O. Saad, J. Mahseredjian, U. Karaagac, L. Gerin-Lajoie, I. Kocar. “CPU based parallel computation of electromagnetic transients for large power grids”. Electric Power Systems Research 162 (2018) 57–63.

[25] I. S. Duff, J. K. Reid, "Algorithm 529: permutations to block triangular form", ACM Trans. on Mathematical Software, 4(2): 189-192, 1978.

[26] George Karypis and Vipin Kumar, “A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0”.

[27] J. Mahseredjian, C. Dufour, U Karaagac and J. Bélanger, “Simulation of power system transients using State-Space grouping through nodal analysis,” International conference on power system transients (IPST2011), Delft, Netherland, June 2011.

[28] F. Pellegrini, “Scotch and LibScotch 5.1 User’s Guide. User’s manual”, 2008

[29] D.P. Koester, S. Ranka, G. C. Fox, “ Parallel Block-Diagonal-Bordered Sparse Linear Solvers for electrical Power Systems Applications”, Presented at the Scalable parallel libraries conference, Mississippi State University, Mississippi, 6-8 October 1993.

[30] A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs”. George Karypis and Vipin Kumar. SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp. 359—392, 1999.

[31] W. F. Tinney, “Compensation Methods for Network Solutions by Optimally Ordered Triangular Factorization”, IEEE Trans. on Power Apparatus and Systems, vol. PAS-91, no. 1, pp. 123-127.

[32] W. F. Tinney, “Compensation Methods for Network Solutions by Optimally Ordered Triangular Factorization”, IEEE Trans. on Power Apparatus and Systems, vol. PAS-91, no. 1, 1972, pp. 123-127. (was 2 in jean’s)

[33] O. Alsac, B. Stott, W. F. Tinney, “Sparsity-Oriented Compensation Methods for Modified Network Solutions”, IEEE Transactions on Power Apparatus and Systems, Vol. PAS-102, no. 5, 1983, pp. 1050-1060. (was 3 in jean’s)

[34] H. W. Dommel, “Nonlinear and time-varying elements in digital simulation of electromagnetic transients,” IEEE Trans. Power App. Syst., vol. PAS–90, no. 6, 1971, pp. 2561-2567. (was 4 in jean’s)

[35] J. Mahseredjian, S. Lefebvre, X.D. Do, “A New Method for Time-Domain Modelling of Nonlinear Circuits in Large Linear Networks”, Power Systems Computation Conference 1993, Avignon, France, 1993. https://pscc-central.epfl.ch/repo/papers/1993/pscc1993_113.pdf (was 5 in jean’s)

[36] H. C. So, “On the hybrid description of a linear n-port resulting from the extraction of arbitrarily specified elements”, IEEE Trans. on Circuit Theory, vol. 12, 1965, pp. 381-387. (was 6 in jean’s)

[37] P. M. Lin, Formulation of hybrid matrices for linear multiports containing controlled sources. IEEE Trans. Circuit Theory, vol. CT-21, Mar. 1974, pp. 169-175. (was 7 in jean’s)

[38] L. O. Chua and L. K. Chen, “Nonlinear diakoptics. Proc. of the international symposium on Circuits and Systems”, Boston, Apr. 21-23, 1975, pp. 373-376. (was 8 in jean’s)

[39] J. Mahseredjian, S. Lefebvre and D. Mukhedkar, “Power Converter simulation module connected to the EMTP”, IEEE Trans. on Power Systems, vol. 6, no. 2, pp. 501-510, May 1991. (was 9 in jean’s)

[40] M. A. Tomim, J. R. Martí, and L.Wang, “Parallel solution of large power system networks using the Multi-Area Thévenin Equivalents (MATE) algorithm,” International Journal of Electrical Power & Energy Systems, vol. 31, no. 9, pp. 497–503, 2009. (was 10 in jean’s)

[41] F. A. Moreira, J.R. Martí, “Latency Techniques for Time-Domain Power System Transients Simulation,” IEEE Trans. on Power Systems, vol. 20, no. 1, pp. 246-253, 2005. (was 11 in jean’s)

[42] F. M. Uriarte, R. E. Hebner, and A. L. Gattozzi, "Accelerating the simulation of shipboard power systems," in Grand Challenges in Modeling & Simulation, The Hague, Netherlands, June 27 - 30, 2011. (was 12 in jean's)

[43] F. M. Uriarte, "On Kron's diakoptics", Electric Power Systems Research, vol. 88, pp. 146-150, 2012. (was 13 in jean's)

[44] Xiaoye S. Li, James W. Demmel, John R. Gilbert, Laura Grigori, Meiyue Shao, Ichitaro Yamazaki. " SuperLU Users' Guide". September 1999.

[45] Xiaoye S. Li, An overview of SuperLU: Algorithms, Implementation, and user Interface, ACM Transactions on Mathmetical Software, Vol. x, No. x, x 2004, Pages 1-24

[46] J. Demmel, S. Eisenstat, J. Gilbert, X Li, J. Liu, "A supernodal approach to sparse partial pivoting".

[47] J. R. Gilbert, T. Peierls, "Sparse partial pivoting in time proportional to arithmetic operations", SIAM J. Sci. Stat. Comput., 9(5): 862-873, 1988.

[48] Alan George and Joseph W. H. Liu, "The Evolution of the Minimum Degree Ordering Algorithm", SIAM Review, Vol. 31, No. 1 (Mar., 1989), pp. 1-19

[49] P. R. Amestoy, T. Davis, I. S. Duff, "An Approximate Minimum Degree Ordering Algorithm", SIAM J. Matrix Analysis & Applic., Vol 17, no 4, pp. 886-905, Dec.1996

[50] P. R. Amestoy, T. Davis, I. S. Duff, "Algorithm 837 : AMD, an approximate minimum degree ordering algorithm, ACM Transactions on Mathematical software, 30(3) :381-388, 2004

[51] T. A. Davis, J. R. Gilbert, S. I. Larimore, Esmond G. Ng. " A Column approximate minimum degree ordering algorithm", ACM Transactions on Mathematical software, 30(3) :381-376, 2004

[52] T. A. Davis, J. R. Gilbert, S. I. Larimore, Esmond G. Ng. " Algorithm 836: COLAMD, A Column approximate minimum degree ordering algorithm", ACM Transactions on Mathematical software, 30(3) :381-380, 2004

[53] R. Singh, A. M. Gole, C. Muller, P. Graham, R. Jayasinghe, B. Jayasekera, D. Muthumuni, "Using Local Grid and Multi-core Computing in Electromagnetic Transients Simulation", International conference on power system transients, 2013, Vancouver, Canada.

[54] Jayanta Kumar Debnath, Wai-Keung Fung, Aniruddha M. Gole, Shaahin Filizadeh "Electromagnetic Transient Simulation of Large- Scale Electrical Power Networks using Graphical Processing Units". 2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)

[55] OpenMP user manual, available online: http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf

[56] J. Mahseredjian, L. Dube, M. Zou, S. Dennetiere, and G. Joos, "Simultaneous solution of control system equations in EMTP," IEEE Trans. Power Systems, vol. 21, no. 1, pp. 117-124, February 2006.

[57] E. P. Natarajan, "KLU-A High Performance Sparse linear solver for Circuit Simulation Problems", Master Thesis, University of Florida, 2005.

[58] U. Karaagac, J. Mahseredjian, L. Cai, H. Saad, "Offshore Wind Farm Modeling Accuracy and Efficiency in MMC-Based Multi-Terminal HVDC Connection", IEEE Trans. on Power Delivery, Vol. 32, No. 2, pp. 617-627, 2017.

[59] A. Haddadi, J. Mahseredjian, "Power System Test Cases for EMT-Type Simulation Studies", CIGRE WG C4.503 report, 2018, pp. 1-142.

[60] http://www.ee.washington.edu/research/pstca/pf14/pg_tca14bus.htm