Accelerating Electromagnetic Transient Simulation of Electrical Power Systems Using Graphics Processing Units

By

JAYANTA KUMAR DEBNATH

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfilment of the requirements of the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made available to the public electronically.

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Aniruddha M. Gole for his valuable support, encouragement and valuable advice throughout this research work. Prof. Gole's immeasurable expertise and insightful suggestions and advices provided me invaluable motivation throughout the work of this thesis. His deep patience and encouragement continually helped me to improve my work.

I would also like to thank my committee members, Prof. Shaahin Filizadeh and Prof. Peter Graham, for their superb insights, support and assistance in bringing the thesis in its complete form.

My sincere thanks go to Dr. Fung, my previous academic advisor who hired me as a graduate student in this University.

Then I'd like to extend my sincere thanks to Prof. Douglas A. Buchanan for extending his helping hand to connect me with Prof. Gole at a very critical point of my program, when I was about to drop out due to extreme financial crisis.

I'd like to extend my sincere thanks to my fellow students and post doctors at the University of Manitoba, especially Dr. Tomas Yebra Vega, Dr. Hui Ding, Dr. Shengtao Fan for their friendly suggestions and discussions related to my work.

Special thanks to Mr. Erwin Dirks for his numerous help during my work in the lab.

I would also like to thank my parents for their continuous support and encouragement over all the long years I spent in school.

Finally I should express my most sincere appreciation to my beloved wife, Shilalipi, who never failed to encourage me in all the stages of my PhD program. She always showed priceless support beyond my imagination.

Dedication

I would like to dedicate this work to my mother, Mrs. Gita Rani Devi (গীতা রাণী দেবী).

Abstract

This thesis presents the application of graphics processing unit (GPU) based parallel computing technique to speed up electromagnetic transients (EMT) simulation for large power systems. Nowadays, desktop computers come with GPUs that support extra computing capability to handle gaming and animation related applications. GPUs are built with highly parallel computing architecture to support the high demand of graphics for various applications mainly related to computer graphics, video processing, and playback. It is possible to use these GPUs for general-purpose computations, such as EMT simulation. Power system components are mathematically modeled in their highest detail in EMT simulation. Traditionally, EMT simulation tools are implemented on central processing unit (CPU) based computers, where simulation is performed in a sequential manner. With the increase in network size there is a drastic increase in simulation time with conventional CPU based simulation tools. This research shows that the use of GPU computing considerably reduces the total simulation time. In this approach, GPU performs computationally intensive parts of the EMT simulation in parallel on its built-in massively parallel processing cores, and the CPU handles various sequential jobs such as flow control of the simulation, storing the output variables, etc.

This thesis proposes a parallel computing algorithm for EMT simulations on the GPU platforms, and demonstrates the algorithm by simulating large power systems. Total computation time for GPU computing, using 'compute unified device architecture' (CUDA)-based C programming is compared with the total computation time for the sequential implementations on the CPU using ANSI-C programming for systems of various sizes and types.

Special parallel processing techniques are implemented to model various power system components such as transmission lines, generators, etc. An advanced technique to perform matrix-vector multiplication in parallel on the GPU, is introduced and implemented in this thesis, which shows significant performance gain in the simulation. A sparsity-based technique for the inverse admittance matrix is implemented in this simulation process to ignore the multiplications involving zeros.

A typical power electronic subsystem is also implemented in this simulation process, which had not been implemented in the literature so far for GPU platforms. GPU computing-based simulation of large power networks with many power electronic subsystems has shown massive performance gain compared to conventional sequential simulations with and without the sparsity technique.

Finally, in this research work, the effect of granularity on the speed up of simulation was investigated. Granularity is defined as the ratio of the number of transmission lines used to interconnect various subsystems to the total size of the network. It should be noted that dividing a network into smaller subsystems requires additional transmission lines. Simulation results show that there is a negative impact on the overall performance gain of simulation with the use of excessive transmission lines in the test systems.

Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

EMT	Electromagnetic Transient
GPU	Graphics Processing Unit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
PCIe	Peripheral Component Interconnect Express
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
GPGPU	General Purpose GPU Computing
EMTP	Electro-Magnetic Transients Program
IGBT	Insulated-Gate Bipolar Transistor
HVDC	High-Voltage Direct-Current
FACTS	Flexible AC Transmission Systems
MATE	Multi-Area Thevenin Equivalent
SM	Streaming Multiprocessor
TL	Transmission Lines
API	Application Programming Interface
GUI	Graphical User Interface
TPC	Thread Processing Center

Chapter 1

Introduction

1.1 Background

Computer-based analysis and simulation of power systems have been used by engineers and researchers around the world for many years. Simulation-based study is mainly used to accurately analyze the behaviour of various power systems equipment, and plan the future expansions of the system that may include advanced equipment and techniques to ensure its stable operations. Electromagnetic transient (EMT) simulation is a widely used technique to analyze power systems transients in the range of nanoseconds to a few seconds [1]. This section briefly discusses the overview of the state of the art techniques used in EMT simulation. More details on this are introduced in the subsequent chapters.

Operation of any electrical power system involves continuous distribution of electromagnetic and electromechanical energy among various system components, such as generators, load, transmission lines, transformers, etc. [1, 2]. Continuous increase in demand for interconnection of various sub-systems has lead to the creation of extremely large and interconnected power systems [1, 3, 4, 5, 6]. In case of disturbances

caused by an unpredicted fault, lightning strikes or any other switching events, various system components (for example generators, transmission lines, transformers, etc.) are subjected to transients that may cause excessive currents or voltages in the network [1, 7]. In the early days (i.e. before digital computer based simulations), Transient Network Analyzers (TNA) were used to study transient effects [8]. TNAs use physical components such as inductors, capacitors and resistors to build a miniature physical model of the power network. Some of the advanced versions of TNAs have used analog simulator elements to model more complicated devices such as electrical machines, which were often emulated with micro-machines [8]. Some of the major limitations of TNAs are high cost, changes in behavior due to component aging, longer set-up time and difficulty in reproducing identical results at a later time [9], etc.

Analytical solution of large power systems is almost always impossible; therefore, numerical simulation is preferred instead [3, 10]. Digital computer-based transients simulations have alleviated many of these problems [9, 11]. As a result design and operation of modern power systems mainly depends on the available numerical simulation tools for electromagnetic transients (such as ATP [12], EMTP [2], PSCAD/EMTDC [13]) [3, 14]. It is to be noted that there are many techniques for digital computer-based EMT simulation, which are broadly classified into frequency-domain [15] and time-domain techniques [9]. In this thesis only the time domain technique is discussed (as it is the focus of this research work). EMT simulation is most commonly used to study fast transients such as lightning strikes, switching impact, power electronic converters, etc. In comparison to other available modeling approaches (such as transient stability studies [16, 17]), EMT simulation [1, 18, 19, 20, 21] models power system equipment in its greatest detail in the time domain. Due to the inherent complexity and computational intensity of EMT simulations, it was originally used for relatively small networks (i.e. networks with few hundred buses).

However due to improvements in computing power, EMT simulation is now widely used to study large (i.e. systems with thousands of busses) systems with fast dynamics. Digital computer-based electromagnetic transient simulation program (EMTP) for multi-node networks was first introduced by H.W. Dommel [2, 4].

EMT simulation models key aspects of the physics of the electromechanical apparatuses including the detailed functionality of analog and digital control systems, various types of switches including power semiconductor switching devices, etc. [18, 19]. Traditionally, EMT simulation tools are implemented on central processing unit (CPU)-based computers, where simulation is performed in a sequential manner. Therefore, conventional CPU-based simulation is time consuming. Many attempts have been made to reduce the simulation time, such as parallel implementation of the EMT algorithm [22], use of the Multi-Area Thevenin Equivalents (MATE) algorithm [23], multi-processor based parallel EMT simulation [16, 19, 22, 24, 25, 26, 27, 28, 29, 30], etc. Multi-processor based EMT simulation implements parallel processing techniques using special computers (such as supercomputers [16, 27]) and computers connected through the network (such as the use of PC clusters, etc. [19, 22, 24, 25, 26]). These approaches improved the performance of EMT simulation. However, the cost of installing a supercomputer and the delay in inter-PC communication in case of a PC-cluster are still major challenges. Even though all of these approaches have improved performance of EMT simulation, there is still room for more speed-up by further exploiting the parallelism inside the EMT algorithm.

This research work focuses on using Graphics Processing Units (GPUs) as a potentially cost effective (significant), high performance alternative to speed up EMT simulations. These days most computers come with built in GPUs, therefore no additional investment is necessary for the hardware. Additionally, the average cost

of one GPU card is around a few hundred dollars (at the time of writing this thesis), which has massive parallel computation capability. GPUs come with thousands of processors onboard capable of performing massive computations (in parallel) related to large matrices with less inter processor communications. These are specially designed hardware for applications such as graphics and visual computing [31]. However, GPUs are also excellent at many general purpose applications such as Monte Carlo simulation [32]. The author of this thesis reported the very first work on GPU based EMT simulation in a conference in 2011 [33]. In the subsequent years further progress of the work was reported [34, 35]. Recently Zhou *et al.* [36] presented a method to accelerate EMT simulation using massive thread capable GPUs. The approach presented by Zhou *et al.* [36] uses a special node mapping structure to exploit parallelism inside the EMT algorithm. In this case non-zero elements of the admittance matrix were rearranged to form a perfect block-diagonal matrix. Then the whole matrix was sub-divided into smaller sub-blocks (interconnected by virtual transmission lines) to represent much smaller decoupled sub-systems. All these sub-blocks were considered as dense matrices and the normal inversion method (without sparsity) was implemented to calculate the unknown node voltages. This approach has shown improvements in the speed up of EMT simulation. However, it did not capture the full benefit of parallelism as it internally treats the system matrix as smaller subsystems virtually decoupled by transmission lines, which require internal communications among those subsystems. Additionally, it introduces extra computational burden related to transmission lines (memory access calls, communication between the subsystems, etc.). Hence, the approach did not fully utilize the vast parallelism offered by the many core GPUs with less inter processor communications. Therefore, the author found that Zhou *et al.* [36] had not gone far enough. Inclusion of sparsity and investigation of interconnection ratio to trade-off mathematical computation effort versus communication bottlenecks could further increase computation speed. In

EMT simulations, the interconnection ratio (i.e. granularity) is defined by the ratio of the number of interconnecting transmission lines between the subsystems inside the networks to the size of the electrical network modelled on the GPU. The approach presented in this thesis organizes the parallelism in an alternative way that considers the equitable distribution of tasks for various processors on the GPU. Thus the proposed approach attempts to minimize inter-processor communication bottlenecks to achieve significant speedup in the simulation. In this research, the conventional EMT algorithm is explored and algorithmic steps that can be run in parallel are identified.

1.2 Objective of the research work

This research work uses Graphics Processing Unit (GPU)-based computing to accelerate EMT simulation for power systems. This section briefly outlines the background in choosing this specialized hardware and the programming techniques to program these GPUs. More details on GPU-based computing will be presented in Chapter 2.

As mentioned in the previous section, many attempts have been made to accelerate EMT simulation. Those approaches accelerated the simulation, however they did not explore the parallelism inside the algorithm, which is suitable for specialized hardware such as GPU. Therefore, the demand for further acceleration in the simulation continues. This demand for higher speed in computer-based simulations was not unique to the power systems engineers only. Computer users around the world demand faster computational speed. Additionally, the introduction of the Graphical User Interface (GUI)-based operating systems for computers accelerated the demand for higher speed to accommodate their rendering applications [31]. Tremendous demand in the last decade from the multimedia and gaming industries for accelerating 3-D rendering has driven researchers around the world including several graphics

hardware companies to the development of high-performance parallel graphics accelerators [31, 37]. This has resulted in the introduction of Graphic Processing Units (GPUs), which are available off the shelf at an affordable price to customers [31, 37]. Nowadays GPUs are an inevitable part of any standard computer (they are used to provide additional computing power to handle high performance gaming and animation related applications). In their early versions, GPUs were not suitable for programmers to implement general purpose computations. Starting from the generation of GPUs launched in 2002 and later (including NVIDIA GeforceFX series and ATI Radeon 9800 and above), developers can develop their own general purpose computations on a GPU. Any general-purpose computations could be accelerated by using these GPU-devices [31]. The main focus of this research is to apply these potentially cost-effective devices (i.e. GPUs) in accelerating EMT simulations. The key idea is to explore the massive parallelism inside the EMT algorithm to accelerate the computation process [24, 26]. Traditional application software is normally developed as a sequential program and executed on the CPU in a sequential manner [26, 31]. Starting around 2003, with the introduction of multi-core processors, parallel programming techniques are attracting more researchers to be involved in accelerating general purpose computations [11, 24, 25, 26, 27, 37]. GPUs have a large number of built-in processing units, capable of performing massive computations in parallel. Nowadays, these GPUs are continuously being used by researchers for accelerating EMT simulations [33, 34, 35, 36, 38, 39].

Accelerating general-purpose computations using GPUs with the help of a parallel programming language and special API (Application Programming Interface) that does not use the traditional graphics API and graphics pipeline model, is commonly known as GPU-computing [40, 41]. The Compute Unified Device Architecture (CUDA) is a commonly used parallel programming architecture to perform general-

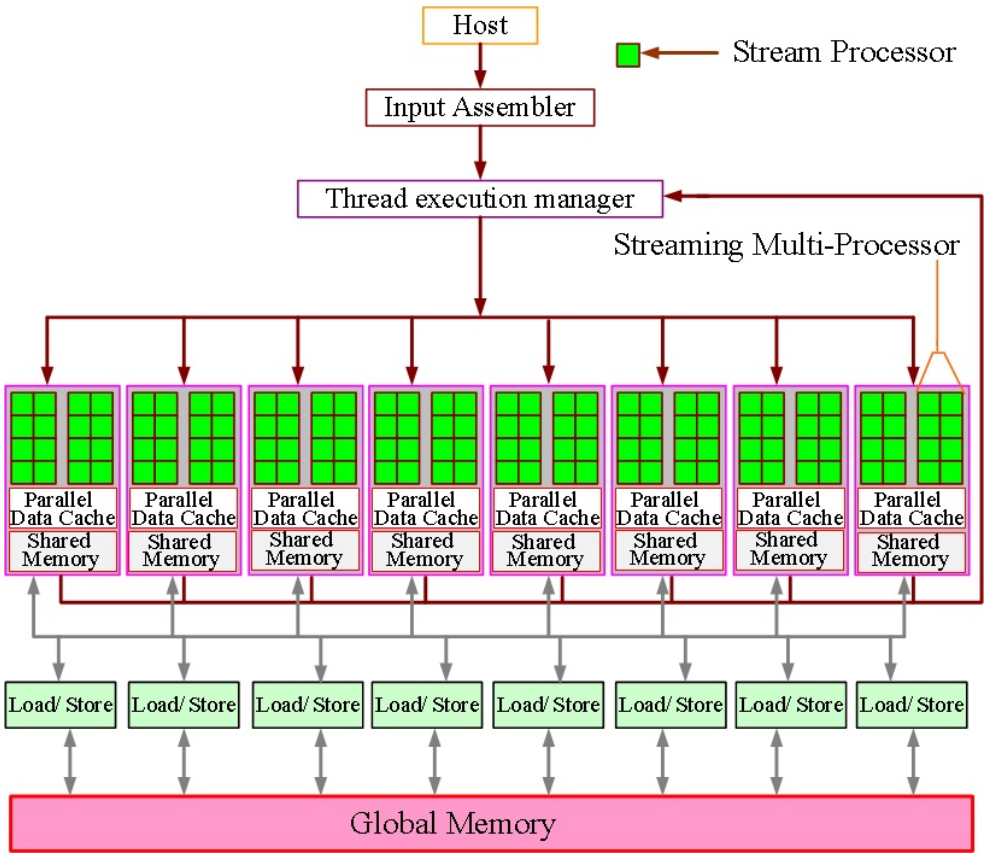


Figure 1.1: Schematic details of a CUDA-enabled (G-80) Graphic Processing Unit (GPU).

purpose computations on the GPUs from NVIDIA corporation [37]. There are several computer languages to program GPUs for general purpose computations (such as OpenCL [42, 43], CUDA C/C++ [37], etc.). Each of these programming techniques has its own merits and disadvantages [44]. Both OpenCL and CUDA-based C/C++ are commonly used programming language for GPU computing. However, OpenCL (first version was released several years after the release of CUDA) is still catching up to implement the programming features available in CUDA. Compared to CUDA, OpenCL still lacks some of the advanced debugging and profiling tools, such as cuda-gdb or cuda-memcheck, etc. [44]. In addition, the availability of the most robust drivers and Software Development Kits (SDKs) for general purpose computing from NVIDIA [37], has made CUDA based GPU programming a near universal tool.

Therefore, in this work the CUDA-based C programming technique was used, which was the only available tool to the author when he started this work. Fig. 1.1 shows the schematic of a CUDA enabled GPU (G80 architecture) [31]. It shows that there are 8-blocks of processors, commonly known as Thread Processing Centers (TPCs). Each TPC has two groups of 8-small processing elements. Each of these groups of 8-small processors is commonly known as a Streaming Multiprocessor (SM). It is due to the architectural issues of the GPUs, groups of processors in SMs are specially organized [31]. The number of processors in a SM may vary from one generation of GPU to another (for example, Fig. 1.1 shows the G-80 architecture with 8 processors in each SM). In general purpose computing, a GPU is used as a co-processor to the main CPU. The part of the computation task with intensive computation is accelerated by the GPUs, which execute the assigned tasks in parallel on multiple processing cores. The EMT algorithm, as will be explained in the upcoming chapters, also has massively parallel and computationally intensive parts suitable to be implemented on these specially designed hardware.

1.3 Overview of the thesis

The thesis provides a discussion of the present-day state of the art in GPU computing and power system electromagnetic transient simulation. It investigates methods to parallelize the simulation problem so that it can be assigned to multiple processors on the GPU. The thesis introduces a special approach that uses equitable distribution of the matrix-vector multiplication job on the GPU, which results in massive performance gains in the simulation. The thesis also presents performance improvements using GPU computing for various test cases involving large power networks. It provides a parametric analysis and comprehensive comparison between sequential computation time on the CPU and parallel computation time on the GPU for systems

of various sizes and types. The thesis consider an investigation of the granularity of the problem, i.e., using an increased number of transmission lines in the network to divide the network into smaller subsystems versus connecting larger subsystems with less number of transmission lines. Following this chapter, the rest of the thesis is organized as follows:

Chapter 2 includes an overview of GPU computing. This chapter starts with a brief introduction to GPUs, their hardware structure and finally an overview of GPU computing using Compute Unified Device Architecture (CUDA)-C is presented. This chapter also discusses some of the barriers, due to the special architecture of the GPUs, that limits the performance gain for general purpose computations.

In Chapter 3 an overview of EMT simulation is presented. This chapter discusses the preliminary concepts of EMT-simulation of a typical power network. This chapter also presents and compares the accuracy of the GPU based EMT simulation with a commercial EMT simulation tool (PSCAD/EMTDC) [13] for a typical circuit example.

Chapter 4 starts with a brief discussion on the state of the art techniques used to accelerate EMT simulation. This chapter then presents the details of the proposed algorithmic steps to adapt the conventional EMT simulation on the GPU platform to accelerate the computations. This chapter presents the proposed sparsity technique implementation details on the inverse admittance matrix to ignore multiplications involving zeros. This chapter also presents the various test cases used in this work. Two sets of test cases were selected and implemented with different level of granularity, which were simulated (simulation based performance is presented in the next chapter) using the proposed sparse and without sparse techniques. Finally, this chapter identi-

fies a special barrier on the GPU-based matrix-vector multiplication implementation, that limits the optimum performance gain in the GPU-based simulation process.

Chapter 5 presents the performance improvements using GPU computing to perform EMT simulations using parallel computing. This chapter also presents the disadvantage of using too many transmission lines in the network, while dividing the network into smaller subsystems. Finally, this chapter compares GPU-based simulated waveforms of the described EMT simulation with the corresponding results using commercially available tool.

Finally in Chapter 6 conclusions and future directions of this research work are presented.

Chapter 2

Overview of graphics processing units-based computing

2.1 Introduction

Graphics processing units are used in many electronic systems such as game-consoles, mobile phones, personal computers, etc. The GPU uses specially designed hardware, most commonly, to accelerate the rendering of images for display purposes. As mentioned earlier, the tremendous increase in demand for high performance gaming, video processing and animation related applications on personal computers, lead to the necessity for separate graphics processors [31]. The programmable GPU was first introduced by NVIDIA in 1999 [31, 37, 45]. At that time, this was the most extensive parallel processor to date with unprecedented floating-point performance and programming capability [31, 45]. Todays GPUs greatly outpace CPUs in general purpose computations in terms of computational throughput and memory bandwidth [43]. Nowadays, GPUs are commonly considered as an alternative to conventional CPUs in accelerating a variety of computationally intensive applications, such as computational fluid dynamics simulation [46].

This chapter starts with some brief historical background on GPUs. It then presents the detailed architecture of modern GPUs. Finally the programming techniques used for general purpose computing using GPUs and some barriers that limit the performance gain in GPU-based simulations are presented.

2.2 Overview of graphics processing units

The Central Processing Units (CPUs) in personal computers used to have a clock speed of around 1-MHz in the early 1980's [45]. The tremendous demand to speed up computations resulted in the increase of this speed to the 1-4 GHz range in more recent computers, which implies a thousand times faster computations in approximately 30 years. It should be noted that increase in clock speed results in more heat generation and imposes challenges to the heat removal mechanisms on the hardware [47]. This ultimately limits the size of integrated circuits, and the maximum clock-rate of CPUs to boost computation speed [41, 48]. This has resulted in extensive effort by researchers, engineers and computer manufacturers around the world to look for an alternative solution to meet the market demand for high performance gaming, video processing, etc. [31, 45]. Multiple processing cores are now available in the main CPUs to perform computations in parallel; this is an increasingly ubiquitous form of parallel processing [28, 29, 24, 26]. This multicore approach is intended to speed up computations by assigning parallel tasks to multiple processing cores in the CPUs. The very first CPU with two processing cores was introduced in 2005 [45]. In the following years, further versions of the CPUs were released with three, four, six and more cores. This trend of using multicore CPUs is also approaching its limit due to issues such as memory-bandwidth constraints, thermal effects, high power consumption, and the dimension of electronic components, etc. [38, 42, 47, 49, 50].

2.2.1 History of graphics processing units

In the 1980's and early 1990's the development and popularization of graphics driven operating systems (by companies such as Microsoft) [45] has increased the momentum to introduce 2D-display accelerators for personal computers. In 1992 Silicon Graphics introduced OpenGL library to program graphics accelerators [31, 42, 48]. Additionally, the continuously increasing consumer demand for 3D graphics, has provided opportunities for companies, such as NVIDIA, ATI technologies and 3dfx Interactive, to release affordable graphics accelerators [42]. NVIDIA released its GeForce 3 series in 2001, which was considered as an important breakthrough in GPU technology by giving programmers/developers some control over the exact computations to be performed on the GPUs [45]. In those GPUs developers had to use shader-languages (high-level graphics programming, based on the syntax of the C) or DirectX libraries to program the GPUs for general purpose computing [48]. Programmers had to fit the general purpose computations (i.e. mostly matrix computations) into the format used for graphics rendering. More recently in 2006, NVIDIA corporation released its parallel programming architecture, CUDA (Compute Unified Device Architecture) and compatible GPUs (GeForce 8800 GTX series) [31, 37, 45]. This was a major breakthrough to provide freedom to the programmers in terms of using these GPUs for any general purpose computations without using any graphics library.

Current GPUs have many independent processing cores, each capable of performing floating point operations (large volume) in parallel. Fig. 2.1 shows the schematic architecture of a CPU and a GPU. The CPU in this figure has only 4 cores and the GPU has 64 processing elements (i.e. GPU cores).

In a GPU, groups of several processing elements (the exact number of processors

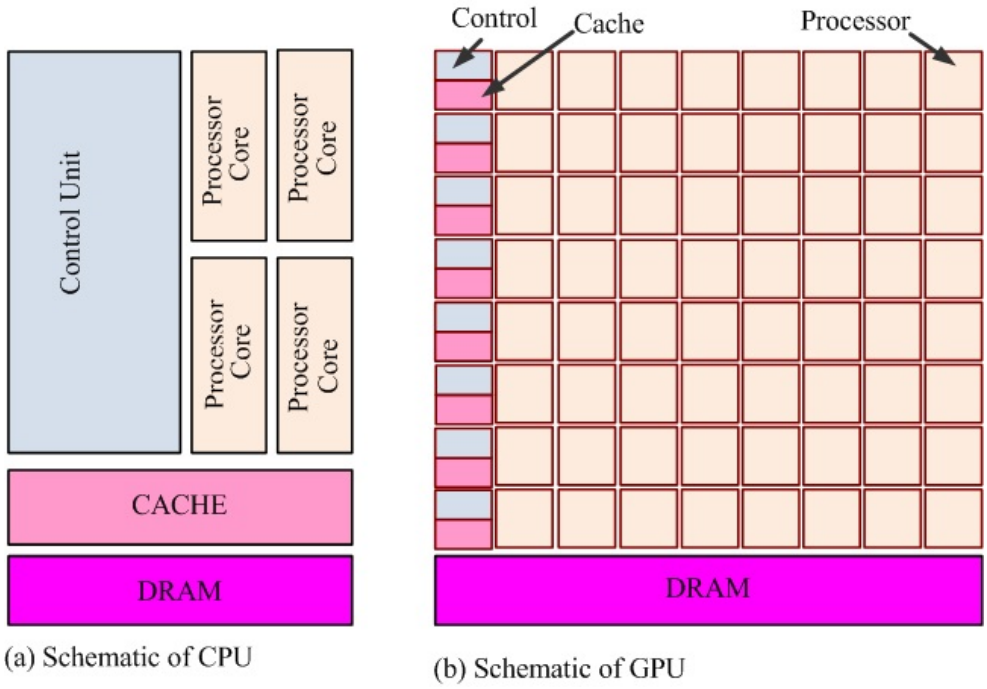


Figure 2.1: Schematic details of (a) a CPU architecture and (b) a GPU architecture [31].

varies across different generations of GPUs) commonly known as Streaming Multiprocessors (SMs). These SMs are used to provide parallel computing ability in the Single Instruction Multiple Thread (SIMT) fashion [31] (explained later). Each streaming multiprocessor has a small amount of shared (cache) memory and a control unit to control the thread operations that specify the code sequence for the cores in the SM. In general GPUs have more cores or processing elements compared to CPUs and are capable of performing large computations in parallel. For example the Intel core i7 CPU has only four cores while the NVIDIA GeForce GTX 590 GPU has 512 cores. It should be noted that due to the lack of enough space to accommodate large cache memory and control units as used in CPU cores, each GPU cores are designed to be very light weight with only one Arithmetic Logic Unit (ALU), which support addition, subtraction and multiplication. Therefore, each GPU processors has very limited resources, which limits the computational capability compared to a CPU-

processor. Hence, GPU processors can not be considered as an alternative to those of the CPUs. Additionally, in each SM there is a Super Function Unit (SFU) to perform computations involving special functions such as logarithms [31]. These days GPUs are considered as a significant computing resource for computers [49]. Recent GPUs have more transistors than modern CPUs [48]. For example the Intel core i7 CPU has about 731 million transistors whereas Nvidia GeForce GTX 590 has more than 3 billion transistors. Transistors in graphics cards are used for the implementation of Single-Instruction Multiple-Data (SIMD) units [31] suitable for high performance parallel computing (i.e. the computations only require repeated identical operations on collections of identical data). On the other hand, many of the transistors on the CPU are dedicated to support non-computing tasks such as branch prediction and caching, etc. As a result CPU performance is better on traditional sequential algorithms.

2.3 Overview of NVIDIA's CUDA

2.3.1 Choosing CUDA-C as a programming language

Use of GPUs for general purpose computing (as opposed to graphics acceleration) started in 2001 [31, 48] and is commonly known as 'General Purpose GPU' (GPGPU) computing [40]. Some applications of GPGPU are real-time computer vision [49], fluid dynamics simulations, molecular dynamics simulations [41], Monte Carlo simulation [32], power system stability studies [38], EMT simulations [36, 33, 39] and so on. Programming languages such as CUDA-based C or Fortran [37], OpenCL [42, 43], DirectCompute [48], etc. can be used to program GPUs for general purpose computations. CUDA-based programming is specific to NVIDIA GPUs [31, 42], while OpenCL is an open standard that can be used to program any hardware including CPUs, GPUs, and other devices from different vendors [42, 43]. OpenCL 1.1 became

available on June 14, 2010 (latest version is OpenCL 2.0) and provided comparable flexibility and programming capability with other programming techniques such as CUDA-based C/C++ [51]. Both CUDA and OpenCL based GPU computing are gaining popularity among general purpose programmers /developers. Although OpenCL provides much of the functionality of CUDA, it is still a work in progress and as of yet does not provide the same level of debugging and profiling tools [44]. CUDA-C is an ANSI C based programming technique with some special keywords, introduced to access the GPU. Due to its inherent similarity with standard C programming, CUDA-C has become more widely used by the majority of the programmers and developers [44]. For the above reasons, CUDA-based C programming is used in this thesis to access the GPUs to implement EMT-simulations.

2.3.2 CUDA based programming

CUDA-C based GPU computing does not require the programmer to be familiar with any graphics API (Application Programming Interface), which greatly reduces the programming effort for general purpose programmers. With the support of its associated specially-built hardware, CUDA provides an effective SIMD-based programming model to implement parallelism in general purpose computations [31]. NVIDIA also released its own compiler for CUDA that distributes parts of the job to be implemented between the CPU and the GPU [37].

A general CUDA program consists of two different parts as specified by the programmer, a) a portion of the program to be executed on the CPU (mostly the sequential portion of the program) and b) a portion of the program to be executed on the GPU (parallel part of the program) [31, 45]. The NVIDIA-C compiler differentiates between codes that will be part of the CPU job and the GPU job during the compilation process. The GPU portion of the program starts with the invocation of a

kernel function, which contains instructions that are to be performed in parallel on the GPU. A *kernel* function generates a large number of threads arranged in various blocks to implement SIMD-based data parallelism. The compiler specifies the number of blocks and threads to be generated during the *kernel* launching to execute the portion of the program in parallel. All threads generated in a single launch of the *kernel* function implement the same instruction in parallel. Therefore, this implementation is commonly known as Single-Instruction Multiple-Thread (SIMT) [31, 45]. SIMT based parallel programming is used in this work. The SIMT is an extension of the commonly known SIMD mode of parallelism [31]. SIMD implements parallelism by executing an instruction on various data. On the other hand, SIMT introduces dynamic way of parallelism that implements thread level execution of an instruction on various data [31]. In this SIMT mode, each instruction is executed with different data in parallel by multiple threads that run on the identical GPU cores [31, 37, 45].

Fig. 2.2 shows the schematic details of the execution of a CUDA program. This program has serial as well as large parallel parts. The part of the program with very little or no data parallelism is executed on the CPU and the rest of the program with massive data parallelism is implemented on the GPU. The execution of the program starts on the CPU, and with the launch of a *kernel* function, task execution is shifted to the GPU. The set of threads generated by a *kernel* call is collectively called a *grid*. For example, Fig. 2.2 has two different grids. As soon as all the threads in a single *kernel* finish their computations the corresponding grid terminates execution and program execution control is shifted to the CPU. In GPU computing, parallel part of the program is divided into number of smaller block (block size is fixed by the programmer). Each of these blocks contains parallel threads to perform actual computations on the GPU cores. Collection of blocks involved in a single *kernel* are commonly called a grid. More detail on these may be found in [31, 37, 45]. Fig. 2.2

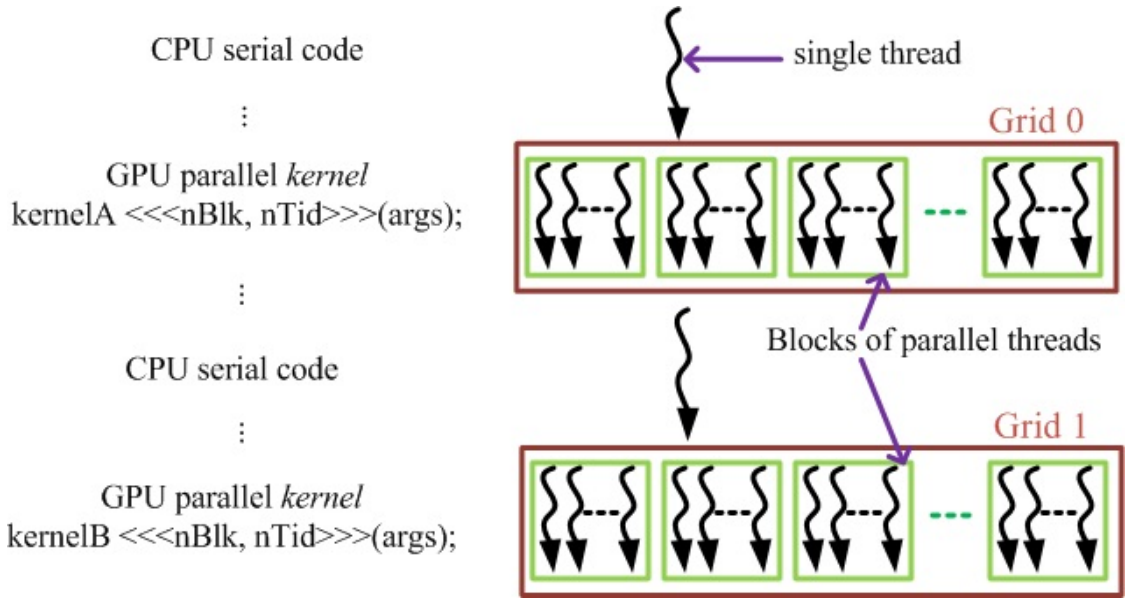


Figure 2.2: Schematic details of a CUDA program execution.

shows the invocation of two grids involving various blocks and parallel threads. It is to be noted that the terminologies used in this thesis are specific to CUDA-based programming techniques, in other programming techniques the same topic might have a different name (for example *thread blocks* are known as *work-groups* in OpenCL).

2.4 Challenges to general purpose computing on the GPU

A major challenge in obtaining the optimal performance on the GPU comes from the task scheduling, which is commonly known as *warp*-scheduling. A *warp* currently consists of 32 threads on NVIDIA hardware and 64 threads on AMD hardware (the number may vary depending on the design of the hardware) [31]. All the threads in a *warp* execute the same instruction in lock-step (i.e. various *warp* groups cannot communicate or share information). To explain this lets consider a *kernel* function with 24 blocks having 4 parallel threads in each block. To fit this job on the GPU,

the CUDA compiler will split it into *warp* groups with 8 blocks in each warp. In this case the *kernel* function will have 3 *warp* groups. The data that various *warp* groups operate on (i.e. thread level) is different. So in case the program has any statement that causes the threads in a *warp* to be unable to execute the same instruction, such as branching depending on some conditional statements, some threads in the warp will be moved to another *warp* during the execution of that instruction. So there would be some loss of the available computational power of the GPU. Additionally, the whole job will be scheduled on the GPU in a group of *warp*s. Therefore, in case the total threads in the job are not integer multiples of the *warp* size then there would be some *warp* groups having less than the maximum number of threads, which will result in less performance in the computations. It is the programmer's responsibility to ensure the effective use of the available computational power due to *warp* scheduling.

The second challenge to maximize performance in GPU execution comes from the lack of adequate data cache on the GPU for individual processors (commonly known as shared memory, which is very quickly accessible by the processors and therefore critical in obtaining high performance). It should be noted that due to the lack of enough space and architectural constraints the GPUs come with a very small amount of shared memory [31, 37] (for example, in the case of $G80$ GPUs the shared memory size is only $16kB$ for each streaming multiprocessor). Therefore, the programmer has to design the algorithm to efficiently use this shared memory to get the optimal performance in the computations.

The final challenge in achieving the best possible performance from GPUs comes from the existence of branching in the algorithm. In general EMT simulations may involve many switching instances, which require the algorithm to meet some preset condi-

tions involving some logical operations (such as AND, OR, etc. to identify the exact instant of switching). The GPUs are specially designed hardware to work on highly parallel job such as video processing [48]. However, the GPUs lack some fundamental computing based on integer and associated operations such as bit-shifts and bitwise logical operations such as (AND, OR, XOR, NOT). Therefore applications involving logical operations are not easily accommodated on GPUs [48, 41]. Hence, applications involving many conditional statements or logical operations are recommended to be performed on the CPU (that reduces the decision making time in branching and hence improves overall simulation speed). More details on the performance limitations of GPU computing may be found in [31, 37, 41, 43, 45, 48].

2.5 Chapter summary

This chapter briefly discussed the background of multi-core CPU and GPU computing. The advantages and disadvantages of different programming interfaces were presented and the reasons for using CUDA-based C programming language in this work were justified. Finally, this chapter introduced some of the barriers in achieving optimal performance gain in the GPU-based simulations. The next chapter presents the basic steps of sequential EMT simulations.

Chapter 3

Introduction to electromagnetic transients simulation

3.1 Simulation tools for power systems

Transients in electrical power networks are typically caused by short circuits, lightning strikes, switching of various power electronic equipment and so on. A transient is a sudden change in a system's states from its equilibrium condition [1, 4]. A transient may sometimes results in high current or voltage variation in the network [1, 19]. Power systems must be able to withstand such transient effects without any damage to the network and its components [4, 17, 52]. Due to their presence in vast geographical areas, power systems are usually complex networks and do not lend themselves to analytical solution of the transients. Therefore, digital computer-based simulation tools are commonly used to simulate the behaviour of power systems. Digital computer-based simulation tools are mainly classified into, (a) load flow study tools, (b) transient stability study tools, and (c) electromagnetic transients simulation tools.

3.1.1 Load flow study tools

Load flow simulation tools numerically calculate the steady state behaviour of power systems, such as flow of electric power through the system, voltage profile for the whole network, etc. [1]. This types of simulation uses simplified phasor models for various power system equipment, per-unit system based computations, and so on. Load flow studies are used to determine AC power system quantities such as voltage, power factor, real power and reactive power, etc. [17]. Load flow studies are used to determine overloading of equipment, such as transmission lines, generators, etc.

3.1.2 Transient stability study tools

Transient stability tools are used to study the electro-mechanical transients in a large power system located over a wide geographical area. These tools capture the slow transient behaviour of power systems in the range of 10^{-1} sec to 20 sec (mainly due to electromechanical devices in the network) [9]. On the other hand electromagnetic transients simulation tools are used to represent the system behaviour in the range of 10^{-7} sec to 10^{-1} sec, which are considered as fast transients in power systems. Transient stability-based simulation techniques represent the slow-moving mechanical parts such as generator rotors, governors, etc. with differential equations, which are numerically integrated. The AC networks are represented by phasor components. Changing quantities in the electrical network are captured via different phasor solutions in each time-step.

3.1.3 Electromagnetic transients simulation tools

EMT simulation is the most comprehensive approach for transient simulation compared to the other available approaches. EMT simulation represents network elements in their greatest detail by modelling the physics of various electromechanical appa-

ratuses including the functionality of analog and digital control systems, switches, etc. [18, 19]. Most EMT programs include a comprehensive library of commonly-used power system components. However, the available models may sometimes not represent specific advanced devices, for example an advanced control system for a specific power electronic device may not be available in the library. In this case, specialized programs may be used. As an example, to design power electronic converters including the precise representation of the power semiconductors, including the detailed physics based behaviour of the semiconductors may require a special program. Commonly-used commercially available EMT simulation tools include MicroTran (Microtran Power Systems Analysis Corporation, University of British-Colombia version of the EMT simulation program) [53], ATP (Alternative Transients Program) [12], PSCAD/EMTDC of the Manitoba HVDC Research Center [13], RTDS (Real Time Power System Simulation) [11, 54], EMTP-RV [55], etc. This chapter presents the basic steps of conventional EMT simulation and explains the details of EMT simulation techniques.

3.2 Overview of network elements substitution

Analysis of power systems starts with the appropriate modelling of various equipment of the system. A model of an equipment consists of the mathematical formulation of the system using sets of algebraic equations or relations that appropriately describes the physical behaviour of the system. Power systems are modelled using lumped elements such as resistances (R), capacitances (C), inductances (L) and distributed elements such as transmission lines and cables [1, 2]. This section outlines the basic steps in EMT simulation of power systems.

3.2.1 EMT models of basic resistive (R), inductive (L), capacitive (C), mutual-inductive (M) components

The voltage across a resistor is given by the following equation (commonly known as Ohm's law [56]). In this equation, v_R is the voltage across the resistor, i_R is the current through the resistor and R is the resistor value.

$$v_R = R.i_R$$

The voltage across a capacitor, $v_C(t)$, is mathematically expressed in terms of the current through the capacitor, $i_C(t)$, by the following equation [56]:

$$v_C(t) = \frac{1}{C}\int_{-\infty}^{t} i_C(t)dt$$

Similarly, current through an inductor, $i_L(t)$ is expressed in terms of the the voltage across the inductor, $v_L(t)$ by the following equation; details may be found in [56]:

$$i_L(t) = \frac{1}{L}\int_{-\infty}^{t} v_L(t)dt$$

Similar differential or algebraic equations can be developed for all other elements in power systems. Hence, the mathematical model of an electrical network consists of a set of ordinary differential equations, where resistances, capacitances and inductances are taken as network parameters. In general, numerical techniques are applied to solve these equations. Numerical integration techniques, such as the trapezoidal rule, have been successfully employed to solve these differential equations [1, 2]. In the most commonly-used approach (due to Dommel [2]) lumped parameters such as capacitors and inductors of a power system are modelled as Norton equivalent sources [56] as shown in Fig. 3.1. In this figure, $i(t)$ is the current through the capacitor (the

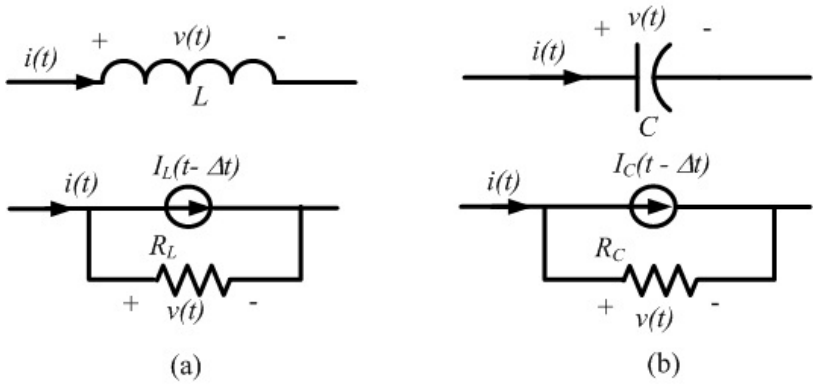


Figure 3.1: Schematic equivalent circuit for (a) an inductor and (b) a capacitor following trapezoidal rule based formulation.

inductor) while $v(t)$ is the voltage across the capacitor (the inductor). $R_C = \frac{\triangle t}{2C}$ and $R_L = \frac{2L}{\triangle t}$ are the Norton equivalent resistances of the capacitive and the inductive components, respectively ($\triangle t$ is the time step used in the simulation). $I_C(t\text{-}\triangle t)$ and $I_L(t\text{-}\triangle t)$ are the history current terms for the capacitive and the inductive branches, respectively, and are defined by the following equations [2]:

For an inductive branch (i.e. Fig. 3.1a):

$$\begin{aligned} i(t) &= I_L(t - \triangle t) + \tfrac{\triangle t}{2L} \times v(t), \\ \text{and} \quad I_L(t - \triangle t) &= i(t - \triangle t) + \tfrac{\triangle t}{2L} \times v(t - \triangle t) \end{aligned} \tag{3.1}$$

For a capacitive branch (i.e. Fig. 3.1b):

$$\begin{aligned} i(t) &= I_C(t - \triangle t) + \tfrac{2C}{\triangle t} \times v(t) \\ \text{and} \quad I_C(t - \triangle t) &= -i(t - \triangle t) - \tfrac{2C}{\triangle t} \times v(t - \triangle t) \end{aligned} \tag{3.2}$$

where $\triangle t$ is the time step used in the simulation process.

Computer-aided simulation of any electrical network requires representation of each capacitive and inductive element using its Norton equivalent model as shown in Fig. 3.1. Transformers also have similar equivalent models [1] and result in multi-port

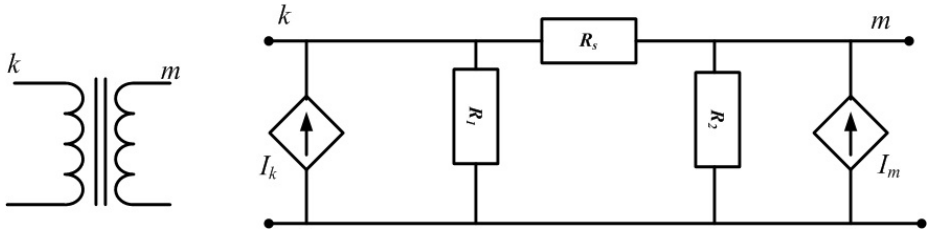


Figure 3.2: Schematic of single phase transformer model as used in this work.

Norton equivalent forms [1]. Fig. 3.2 shows the schematic equivalent of a single phase transformer using the above mentioned integration technique. In this case (i.e. in Fig. 3.2) various inductances of the transformer are mathematically replaced using the above trapezoidal integration-based representation and replaced by R_1, R_2, and R_s, respectively. The currents I_k and I_m (in Fig. 3.2) are the Norton equivalent current after replacing the transformer equivalent inductances using trapezoidal integration based representation. In Fig. 3.2 k represents the primary and m represents the secondary side of the transformer. More details on transformer modeling may be found in [1]. A similar Norton equivalent [56] representation may be developed for other network elements such as transmission lines and cables (more details on modelling transmission lines will be presented in Chapter 4). Switching systems including semiconductor equipment such as thyristors or IGBTs (Insulated Gate Bipolar Transistors) in HVDC (High Voltage Direct Current) and FACTS (Flexible Alternating Current Transmission System) devices can be represented by an equivalent resistor whose value is set according to the switching state of these devices.

3.2.2 Solution of the admittance matrix based formulation of EMT

After replacing every component in the network with its Norton equivalent model a purely resistive network emerges, which can be solved (for unknown voltages) using a nodal admittance-based approach [1, 13, 21, 25, 56, 52]. The mathematical repre-

sentation (using Nodal analysis [56]) of any power system has the following general form [1, 4, 21, 38] at any instant of time, t:

$$[Y] \times [V] = [J] - [I_H] \tag{3.3}$$

where:

$[Y]$ is the nodal admittance matrix.

$[V]$ and $[J]$ are vectors of various node voltages and of injected currents at various nodes at instant t, respectively.

$[I_H]$ is the vector of history currents injected at various nodes. They are calculated from the system states in the previous time-steps.

The only unknown in (3.3) is the vector of node voltages, $[V]$ [1, 2, 25]. The solution of (3.3) requires calculation of the inverse of the admittance matrix, $[Y]$.

3.2.3 Inclusion of switches

In the case there is no switching equipment in the network, the admittance matrix is fixed for the whole duration of the simulation. Therefore, calculation of the inverse of the admittance matrix, $[Y]$, is required only at the beginning of the simulation process. In general, networks having power electronic components will require on the fly calculation of the inverse of the admittance matrix (as the network admittances are no longer fixed). There are some alternative methods to avoid direct calculation of the inverse of the admittance matrix, such as LU-decomposition technique [57]. In this thesis the binary type of switching technique was implemented. The binary type of switching algorithm [58] is also an alternative option, where various admittance matrices for different switching events are formed at the beginning of the simulation process [5]. The inverses of those matrices are calculated at the beginning of the

simulation process. Hence, the potential inverses of the anticipated post-switching configurations are pre-calculated and inserted in the solution when required. In this work, the Gauss-Jordan method based general inversion method was implemented on the CPU [57] to perform computations related to inverse admittance matrices.

Transmission line model (TLM) technique [59] is another popular method to model power electronics (i.e. switches) in EMT-simulation, which could be used as an alternative option of having fixed admittance matrix for the entire simulation process. Fixed admittance matrix does not require on the fly calculation of the inverse of the admittance matrix. In this technique (i.e. TLM) the ON state and the OFF state of a power electronic device is represented by an inductive and a capacitive branch, respectively. However, the challenge of this approach is the selection of appropriate values for the inductive and capacitive components in representing the switches [59]. Details on recommended practices regarding modeling of non-linear power electronics equipment may be found in [5, 7, 9, 58, 60, 61].

In the following section an example of a typical electrical circuit simulation is presented, which uses the above mentioned techniques. This circuit is simulated on the commercial tool, PSCAD/EMTDC, and also using GPU-computing technique (GPU-computing technique for EMT-simulation will be presented in Chapter 4). The output voltages and currents are also compared to demonstrate the accuracy of the GPU-based simulation.

3.3 Typical example of EMT simulation

Simulation of the circuit shown in Fig. 3.3 (without switching) using the techniques presented above will be discussed in this section. This circuit has a DC voltage source,

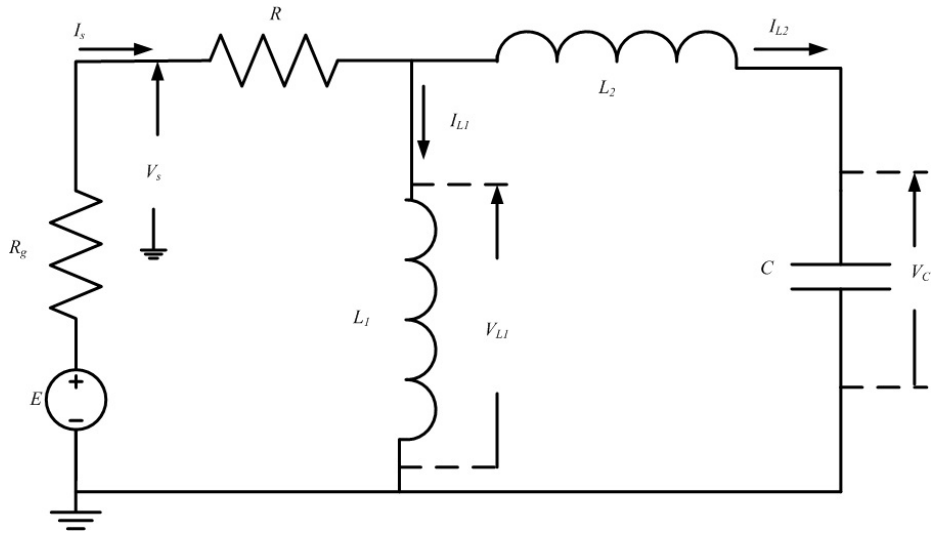


Figure 3.3: Typical electrical circuit for EMT-simulation, as an example.

$E = 10V$, other circuit parameters are, $R_g = 2.0\Omega$, $R = 3.0\Omega, L_1 = 0.10H$, $L_2 = 0.05H,$ and $C = 100\mu F$, respectively. The parameters of interest of this circuit are (as shown in Fig. 3.3), V_s voltage across the source, V_{L1} voltage across the inductor I_{L1}, V_C voltage across the capacitor C, I_s current through the resistor R, I_{L1} current through the inductor L_1, and I_{L2} current through the inductor L_2. In the first step of the simulation, the circuit components (of Fig. 3.3) are replaced with their Norton equivalent models to form the desired nodal-admittance matrix. Fig. 3.4 shows the equivalent circuit after substituting Norton equivalents for all the inductors and capacitor of the circuit. As can be seen, this equivalent circuit in Fig. 3.4 has three nodes as marked by the dotted circles (colored). History currents for the various Norton equivalent sources of Fig. 3.4 are defined by the equations below, where $\triangle t$ is the time step used in the simulation process:

For inductor L_1 in Fig. 3.3:

$$I_{L1}(t) = I_{L1H}(t - \triangle t) + \frac{1}{R_{l1}} \times V_{L1}(t), \quad \text{where } R_{l1} = \frac{2 \times L_1}{\triangle t}$$

$$\text{and} \quad I_{L1H}(t-\triangle t) = I_{L1}(t-\triangle t) + \frac{1}{R_{l1}} \times V_{L1}(t-\triangle t)$$

where, $I_{L1}(t)$ is the current through the inductor L_1, V_{L1} is the voltage across the inductor L_1 and $I_{L1H}(t-\triangle t)$ is the history current of inductor L_1.

For inductor L_2 in Fig. 3.3:

$$I_{L2}(t) = I_{L2H}(t-\triangle t) + \frac{1}{R_{l2}} \times (V_{L1}(t) - V_C(t)), \quad \text{where } R_{l2} = \frac{2 \times L_2}{\triangle t}$$

$$\text{and} \quad I_{L2H}(t-\triangle t) = I_{L2}(t-\triangle t) + \frac{1}{R_{l2}} \times (V_{L1}(t-\triangle t) - V_C(t-\triangle t))$$

where, $I_{L2}(t)$ is the current through the inductor L_2, V_C is the voltage across the capacitor C and $I_{L2H}(t-\triangle t)$ is the history current of inductor L_2.

For capacitor C in Fig. 3.3:

$$I_C(t) = I_{CH}(t-\triangle t) + \frac{1}{R_C} \times V_C(t), \quad \text{where } R_C = \frac{\triangle t}{2 \times C}$$

$$\text{and} \quad I_{CH}(t-\triangle t) = -I_C(t-\triangle t) - \frac{1}{R_C} V_C(t-\triangle t)$$

where, $I_C(t)$ is the current through the capacitor C and $I_{CH}(t-\triangle t)$ is the history current of capacitor C.

The resulting mathematical representation of the equivalent circuit in Fig. 3.4 at any instant of time t, using Nodal analysis [56], is shown below:

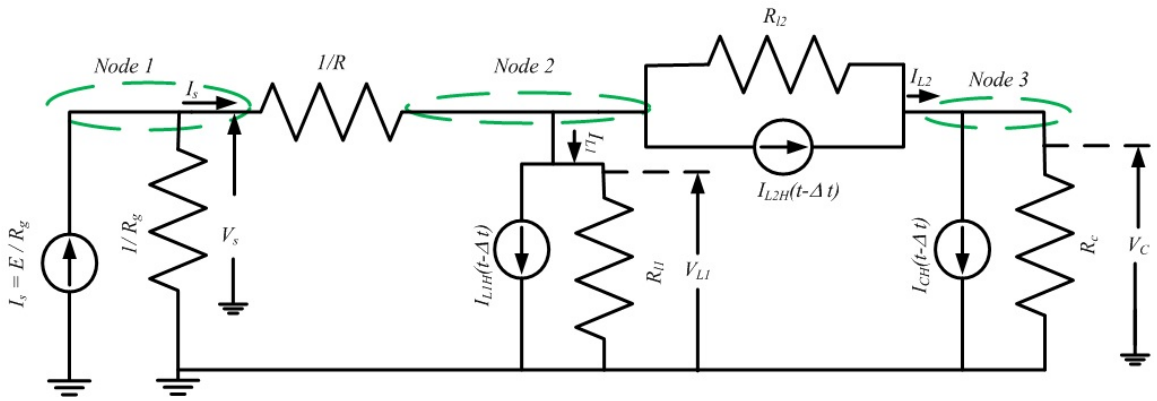


Figure 3.4: Equivalent circuit after replacing Norton equivalents for Capacitors and Inductors following Dommel's [2] formulation in circuit of Fig. 3.3.

$$\begin{bmatrix} (\frac{1}{R_g}+\frac{1}{R}) & (-\frac{1}{R}) & 0 \\ (-\frac{1}{R}) & (\frac{1}{R}+\frac{1}{R_{l1}}+\frac{1}{R_{l2}}) & (-\frac{1}{R_{l2}}) \\ 0 & (-\frac{1}{R_{l2}}) & (\frac{1}{R_{l2}}+\frac{1}{R_C}) \end{bmatrix} \times \begin{bmatrix} V_s(t) \\ V_{L1}(t) \\ V_C(t) \end{bmatrix} = \begin{bmatrix} \frac{E}{R_g} \\ -I_{L1H}(t-dt) - I_{L2H}(t-dt) \\ I_{L2H}(t-dt) - I_{CH}(t-dt) \end{bmatrix} \tag{3.4}$$

The above equation (i.e. 3.4) involves matrix-vector multiplication. The only unknown in this equation is the vector of various node voltages (i.e. $[V_s(t)\ V_{L1}(t)\ V_C(t)]^T$), at instant t. All sources are either known or in the case of history currents are known from the earlier timestep. Solving this equation for unknown node voltages requires calculation of the inverse for the admittance matrix. The inverse of the admittance matrix for this particular circuit can be precomputed as there is no switching. The admittance matrix inversion is usually performed at the start of the simulation process or when a circuit breaker or power electronic switch operates and changes the circuit's impedances. This admittance matrix was inverted using the Gauss-Jordan method-based general inversion method [57], as mentioned earlier. To solve for the

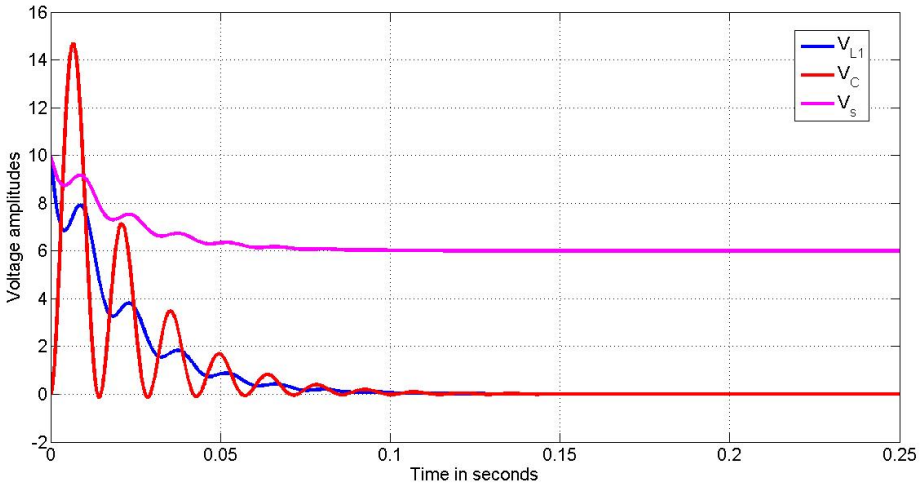


Figure 3.5: Simulated voltage waveforms for the circuit of Fig. 3.3 using PSCAD/ EMTDC [13].

unknown voltage (or current) of (or through) a particular node (branch) for a particular duration of time, various steps of the simulation processes (e.g. matrix-vector multiplication, history current updating and source current updating, etc.) are repeated iteratively with a time step of $\triangle t$. The simulation process starts from the instant $t = 0$ and continues until the end of the simulation period is reached. It should be noted that other formulations are also possible to simulate any electrical network, such as MNA (Modified Nodal Analysis) [62], in which the unknown vector is a combination of voltages and currents. However, these methods have not been implemented in this thesis.

3.3.1 Simulation using CPU-based tool PSCAD/ EMTDC and using GPU computing

Fig. 3.5 shows various node voltages for the circuit of Fig. 3.3, simulated using commercially available (CPU based) simulation tool PSCAD/EMTDC [13]. As is obvi-

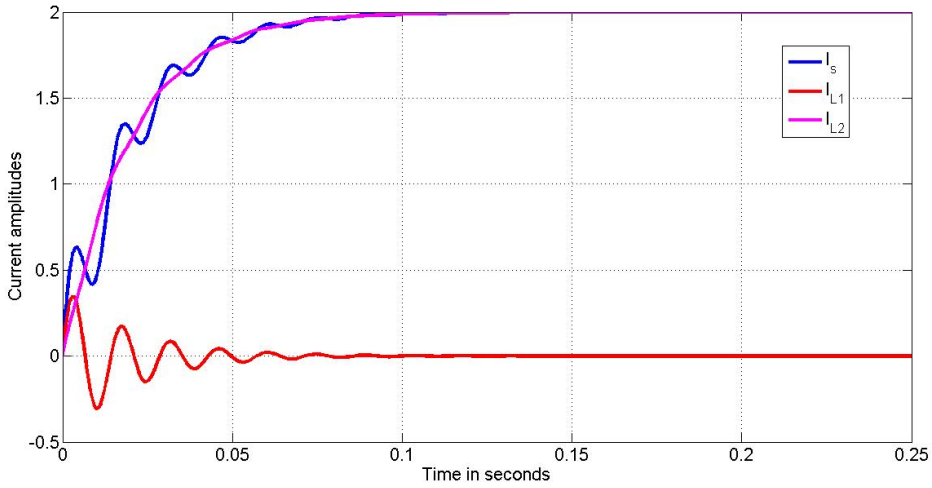


Figure 3.6: Simulated current waveforms using PSCAD/EMTDC, for the circuit of Fig. 3.3.

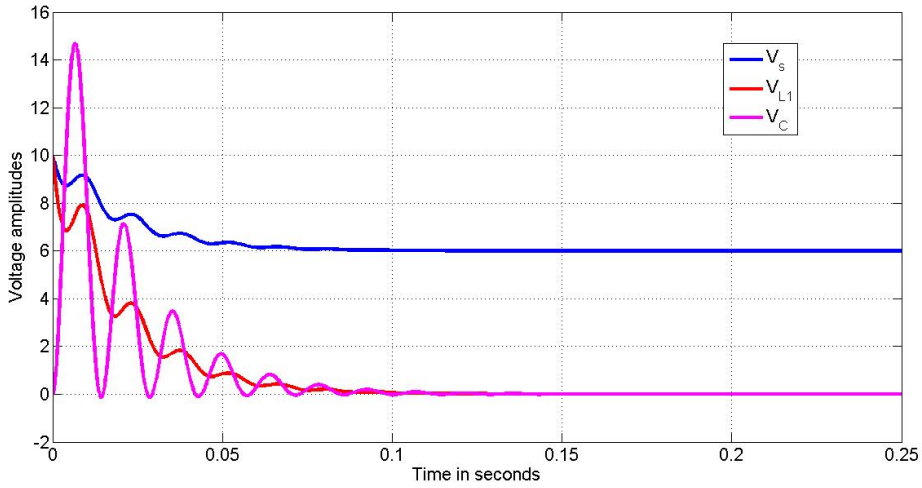


Figure 3.7: Simulated voltage waveforms for the circuit of Fig. 3.3, using GPU-computing.

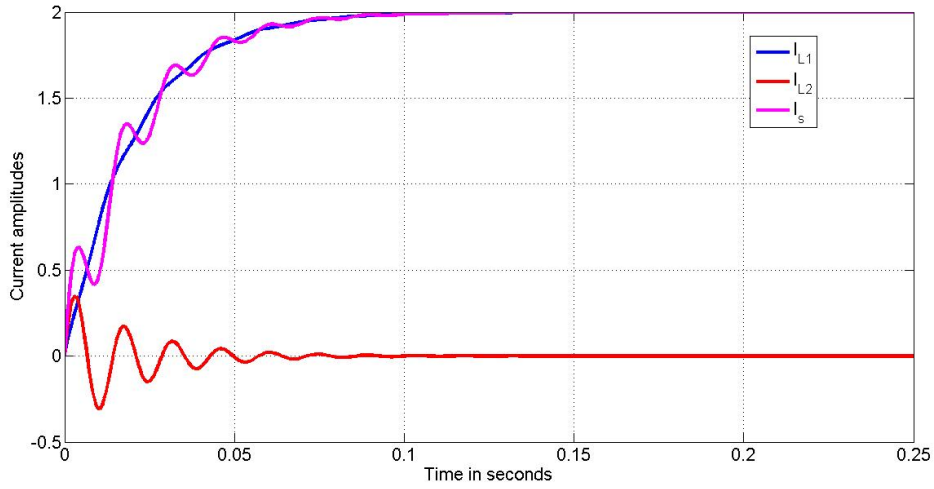


Figure 3.8: Simulated current waveforms using GPU-computing for the circuit of Fig. 3.3.

ous, the inductive branch voltage settles down to zero volts after its initial transients that last for a few miliseconds. After the initial transients, the capacitive branch is short circuited by the inductive branch L_1 (shown in Fig. 3.3); therefore, the capacitive branch voltage is also settled to zero volts after the transients. Also, Fig. 3.6 shows various currents for the circuit of Fig. 3.3, simulated using PSCAD/ EMTDC. Similarly in this case, the inductive branch current reaches its peak value after the transients (due to the reason mentioned above) and the capacitive branch current reaches zero after initial transients (as this branch is short circuited after the initial transients). Next this same circuit (Fig. 3.3) was simulated using CUDA-C programming technique in parallel on the GPU (here only the simulation results are presented, details on CUDA-C programming techniques for EMT simulation are presented in Chapter 4). Fig. 3.7 shows the voltages at various nodes of the circuit, simulated using GPU-computing and Fig. 3.8 shows various currents for the circuit simulated using GPU-computing. To compare GPU computing based simulation results of Figs 3.7 and 3.8, with the rersults obained using PSCDA/EMTDC based simulation (i.e. of Figs 3.5 and 3.6), simulated current waveforms were plotted (on

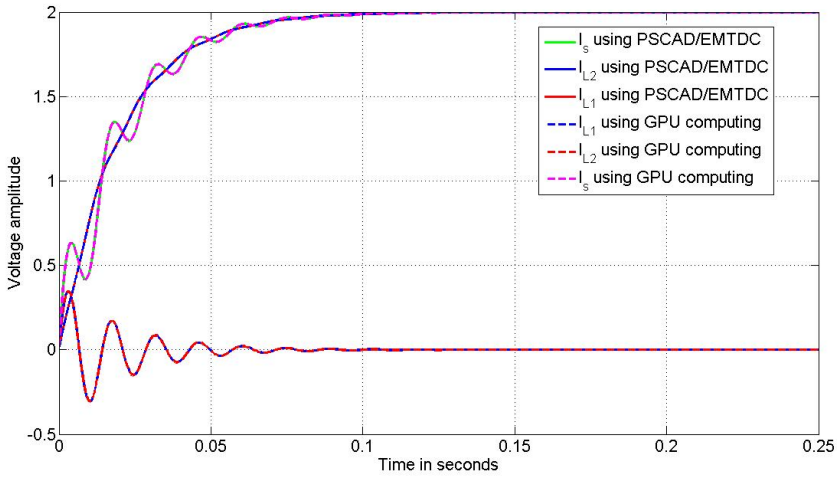


Figure 3.9: Comparision of simulated current waveforms using GPU-computing and PSCAD/EMTDC simulation for the circuit of Fig. 3.3.

top of each other) in Fig. 3.9. It is clear from Fig. 3.9 that the simulated current waveforms are identical. Hence, it is clear that GPU-based simulation generates exactly the same results as generated by a commercially available simulation tool (i.e. PSCAD/EMTDC [13]). In this case, the performance improvements from GPU-computing were not recorded (as the circuit had only three nodes). A comprehensive comparison of performances are presented in Chapter 5. In cases where networks are comparatively small in size, CPU-based simulation outperforms GPU computing as communication between the CPU and GPU is time consuming (i.e. transferring data between the CPU and the GPU creates a communication bottleneck). Details on adapting this conventional EMT simulation on the GPU will be presented in later chapters. Based on the above discussion a typical flow chart of EMT simulation is presented in Fig. B.7. This flowchart schematically explains the typical steps for sequential version of EMT simulations.

3.4 Chapter summary

This chapter briefly introduced the basic steps in conventional EMT simulation. A typical example was implemented using the basic steps. Simulation of this circuit was implemented on the CPU using conventional sequential computing and on the GPU using parallel computations (will be discussed in the next chapter). A schematic flowchart for EMT simulation (sequential) using the techniques introduced in this chapter was presented in Fig. B.7. Implementation of parallel techniques to adapt EMT simulations on the GPU and parallel modelling of various equipment on the GPU are introduced in the next chapter.

Chapter 4

EMT simulation using GPUs

4.1 Introduction

EMT simulation, as presented in the previous chapter, is a time domain simulation tool. Compared to other modeling approaches, such as transient stability analysis (TSA) tools, EMT simulation [1, 18, 19, 20, 21] models power system components in their greatest detail. As a result EMT simulation can simulate travelling wave phenomena in transmission lines, switching of power electronic devices, etc., which are normally not possible using TSA tools [1]. Due to the inherent complexity and computational intensity of EMT simulations, it was originally used for relatively small networks. However, due to the advances in digital computer technology, EMT simulation is now widely used to study large power systems (e.g. power networks having upto few hundred busses are usually considered as small networks) with fast acting dynamics. Simulation of large existing or planned networks take considerable amount of computer time on conventional EMT programs that uses serial computing. This is particularly true for power networks involving power electronic converters such as High Voltage DC (HVDC) transmission systems or Flexible AC Transmission Systems (FACTS) devices. These switching devices require small timesteps (the order of

$1 - 10\ \mu s$) and also require online (i.e. on the fly) updating and retriangularization of the admittance matrix [63].

It is well known that EMT simulation consists of many parallelizable tasks such as matrix-vector multiplication, transmission lines related computations, etc. [9, 22, 24]. Indeed the use of transmission lines as an interface between subsystems has been exploited in real-time digital simulators [11, 64]. Due to the delay in signal transmission over transmission lines, the networks on either side are isolated in any given instant of time and can be solved in parallel. As mentioned earlier, GPUs could be used to perform those parallel tasks on their multiple processing cores and could considerably speed up the EMT simulation. This chapter starts with a brief overview of the state of the art of EMT simulation techniques to accelerate computations, then it presents the details of adapting the EMT algorithm on the GPUs to obtain optimal speedup in the computation. This chapter also introduces the test cases used in this work.

4.2 Overview of earlier approaches in accelerating EMT simulation

With the introduction of digital computer based EMT simulation, there was a tremendous demand to increase the simulation speed and hence to increase the capability of EMT simulation to include large power networks [23]. Many attempts (as mentioned earlier) have been taken to accelerate EMT simulation by parallelizing the algorithm [25]. Most of those approaches implemented parallel EMT simulation using parallel processors, such as parallel implementation of the EMT algorithm [22], multi-processor based parallel EMT simulation [28, 29], etc. Introduction of multiple-core based desktop computers paved the way to speedup simulatins using various methods, such as multi-area Thevenin equivalents (MATE) algorithm [23]. Other

approaches such as time-domain transformation method [63], grid processing based implementations [30], etc. tried to accelerate EMT-simulation using networked multi-core computers (commonly known as pc-clusters) [19, 25, 26]. Supercomputers also have been used to accelerate EMT simulations [16, 27]. A brief overview of various directions in performing EMT simulation using parallel processing techniques are summarised in the report of the IEEE PES Task Force on Computer and Analytical Methods [24].

Early attempts at parallelization used the natural latency of transmission lines in decoupling the networks into subsystems that could be simulated in parallel. A time delay for the propagation of electrical signals on a long transmission line is imposed by the special theory of relativity that states that no signal can travel faster than the speed of light. This means that a 300km long transmission line will require at least 1ms of time to receive information sent from one side of the transmission line to the other side (assuming speed of light, $c = 3 \times 10^6$km/sec). Therefore, if the propagation delay for the signal through the transmission line is greater than one timestep, information from one side of the transmission line cannot influence the sub-system on the other side of the transmission line in a given timestep. Hence, the subsystems can be treated as isolated and solved in parallel. Most real-time simulators (such as RTDS [5, 11]) use this method to split large networks.

The method presented in [22] proposes parallel solution of differential equations using the trapezoidal integration technique to perform EMT simulation. In this approach unknown parameters of the assigned simulation are fitted into a vector and computations related to each element of this vector are simulteneously performed in different processors. This approach requires $\frac{T}{2}$ number of parallel processors, where T is the number of discrete time steps needed for the solution of the differential equations.

The major drawbacks of this approach were the requirement of the size of unknowns to be a power of 2 (for best performance) and the required number of dedicated parallel processor are also unrealistic for large systems with thousands of busses.

The Multi-area Thévenin equivalent (MATE) algorithm was introduced by J.R. Marti *et al.* [23, 64] to speed up EMT simulation. The MATE algorithm uses the concepts of Modified Nodal Analysis (MNA) [62] and DIAKOPTICS [65]. MNA is an extension of the commonly known nodal analysis technique with the capability of calculating some branch currents along with the calculation of various node voltages [62]. DIAKOPTICS [65] is a special node mapping technique to split large admittance matrices into smaller subsystems by using virtual tie-lines. This approach first solves individual subsystems simultaneously and then those solutions are combined (modifications are performed if needed) to generate the final solution of the network [28]. The MATE algorithm also uses parallel processing techniques (along with MNA and DIAKOPTICS, which divide large networks into smaller subsystems) that facilitates faster processing. In this approach large electrical networks are partitioned into smaller subsystems by tearing apart some of the branches using long transmission lines [64]. Hence, all of these subsystems are completely decoupled from the neighboring subsystems. These subsystems are solved independently by using the Thevénin equivalents for each link. Transmission lines are considered as links between these subsystems. Equivalent currents for various links are computed separately and are injected into the subsystems.

Recent approaches presented by the author of this thesis [33] and Zhou *et al* [36] also used similar approaches to exploit the parallelism inside the EMT-simulation and implemented them using GPU. The approach presented by Zhou *et al.* [36] uses a special Node Mapping Structure (NMS), which re-formats the original admittance

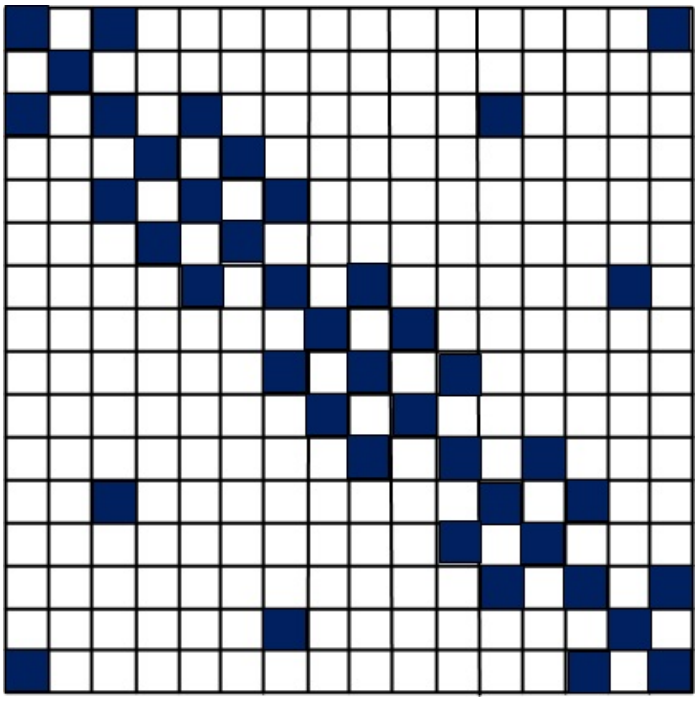


Figure 4.1: Schematic view of the admittance matrix for a power system.

matrix into a perfect block diagonal form. This approach is similar to the DIAKOPTICS [65] based approach, which is used to solve large systems by tearing them into smaller subsystems. Zhou *et al.* [36] divided the whole system into smaller subsystems decoupled by virtual transmission lines. All of these sub-blocks were treated as a dense system (i.e. all the busses of those smaller subsystems were physically connected to each other) and a normal inversion method (without sparsity) was implemented to calculate the unknown node voltages. To explain this approach a schematic of the admittance matrix is considered and is shown in Fig. 4.1. This matrix has diagonal and non-diagonal elements as is usually the case in a real power system. A view of the original admittance matrix of the IEEE 39 bus system is shown in Fig. 4.2, where non-zero elements are white spots and dark spots are representing zeroes. As seen in Fig. 4.2 the admittance matrix contains diagonal as well as non diagonal elements (non-zero). It should also be noted that the admittance matrix (i.e. in case of nodal analysis based EMT simulation) is symmetric. In the very first step of the approach

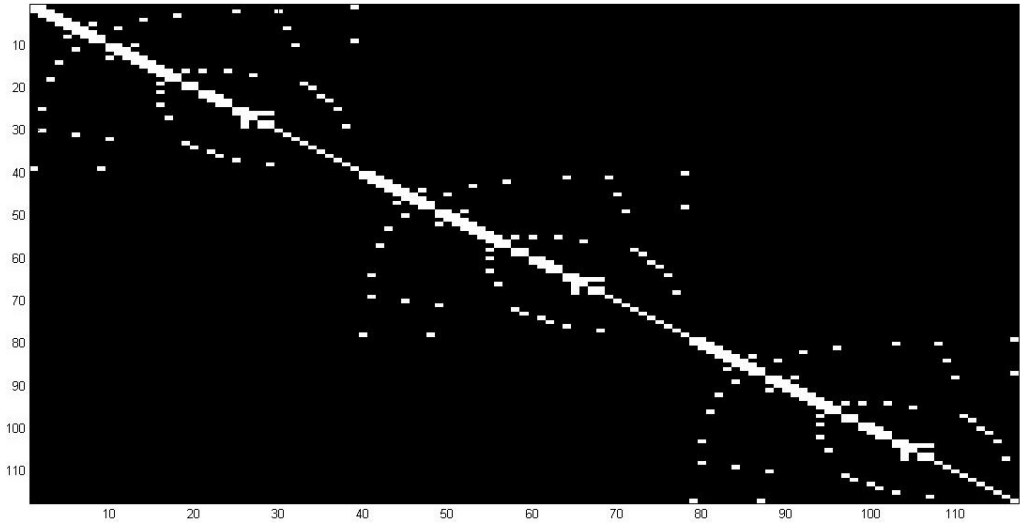


Figure 4.2: View of the admittance matrix of an IEEE 39 Bus system (white spots represents nonzero elements and dark spots represents zero elements in the admittance matrix).

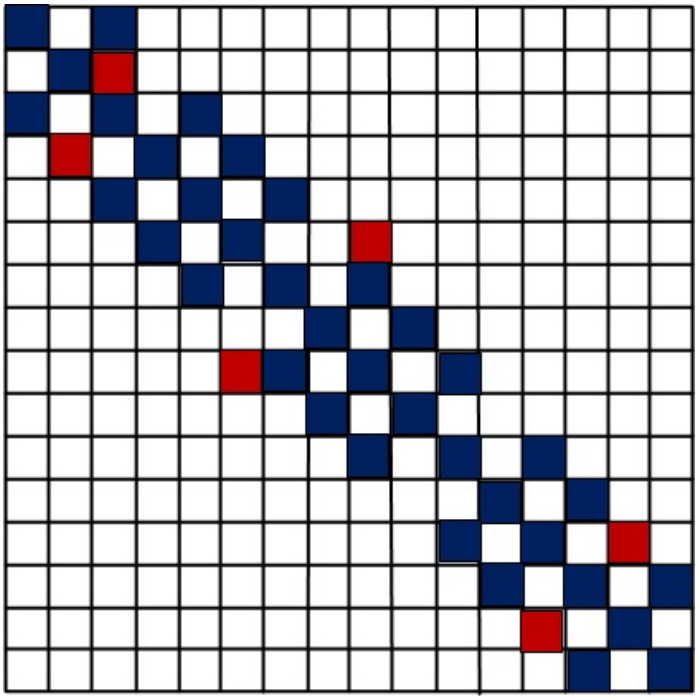


Figure 4.3: Schematic of re-arranging the non-diagonal elements of the matrix to form a block-diagonal matrix.

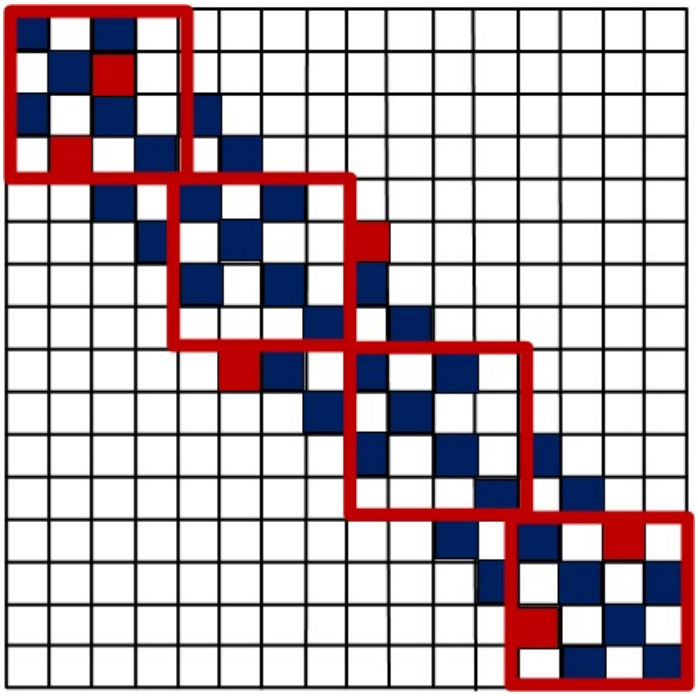


Figure 4.4: Schematic of dividing the system matrix into smaller sub-blocks as proposed by Zhou *et al.* [36].

by Zhou *et al.* [36], the admittance matrix was converted into a block diagonal matrix using their special NMS algorithm. Schematically this resultant admittance matrix is shown in Fig. 4.3, where all the elements are aligned accordingly to represent a perfect block diagonal matrix. In this case the non-diagonal elements (non-zero) of the matrix of Fig. 4.1 are regrouped to make the matrix a perfect block-diagonal. In the final stage (of Zhou *et al.*'s approach) this matrix was divided into smaller subblocks along the diagonal elements, as shown schematically in Fig. 4.4. In this figure, (for explanation purposes) smaller sub-blocks were given an arbitrary size of 4×4 (though the approach by Zhou *et al.* [36] may have used a different block-size). As seen in Fig. 4.4 the smaller matrices also contains some zero elements (i.e. resulting in a moderately sparse matrix). Zhou *et al.* [36], however, treated these smaller matrices as dense and implemented a normal inversion method for these smaller subsystems. It should also be noted that the elements outside the boundaries of individual

subblocks (as shown by smaller diagonal blocks in Fig. 4.4) were treated as virtual transmission line links. This approach has shown some acceleration in the EMT simulation (using GPUs), but it did not capture the full capabilities of the GPUs to boost the speed of EMT simulation. For example, Zhou *et al.*'s approach introduced many additional virtual transmission lines in the network. Computations related to these transmission lines have less parallelism then other parallel computaions such as matrix-vector multiplications. Additionally, transmission line-related computations introduce communication-related burden, additional memory accesses, etc., which make the computations slower on the GPU. Therefore, introducing too many transmission lines into the system by reducing the inherent parallelism inside the EMT algorithm negatively affects the performance gain (details on this will be presented in Chapter 5). In general GPUs are for massively parallel computations such as the case of matrix-vector multiplications, where less amount of memory accesses and inter-processor communications are preferred.

The approach proposed in this thesis organizes the parallelism in a different way that takes into account the equitable distribution of tasks for massive threads on the GPU. This proposed approach does not require the time-consuming additional algorithms (as used by Zhou *et al.* [36]) to subdivide the original admittance matrix into smaller subsystems (as explained using Figs. 4.1- 4.4). Instead the original test system (i.e. IEEE 39 Bus system) was used as the basic building block (will be elaborated later) and used the highly parallel GPU to perform the computations related to this matrix in parallel, which reduces the communication bottleneck considerably (as will be shown later). The approach results in fewer transmission lines in the systems in an attempt to minimize inter-processor communication bottlenecks to get more significant speedup in the simulation.

4.3 Parallel implementation of EMT simulation for GPU-computing

EMT simulation as presented in Chapter 3 implements the algorithm in a sequential manner. The first step of this sequential EMT simulation is to replace all the components of the power system with their Norton equivalents. These subsititutions of equivalent models make the network purely DC (resistive), which contains resistances and current sources only. The admittance matrix and the injected current vector (with proper initial conditions) for the whole network is formed. Then in the simulation loop (i.e. the code that updates the unknown system variables in each time-step), all the equivalent current sources and node voltages need to be updated in every time step, $\triangle t$. The inclusion of power electronic equipment or other switching components in the network may cause a topological change of the network (as mentioned earlier) during the simulation process. In that case online (i.e. on the fly) updating of the admittance matrix is also required, which may cause additional delay in the simulation (additional time delay in updating the admittance matrices was not considered in this thesis). This section introduces the inherent parallelism in the EMT-algorithm and discusses various techniques to perform EMT-simulation related computations on the GPU in parallel.

4.3.1 Parallelism in the matrix-vector multiplication

The mathematical representation of any electrical network (using nodal analysis [56] and Norton equivalents for various network components) has the general form as shown in (3.3) at any instant t [1, 4, 38]. Solving this equation for the vector of

unknown node voltages, V yields

$$[V] = [Y]^{-1} \times [J - I_H]$$

where $[Y]^{-1}$ is the inverse of the admittance matrix.

As seen in the above equation solution for the vector of unknown node voltages requires matrix vector multiplication. The size of the admittance matrix depends on the number of nodes in the network. For example in the case of a circuit with N nodes (e.g. N is chosen to be in the range of few hundred or even more), the admittance matrix will be an $N \times N$ matrix. Solution for the vector of unknown node voltages requires the multiplication of this potentially large matrix with a vector of size N. It is shown in the author's earlier works [33, 34, 35] that matrix-vector multiplication consumes upto 90% of the total simulation time. This matrix vector multiplication can be significantly speeded up by using a GPU.

Computation of the unknown node voltages involve matrix-vector multiplication where each row of the matrix is multiplied with the vector. Multiplication of each row with the vector involves multiplication of each element of the row with the corresponding element of the vector. Finally these element-wise multiplication results are added together to generate the appropriate element of the resultant vector. This conventional matrix-vector multiplication is shown schematically in Fig. 4.5. It should be noted that the multiplication process for one row of the matrix with the vector is completely independent from the multiplication of other rows. Therefore, multiplication of each row with the vector can be assigned to an independent thread on the GPU to perform matrix-vector multiplication in parallel. In this case, given sufficient processing cores, the total time for the matrix-vector multiplication would approxi-

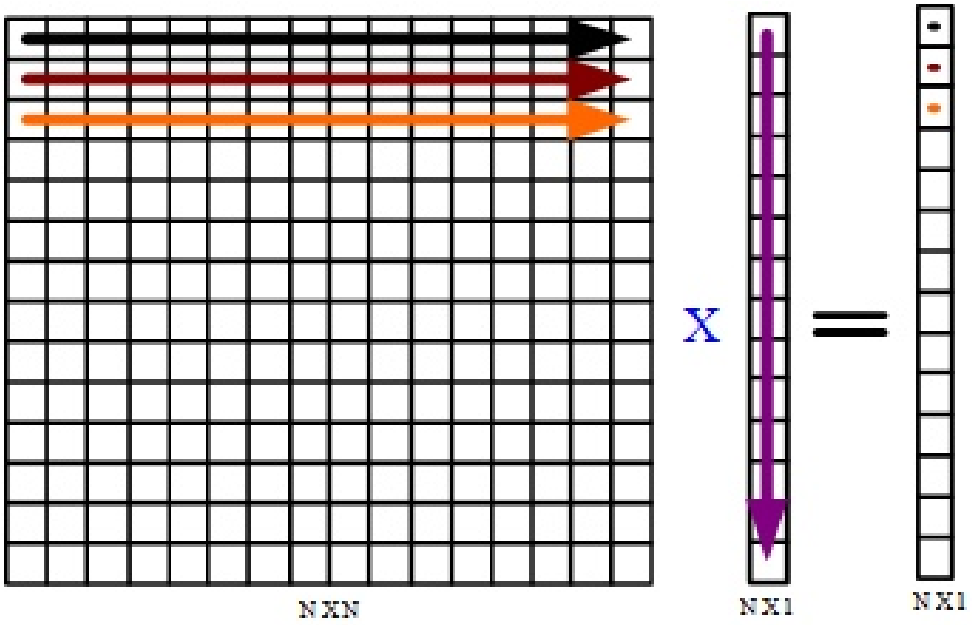


Figure 4.5: Schematic of conventional matrix-vector multiplication process.

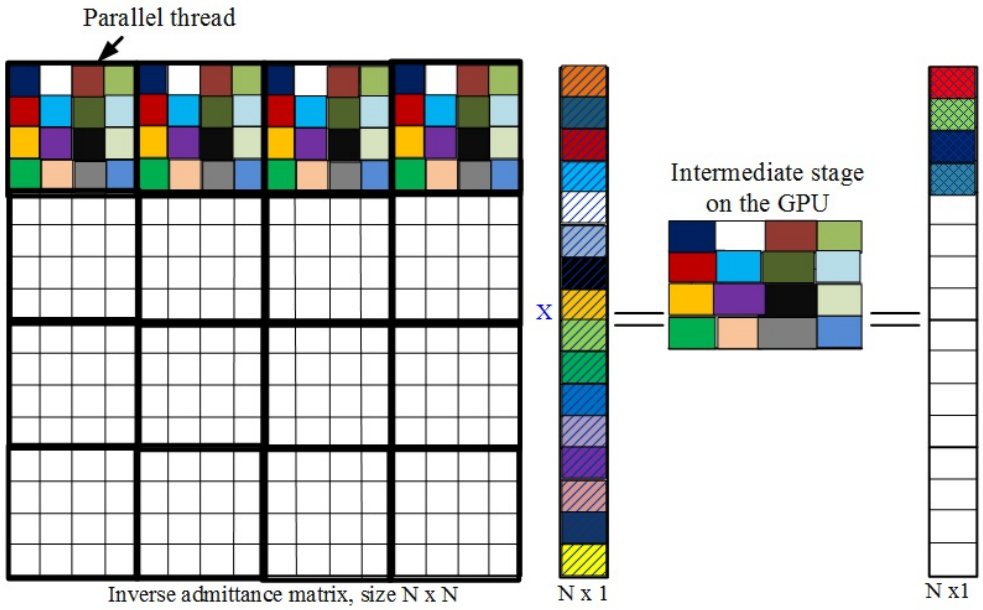


Figure 4.6: Schematic of matrix-vector multiplication by splitting the matrix into various blocks (suitable for GPU based implementation due to the availability of many processing elements).

mately be the time of performing multiplication of a single row with the vector (in this case ideal situation with no hardware limitations is assumed). This method of exploiting parallelism in matrix-vector multiplications is useful if the matrix size is relatively small (due to the limitations imposed by the hardware architectures of GPUs) [31, 37, 45]. Therefore, the maximum dimension of the matrix to perform matrix vector multiplication is actually limited. The size may vary depending on the particular version of the GPU. To overcome this limitation when the required number of threads per block exceeds the available hardware limit, the whole matrix and the vector must be divided into a number of small blocks. Each of these blocks corresponds to a certain number of threads (defined by the user/programmer at the beginning of the simulation, which can be any number between 1 and the block-size) to perform computations in parallel. This approach of matrix-vector multiplication was implemented in the author's earlier works [34, 35]. Note that the total number of floating point operations required to perform a matrix-vector multiplication between a matrix of size $M \times N$ and a vector of size N is $M \times (2 * N - 1)$. Therefore, if the matrix is divided along the row in 4 different blocks, essentially the total number of computations will be reduced to a fourth, i.e. $\frac{M}{4} \times (2 * N - 1)$. In this thesis the matrix was divided along both the x and y directions simulteneously, which results in a further reduction in the required number of computations and hence higher speed up (i.e. the blocks were created on both M and N variables in the above example). The approach to implement matrix-vector multiplication used in this work is shown schematically in Fig. 4.6. In this case parallel threads are deployed in both the x and y directions in every block (i.e. as shown in Fig. 4.6 each colored square represents a thread). These blocks are assigned to the GPU in parallel and all the computations are performed in parallel. Multiplication results of these blocks are temporarily stored in the shared memory of the GPU before producing the final results that will be transferred to the global memory of the GPU (this is shown as intermediate stage

in Fig. 4.6). This intermediate stage contribute to increase the number of parallel threads to be deployed during the invocation of the *kernel*-function (i.e. the function that contains instruction to perform computations in parallel on the GPU), which ensure faster processing. In this case, parallel threads are deployed in various blocks generated along the x and y axis direction of the matrix. For example for a 39 bus test system (3-phase) the admittance matrix dimension will be 117×117. If a block size of 3 is chosen for both the x-axis and y-axis dilmensions, then total of 39 blocks will be deployed during the *kernel*-invocation, along the x and y axis respectively. Similarly parallel threads will be deployed for each block in parallel. These threads perform computations in parallel and stores their results on the shared memory in parallel, which significantly reduces the total computation times for the matrix-vector multiplications and minimises the memory access times from the GPU (detailed results on performance gains will be presented later). Additional details on performing this multiplication on the GPU are further explained in Fig. A.1. Implemention details of this matrix-vector multiplication on the GPU is explained using the Algorithm 1 in Appendix A. This method of performing matrix-vector multiplication can handle matrices of arbitrarily large dimensions. More details on matrix-vector multiplications in parallel may be found in [31, 37, 45].

4.3.2 Parallelism in history currents calculations

Power systems are usually modeled using lumped parameter equivalents such as resistive, inductive, and capacitive elements. Similar models exist in the case of distributed elements such as transmission lines and cables. A branch of a power system containing a resistive and an inductive element in series is shown in Fig. 4.7a. This RL branch can be replaced with a Norton equivalent consisting of a single equivalent resistance and an equivalent current source [2, 1]. Norton equivalent of the RL branch (Fig. 4.7a) is shown in Fig. 4.7b. Calculation of the history currents for the

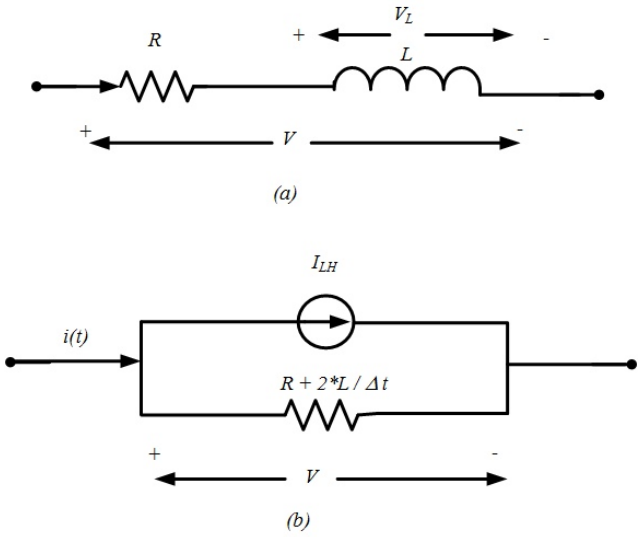


Figure 4.7: Schematic of a) a general RL-branch and b) Norton equivalent for the RL-branch following Trapezoidal rule based formulation.

resistive-inductive branch of Fig. 4.7a is shown in (4.1) below [1]:

$$I_{LH_{new}} = \frac{\frac{2\times L}{\triangle t}}{R + \frac{2\times L}{dt}} \times (1 - \frac{\triangle t \times R}{2 \times L}) \times (I_{LH_{old}} + \frac{V(t)}{R + \frac{2\times L}{\triangle t}}) + \frac{\triangle t}{2 \times L} \times V(t) \qquad (4.1)$$

where $I_{LH_{new}}$ is the new/updated history current for the Norton equivalent,

$I_{LH_{old}}$ is the history current from the previous time-step,

$V(t)$ is the value of the voltage across the RL-branch,

R is the resistance value and L is the value of the inductance.

Therefore, history current calculations (as seen from (4.1)) require the current of the branch from the previous time step and the present value of the voltage across the RL branch. The values of various node voltages are available in each iteration after performing the matrix vector multiplication. Therefore, by passing these voltage values and old history currents to a new *kernel*-function (as introduced earlier), all the history current computations can be performed in parallel on the GPU. Branches having capacitors require history current computations using (3.2). These computa-

tions are also suitable to be performed in parallel on the GPU in a similar manner. Details on implementing history currents computations on the GPU are explained in Algorithm 2 in Appendix A.

4.3.3 Simulation of synchronous generators

Electrical generators/motors are the energy conversion units in power systems. There are various types of electrical generators such as synchronous generators, induction generators, DC generators, etc. [17]. Among them synchronous generators are mostly used in various power generating stations. Therefore, modelling of this particular type of electrical generators are included in this thesis. It is common to model these generators as separate subsystems and interface them as Norton equivalent sources (i.e. a current source in parallel with an equivalent admittance [56]) to the original network [1, 20, 66]. In this thesis synchronous generators are modeled as a separate subsystem and interfaced with the main network as a Norton equivalent source [20, 17, 66]. The generator model implemented so far only includes a field winding on the d-axis, and armature windings on the $d\&q$-axis. Amortisseur windings are not included. The reader is reminded that the purpose of this research is to investigate and demonstrate the GPU-based implementations and the most detailed models have not been used in this thesis.

$$
\begin{aligned}
\tfrac{d\psi_d}{dt} &= v_d - i_d R_d - \omega\psi_q \\
\tfrac{d\psi_q}{dt} &= v_q - i_q R_q + \omega\psi_d \\
\tfrac{d\psi'_f}{dt} &= v'_f - i_f R'_f \\
i_q &= [L_{mq} + La]^{-1}\psi_q \\
\begin{bmatrix} i_d \\ i'_f \end{bmatrix} &= \begin{bmatrix} L_{md} + L_a & L_{md} \\ L_{md} & L_{md} + L_a \end{bmatrix}^{-1} \begin{bmatrix} \psi_d \\ \psi'_f \end{bmatrix} \\
J\tfrac{d\omega}{dt} &= T_{mech} - T_{elec} - D\omega
\end{aligned}
\tag{4.2}
$$

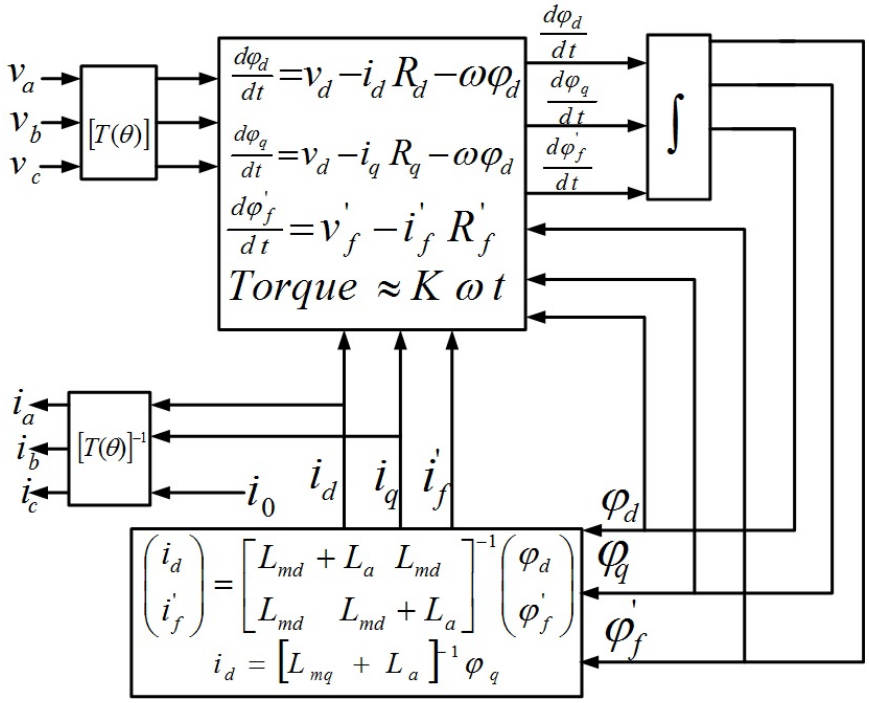


Figure 4.8: Schematic of the generator model as used in this work [1].

Basic mathematical equations used to model a synchronous generator are listed in (4.2) and can be found in [1, 20, 66]. In (4.2), ψ_d and ψ_q are the direct and quadrature axis flux linkages respectively. $V_{a,\ b,\ c}$ are the three phase AC input voltage to the generator terminals, $i_{a,\ b,\ c}$ are the three phase AC currents to be injected in the network respectively. $L_a,\ L_{md},\ L_{mq}$ are the various inductances of the generator's field winding. $i_d,\ i_q,\ i_0,\ i_f^{'}$ are the direct, quadrature, zero-axis and DC field current respectively. ω is the generator speed, $T_{mech},\ T_{elec}$ are mechanical and electrical torques respectively.

Fig. 4.8 shows the schematic diagram of the generator model used in this research with the necessary equations, input, and output parameters. In this case the generator models take their terminal voltages as input and calculate various currents to be injected back into the network using (4.2), also shown in Fig. 4.8. These currents are injected back into the network as current sources in parallel with an equivalent

resistance [1, 20, 66]. All the input terminal voltages from the network are first transformed into *dq0* domain. The *dq0* domain transformation is used to transform three-phase quantities (such as voltage and/or current) into a rotating reference frame attached to the rotor. Hence, in steady state 60Hz balanced currents and voltages on the rotor side become dc quantities in the *dq*-frame [17]. All the internal computations for the generator model are performed in this *dq0* domain, as shown in (4.2). Finally the output currents are transformed back into the phase domain and injected into the main network.

A large power system typically contains many synchronous generators. To implement generator related computations on the GPU, each generator could be assigned a block containing several threads to perform computations in parallel. In this thesis the synchronous generator related computations are performed using two different *kernel*-functions. One *kernel*-function rearranges the generator terminal voltages in a matrix form, which speeds up the memory access during the execution of the other *kernel*-funcion, which performs the main generator related computations. Fig. A.2, shows the schematic of the reformatting of various terminal voltages of the generator using the above *kernel*-function. Additional details may be found in page 110. Three threads in every block (for each *kernel*-function) were assigned and each block performed computations related to one three-phase generator. Details on implementing synchronous generators on the GPU is explained using the Algorithm 5 in Appendix A.

4.3.4 Injected current vector updating

Conventional EMT simulation requires updating the injected current vector in every time step. Injected currents may come from capacitive branches, inductive branches, generators and so on. Real power systems normally connect generating stations to the

load centers over vast geographic areas, which give rise to a very irregular (arbitrary) structure. Therefore, the injected current vector for the power system is also irregular. Hence, exploitation of parallelism in node interconnection (i.e. in the injected current vector) is limited; and this part of the EMT-simulation has the least amount of parallelism. In this thesis computations related to current vector updating were performed on the GPU. However, due to the lower amount of parallelism, they could also have been performed on the CPU. For every node of the network that connects to a current source (which may come from a host of elements, e.g., capacitors, inductors, transmission lines or generators, etc.) the total injected current vector is updated in every time step. In this case the injected currents for each node are added together and the final current vector is updated. As mentioned earlier, current vector updating for every subsystem is performed on the GPU using one single thread for each phase and each subsystem related current vector was assigned in one block.

4.3.5 Acceleration of other power system components using GPU

4.3.5.1 Transmission lines

Electrical power generating stations are often established in remote locations far from the consumption centers. Transmission lines are used to transfer bulk electrical energy from the generating stations to the demand centers. Accurate mathematical models are required to simulate these transmission lines and analyze their transient effects. Transmission lines have their accurate mathematical model in the time and frequency domain [1, 2, 67]. Those advanced transmission line and cable models take into account the frequency dependence of the line parameters. In this thesis only the simpler lossless Bergeron Model [68, 69] is implemented. In this work transmission lines are used to interconnect the basic test systems to create larger networks,

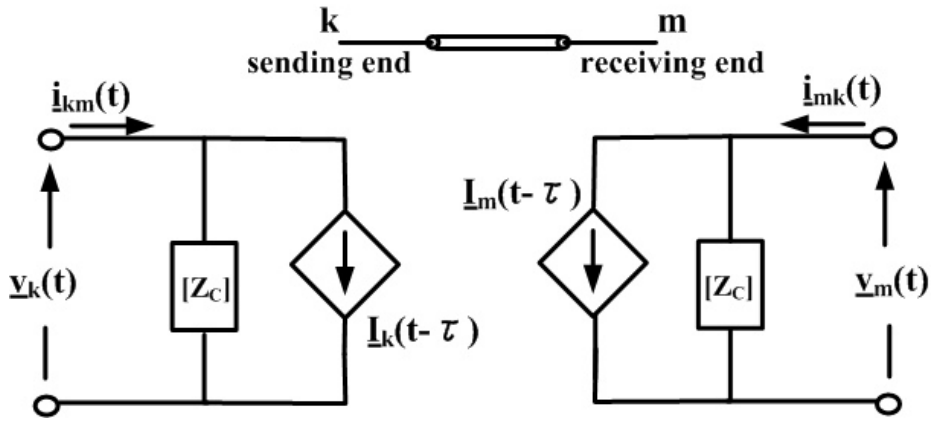


Figure 4.9: Modal equivalent of transmission line modeled using Bergeron's model [1].

which acts as tie-lines (other researchers also used this simplified model in the literature [36, 38, 23]). Therefore, GPU implementation of other models for transmission lines are left for future research. However, the parallelization approach would be identical for more advanced models of TL.

The Bergeron model is a time domain distributed parameter model [1, 68] and it was first developed by Louis Bergeron in the 1940s [70] and the technique was applied to EMT simulation by Prof. Hermann Dommel in the late 1960s [68, 69]. Fig. 4.9 shows the schematic of a single phase transmission line modeled using the Bergeron model. In this model, transmission line related computations are performed separately. Transmission lines parameters are transformed into single-phase equivalent lines using modal transformation matrices. All the computations related to transmission lines are performed in the modal domain. Equivalent transmission lines are interfaced into the main network as current sources in parallel with an equivalent Norton resistive network. The Norton network and the history current sources are calculated by applying inverse modal transformation to the corresponding modal quantities [2, 21]. The history currents are due to the remote end. Hence, they are delayed by the travel time of the transmisssion line. The history current computa-

tions for transmission lines in the modal domain use the following equations (for a single mode):

$$I^{'}_{jkm}(t-\tau) = -\frac{1}{Z^{'}_{j}}\ v^{'}_{jm}(t-\tau) - i^{'}_{jmk}(t-\tau)$$
$$I^{'}_{jmk}(t-\tau) = -\frac{1}{Z^{'}_{j}}\ v^{'}_{jk}(t-\tau) - i^{'}_{jkm}(t-\tau) \tag{4.3}$$

where, $Z^{'}_{j} = \sqrt{\frac{L_j}{C_j}}$ is the characteristic impedance of the Transmission Lines (TL) in the modal domain, $\tau = d/c$ is the travel time delay introduced by the TL, where d is the length of the line and c is the velocity of propagation of electromagnetic waves in free space ($\simeq$ 300 km/s) and equals to $\sqrt{L_j C_j}$ (L_j, C_j are the line inductances and capacitances per unit length per phase). $i^{'}_{jkm}$ and $i^{'}_{jmk}$ are the modal domain current at the sending end (k) and receiving end (m), for mode j. Also, $v^{'}_{jm}$ and $v^{'}_{jk}$ is the modal voltage at the receiving end (m) and sending end (k) of the transmission line for mode, j.

Transmission lines in the network introduce parallelism in two ways. Firstly, as seen from the above equations, individual phase related computations are separated using the modal transformation matrices and are thus suitable to perform computations in parallel. Secondly, as information does not propagate faster than the speed of light, if the line travel-time is longer than one time-step, effectively the two sides of the transmission lines are computationally decoupled and can be executed in parallel on the GPU. This latter form of parallelism is widely used in various real-time EMT simulations tools such as RTDS [5, 11]. In this research work, both of these parallelization techniques are implemented to obtain the optimal speed up in the simulation process. Transmission line related data are aligned into a matrix (similar to those described earlier for synchronous generators) and shared memory is used in the calculations to ensure faster access (hence faster computations) from the GPU. It

should also be noted that computations related to the sending and the receiving ends of the transmission lines were performed in parallel (using separate *kernel*-functions), which ensured more parallelism in the simulation. Details on implementing transmission lines related computations on the GPU are explained using the Algorithm 3 in Appendix A.

4.3.5.2 Power-electronic subsystems

Power systems are designed to continuously supply power to ever-changing load profiles throughout the day. Power systems usually consist of passive components. Hence precise control of the right amount of power to meet the demand accurately is critical. Therefore, power electronic devices are commonly used to control the power flow and power quality in the networks. These days many power electronic components are used in various power system devices such as Flexible AC Transmission Systems (FACTS) devices, High Voltage DC (HVDC) transmission systems, etc. Power electronic devices are used to precisely control the amount of current flow through various power system equipment. In this research a simple power electronic subsystem, namely a full bridge power converter, was implemented to demonstrate the capability of the GPU to simulate the behaviour of power electronic devices. This basic system demonstrates the methodologies involved in modelling power electronic subsystems. Implementation of an advanced HVDC system will require the implementation of advanced control/switching sequences on top of this basic subsystem, which will not significantly change the computation methodologies and hence the parallelism will not be unduly affected. The main focus of this research work was the development of the simulation methods to accelerate EMT simulation and the simple diode bridge is used to demonstrate the basic switching. In a commercial implementation, obviously, a full suite of power electronic equipment models will be available.

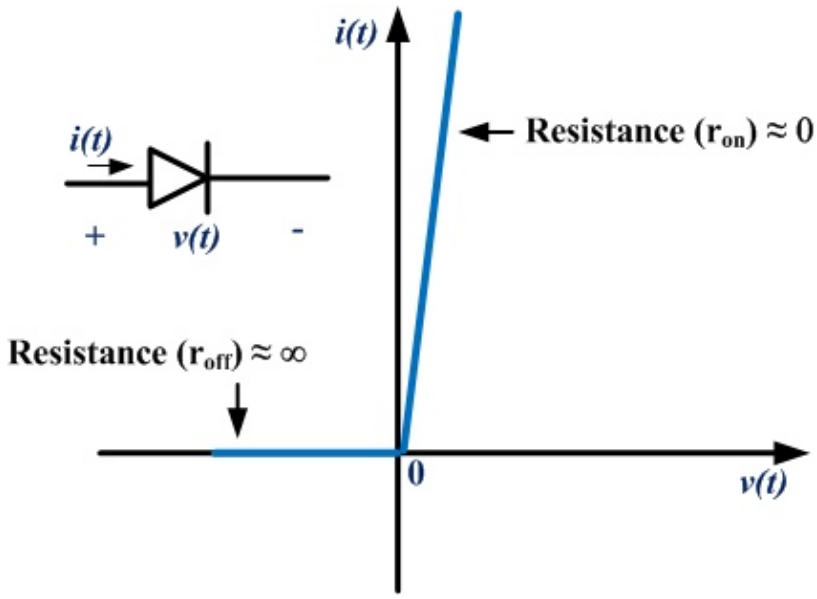


Figure 4.10: Schematic characteristic of a diode as used in this thesis for the diodes used in power electronic subsystem shown in Fig. 4.11. Showing the two resistive 'ON' and 'OFF' states.

The diode characteristic was approximated using a near-ideal switch characterisitic shown in Fig. 4.10, where the switch is in the ON-state (i.e. conducting) whenever there is a positive voltage across the diode and is in the OFF-state (i.e. non-conducting) whenever the voltage accross the diode is negative. The diode is turned on (i.e. conducting state and its resistance set to a very low value) when it is forward biased, and reverts back to the off state when the current goes negative (i.e. reverse biased with a very high resistance value). Additional logic (such as tracing the current through the diodes) determines the transitions between conducting and non-conducting states of various switches (diodes) in the network of Fig. 4.11. More details on identifiying the transition instants for power electronic switches can be found in [1, 56].

Fig. 4.11 shows the schematic of the power electronic subsystem (the diode bridge). Input to this system is a three-phase AC voltage and the output is a DC voltage. In

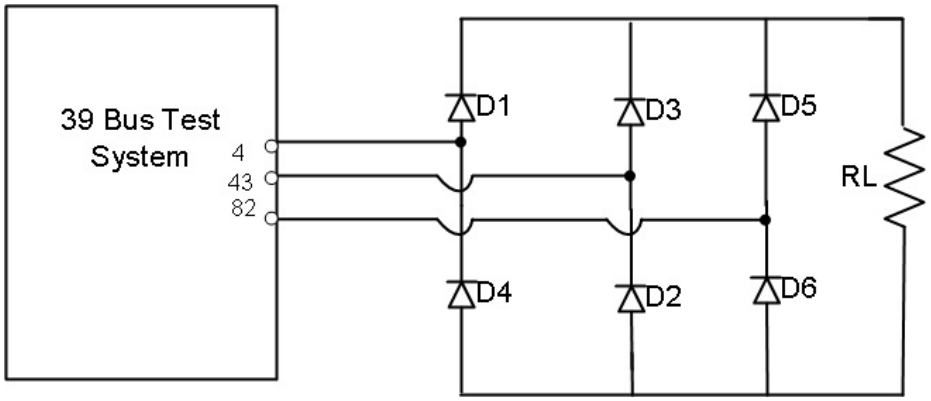


Figure 4.11: Schematic interconnection of the IEEE 39 Bus system interconnected with a full-bridge diode rectifier.

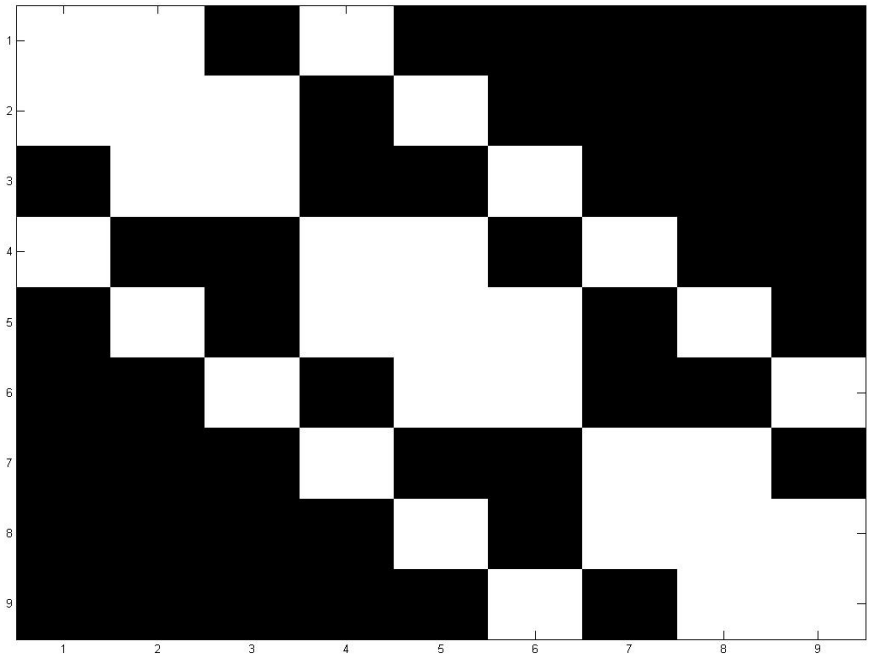
this network (i.e. the circuit of Fig. 4.11) the switches (diodes) turn ON (i.e. conducting state) and OFF (i.e. non-conducting state) multiple times within a cycle, depending on the applied AC voltage at the input terminals of the bridge. The swithches of Fig. 4.11 were modeled as resistors with binary switching states (very large conductance for the ON-state and very small (ideally zero siemens) conductance for the OFF state) [5, 58].

As mentioned earlier, implementation of switching introduces additional admittance matrices in the simulation process due to changes in network admittances. In this thesis, inverses corresponding to these admittance matrices were precomputed on the CPU and stored on the GPU memory and were inserted into the simulation whenever needed. This approach is viable even if a Pulse Width Modulation (PWM) type of inverter is used with multiple switchings in each cycle, because, matrices for subsystems with converters can also be pre-inverted and stored on the GPU memory. It may be possible that some configurations are missed in the pre-computation process. In that case, the CPU can be used to quickly compute the required inverses. Once computed it could be properly tagged and used later in the simulation if required again. Note that this is not a real-time simulation, so occassional slowing down caused by this

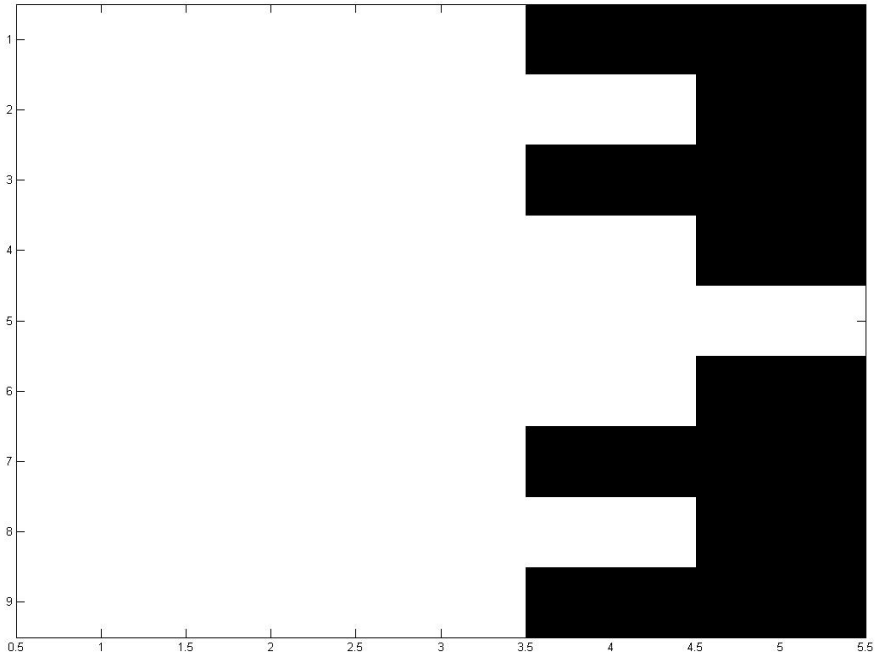
process is acceptible. Additionally, any subsystem with switchings may be given a smaller size, in order to minimize the storage and computational burden.

4.4 Sparsity implementations

In general power systems are loosely inter-connected (i.e. individual buses are connected only to few of the neighbouring buses) and the resultant admittance matrices are mostly sparse in nature (i.e. most of the elements are zero as shown earlier in Fig. 4.2, for the example of a 39 bus system). In most cases the admittance matrix could be approximated as block-diagonal, having most of the elements along the diagonal only. As mentioned earlier, EMT simulation involves matrix-vector multiplications. In [33, 34, 35], the author of this thesis has shown that the matrix-vector multiplication on the CPU (i.e. sequential implementation) may consumes upto 90% of the total simulation time. In case of dense matrix implementations, all the elements of the admittance matrix (i.e. zeros and non-zeros) are multiplied with the vector to determine the new state of the network. Therefore, multiplication operations involving these zeros (while simulating the network) reduce the available computing power and hence the performance. Various techniques have been taken to ignore multiplications involving zeros, such as the compensation technique [71]. In the literature, there are various approaches to implement sparsity on the CPU [72, 73], but require additional effort to implement parallelism for special hardware such as GPU. Hence in this thesis, a lookup table based approach has been applied on the inverse-admittance matrix to ignore multiplications involving those unwanted zeros. In this approach, a table containing the addresses (i.e. subscripts of the matrix) of the nonzero elements (as shown in Fig. 4.12, white spots represents nonzero elements) is formed. Fig. 4.12b shows the schematic of the table containing the addresses of the non-zero elements. During the simulation process, an instruction reads the address of



(a) Schematic of the sparse matrix before reformatting (dark spots represents zero elements)



(b) Schematic of the sparse matrix after reformatting

Figure 4.12: Schematic of inverse admittance matrix before and after reformatting according to the sparsity technique used in this thesis (nonzero elements are white; zeros are black).

the non-zero elements from the table. Then another instruction read those elements from the inverse admittance matrix. Finally multiplication with the corresponding elements of the vector is performed. In general, power systems are loosely connected and the number of zeros in every row of the admittance matrix is high. Applying this algorithm saves significant time as the instructions do not need to perform the expensive multiplications involving floating points, which resulted in improved performance in the simulation. This approach was applied on all the test cases (i.e. less and high granular test cases). Details on performance improvements in the simulation process using this approach are presented in the next chapter.

Through testing on several example cases, it was determined that for a typical power system, the inverse-admittance matrix is highly sparse in nature. For example, in case of a 39 bus test system, the admittance matrix contains 13296 number of zeros before inversion and 9126 number of zeros after inversion (i.e. significant amount of zeros). Due to numerical issues in most cases the inverse admittance matrix is full of nonzero elements with individual values in the range of $[0,\ 10^{-10}]$, which are practically zero. It was observed by simulating various test cases that setting those elements to zero does not noticeably affect the simulation accuracy. Therefore, the algorithm used a threshold value to set such elements of the inverse admittance matrix to zero. This technique of implementing sparsity is illustrated using Fig. 4.12. Fig. 4.12a shows the schematic of the inverse admittance matrix (after using that threshold). Finally, this approach was also parallelized using the similar approach as used in case of matrix-vector multiplications.

4.5 Test cases used for GPU-based EMT simulation

GPU-based EMT simulation is intended to speed up computations for large power networks. Therefore, large test cases are required to evaluate the performance improvements of the proposed GPU-based EMT simulations. The focus of this thesis was to demonstrate that the GPU-based EMT simulation is computationally faster than traditional CPU based EMT simulation with increasing network size. Instead of using an existing large real system, test systems were constructed using a pre-defined structure that could be easily scaled indefinitely. This approach has also been used by other researchers in developing parallel simulation tools for power systems [23, 36, 50]. The development of the GPU-based simulation tool involves numerous effort in parallel implementation of the algorithms, parallelization of various component models, coding, testing of the simulation results, etc. Additionally, implementation of a particular realistic power network using the described GPU-based simulation tools will use the same parallelization techniques for various components as discussed above. Hence the focus was not given in implementing an existing real power system (although it was demonstrated in Chapter 3 that GPU-based simulation produces the same results as produced by commercial tools, such as PSCAD/EMTDC). As explained earlier, transmission lines are used as a link between subsystems to create large networks. Although inclusion of transmission lines in the network introduces exploitable parallelism, it also introduces additional computations related to the calculation of the transmission line variables themselves and also increases the inter-processor communication. Computations related to transmission lines require memory access calls for the various sending and receiving ends and communications between the sending and receiving ends, etc. It is a commonly used technique in power system to divide a large network into smaller subsystems using transmission lines. Therefore, dividing

a large network into smaller subsystems will require additional transmission lines. On the other hand additional transmission lines will introduce the above mentioned delay in the computations. It is to remind that transmission lines in this thesis are implemented using lossless Bergeron model. Implementation of other advanced model for transmission lines, such as frequency dependent transmission line models will introduce additional computations for themselves. Thus, too many transmission lines in the network can reduce the overall parallelism and adversely affect the simulation speed. The ratio of the total number of transmission lines used to interconnect the neighbouring subsystems to the total number of buses in the large system is defined as a measure of granularity. Therefore, higher value of granularity means, more transmission lines in the network and smaller subsystem size. In this thesis two sets of test cases with different granularity were used and described below.

4.5.1 Test cases based on IEEE 39 Bus system (low granularity)

In this case, test systems of increasing size were created by interconnecting a basic building block containing power electronic subsystem, as shown in Fig. 4.13. This subsystem demonstrates switching methodology on the GPU with a basic power electronic subsystem. Most advanced HVDC devices will require the implementation of additional control sequences for the algorithm, which will not affect the parallelism. This building block system of Fig. 4.13 was constructed by interconnecting an IEEE 39 bus system with the power electronic subsystem introduced in Fig. 4.11. It should be noted that there is no distributed parameter transmission line inside this system. The links between various buses of this system are implemented using π-equivalents. This system provides scalability to create larger test cases by interconnecting these building blocks using distributed parameter transmission lines. Additionally, the inclusion of the basic power electronic diode bridge rectifier enabled the capability to

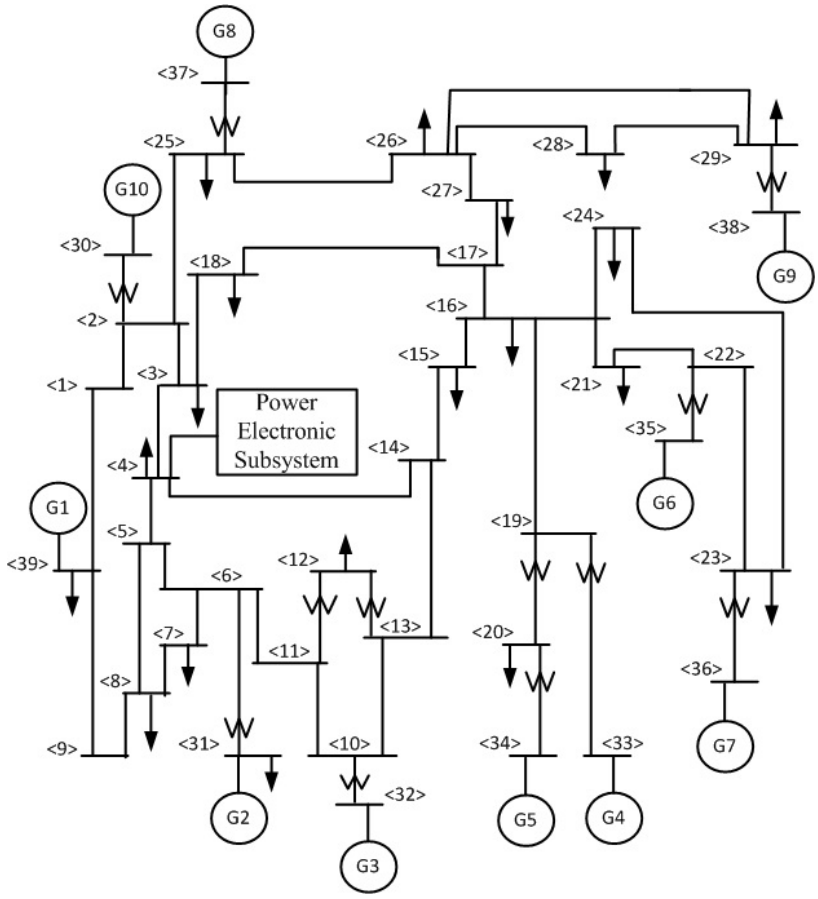


Figure 4.13: Schematic of the IEEE 39 Bus system connected with the rectifier, referred to as *Building Block* 1 system.

check the performance of the GPU-based simulation when power electronic equipment is present in the network. It should be noted that test systems constructed using this basic block have an asymptotic value of the granularity equal to 0.154 as the network size increases towards infinity (i.e. $\lim_{N\to\infty}$granularity(N) = 0.154, where N is the size of the network).

This approach (i.e. creating large network by interconnecting the basic block) closely resembles large power networks having many devices interacting simultaneously in an existing power network. It should be noted that earlier approaches did not implement power electronic subsystems on the GPU.

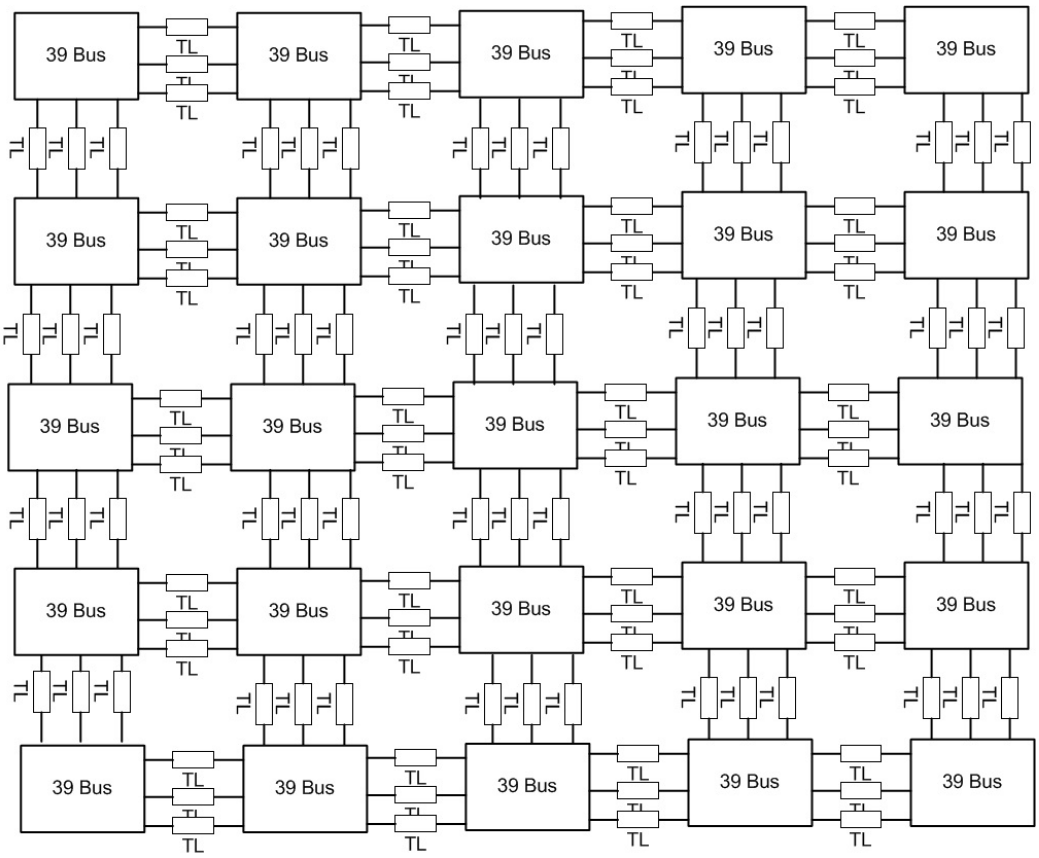


Figure 4.14: Schematic of a 975 Bus less granular test system created by interconnecting *building block* 1 system of Fig. 4.13.

Henceforth the basic system of Fig. 4.13 will be called *building block* 1. The *building block* 1 system has ten three-phase generators, twelve three-phase transformers and one power electronic subsystem. Larger test cases were created by interconnecting this *building block* 1 system using transmission lines. As an example, Fig. 4.14 shows the schematic of a test system having 975 buses, which was created by interconnecting 25 instances of *building block* 1. The system of Fig. 4.14 has 250 three-phase generators, 25 power-electronic subsystems, 300 three-phase transformers and 120 three-phase transmission lines. It should be noted that these test systems do not represent any real system, but are used to (artifacts) easily construct arbitrarily large systems to evaluate the capability of the proposed GPU based parallel EMT simulation.

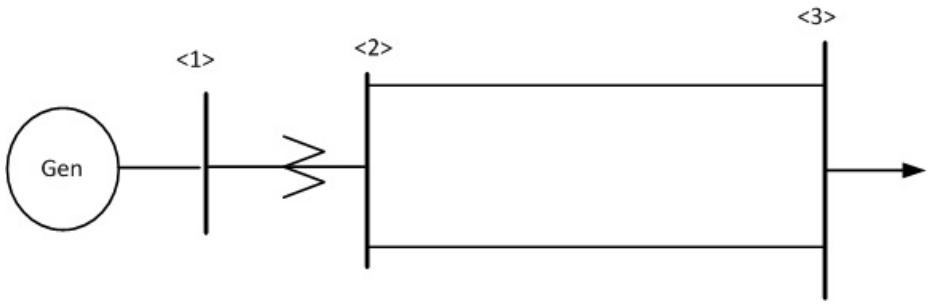


Figure 4.15: The 3 bus system used in this work as *building block* 2 (links between various buses are implemented using Π-equivalents).

4.5.2 Test cases created by 3 bus system, (high granularity)

To investigate the performance with increased interconnection density (transmission lines), additional test cases with a different *building block* system were implemented. The interconnection density (as explained above) is determined by the ratio of the number of transmission lines used to interconnect various *building blocks* to the total number of busses present in the network. A higher number of transmission lines require increased computation burden which involves higher amount of communication between the interconnecting subsystems, memory access calls and so on. This ultimately leads to reduced performance gain in the simulation.

The effect of the increased number of transmission lines on the total simulation times was explored using this more granular test system (shown in Fig. 4.15). The asymptotic value of granularity using this system as a building block is 1.33 as the network size increases towards infinity. In this case a basic building block of only 3 buses (as opposed to the 39 bus *building block* 1 introduced earlier) was used. It consists of one generator and one three-phase transformer (power electronics were not included in this case). The 3-bus system is shown in Fig. 4.15. Henceforth this 3-bus system will be called *building block* 2. There is no distributed parameter transmission line with in the *building block* 2 systems and the links between various buses of *building block*

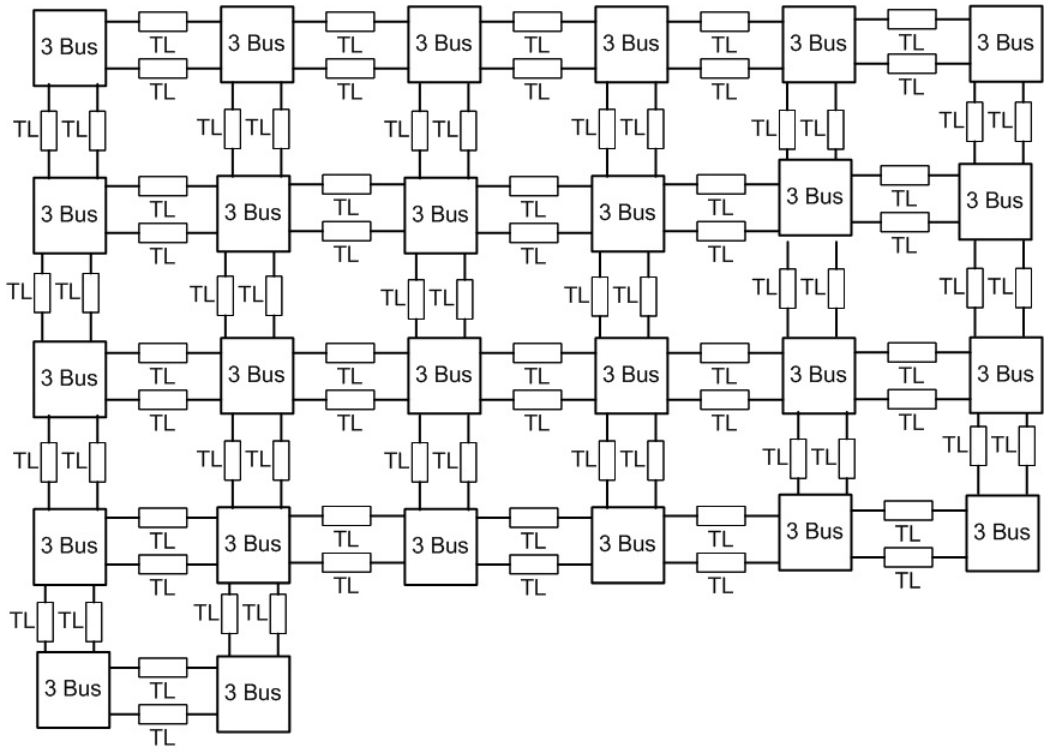


Figure 4.16: Schematic of a 78 bus highly granular test system created by interconnecting *building block* 2 system of Fig. 4.15.

2 system are implemented using π-equivalents. Larger test cases were implemented by interconnecting this 3-bus system using lossless transmission lines. As an example, Fig. 4.16 shows the schematic of a 78-bus test system created by interconnecting 26-instances of *building block* 2 using lossless transmission lines. With 26 instances of parallelizable subsystems, it is more granular than a corresponding 78-bus system constructed using the *building block* 1, which would only have 2 parallelizable subsystems (i.e. for same number of nodes). However, it has 80 interconnecting 3-phase transmission lines between the subsystems, whereas the corresponding system built with the *building block* 1, would only have 3 three-phase transmission lines. Simulation results (presented later) show that the larger number of T-lines in the test cases have a negative impact on the total simulation times. More detailed schematics of various test cases constructed using *building block* 1 and *building block* 2 systems are

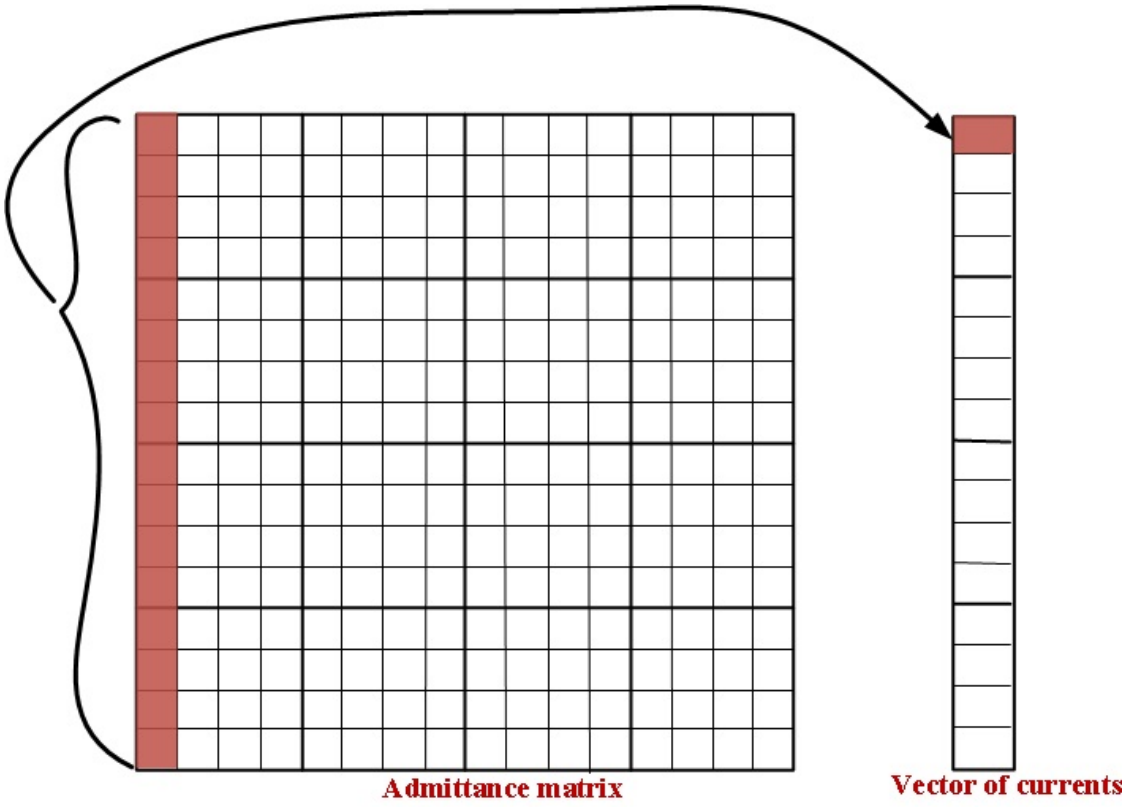


Figure 4.17: Schematic of accessing a particular memory location simultaneously by many parallel threads during matrix-vector multiplications on the GPU.

presented in Appendix B.

4.6 Hardware imposed barrier for GPU-based EMT simulations

GPUs are specially designed hardware to accelerate massively parallel algorithms, such as rendering in computer graphics. Most of the general purpose computations are a mixture of serial as well as parallel computations. Therefore, general purpose computations must be parallelized to apply them on GPUs to achieve the desired performance improvements. Some of the barriers in getting optimal performance gain using GPU computing were presented earlier in section 2.4. This section presents

another barrier in GPU-based EMT-simulation, which is specific to matrix-vector multiplications and may result in less performance gain in the parallelised EMT simulation.

In the case of performing the matrix-vector multiplication on the GPU a critical situation in accessing the required memories on the GPU occurs. Fig. 4.17 shows the situation of accessing the very first memory location of the vector of currents by the very first element of every row of the matrix simultaneously, which is required repeatedly in GPU-based parallel EMT simulation. It was explained earlier that multiplication of each row of the matrix with the vector is implemented in parallel. As seen from Fig. 4.17, the very first element of each row has to be multiplied with the very first element of the current vector. A similar situation exists for every other element of the current vector as well. As the multiplication job is performed in parallel, all the threads corresponding to those elements will try to access (as well as write after computations) the memory location simultaneously in parallel. As the size of the matrix becomes large the number of parallel threads trying to access that particular memory location would be large and they will have to wait until they are able to read the data (GPU allow concurrent reading on the shared memory [37]). This waiting for reading the data places an obvious constraint on the expected performance gain of the simulations. To overcome this barrier, in this thesis the matrix was divided into many subblocks and an attempt has been taken to use the shared memory (as opposed to global memory) as much as possible as shared memory supports concurrent reading of data [37]. At the same time, use of transmission lines to divide large networks into smaller subsystems (i.e. smaller size of the admittance matrix) reduced the amount of concurrent data access requirement. Note however that (as mentioned earlier) having too many transmission lines also slows the overall computational performance. This subsystem based EMT simulation also reduced the

size of the admittance matrix and hence it contributed to reduce the total number of parallel access of the memory location. However further investigation in this topic is still needed to accelerate the memory accesses, which may be a good direction of this thesis in the future.

4.7 Chapter summary

The algorithmic steps to adapt conventional EMT simulation to a parallel GPU-based implementation of EMT simulation was presented in this chapter. This chapter started with a brief overview of the state of the art techniques for EMT simulation. Then it presented methods to parallelize computations of various power system equipment such as generators, transmission lines, etc. An effective parallelization technique suitable for GPU-based matrix-vector multiplication (the most time consuming part of EMT simulation) was implemented and presented in this chapter. This approach ensured the effective use of GPU shared memory to speed up EMT simulation by introducing an intermediate stage, which enabled the use of more threads in the matrix-vector multiplication process. A typical power electronic subsystem was included in the simulation to demonstrate the capability of the GPUs to simulate electromagnetic transients for various switching devices. This chapter also introduced a sparsity technique that was included in the GPU-based implementations, to further accelerate the conventional EMT-simulation. Then, various test cases used to demonstrate the acceleration of conventional EMT-simulation using GPU-computing were presented. Two basic test systems (one with only 3-busses and the other with 39-busses) were introduced and were used to create larger test cases suitable for evaluating simulation performance on GPU based platforms. Two different simulation cases with different interconnection density were introduced so that investigation on performance based on granularity could be explored. Finally, this chapter showed a special instance

of memory access for matrix-vector multiplications, which ultimately acts as a barrier in getting optimum performance gain for larger power systems. It presented the approaches used in this thesis to overcome this barrier. Performance comparisons between GPU-based EMT simulation over conventional EMT simulation are presented in the next chapter.

Chapter 5

Performance improvements and evaluation of GPU-based EMT simulation

5.1 Introduction

EMT simulation was introduced in Chapter 3. The algorithmic steps to parallelize this EMT simulation for GPU-based computations were introduced in Chapter 4. Various test cases used to demonstrate the performance of the proposed algorithm were also introduced in Chapter 4. This chapter presents performance results obtained using the described parallelization techniques for EMT simulations on the GPU. To evaluate the improvements in total computation times, two separate programs (one written in CUDA-C to run on the GPU using the parallel EMT algorithm proposed in Chapter 4 and the other written in ANSI-C to run on the CPU in sequential manner) were created for each of the test cases presented in Section 4.5 of Chapter 4. Additionally, two separate programs (to run on the CPU and on the GPU as before) were created to implement the sparsity handling technique that ignores the unneces-

sary multiplications involving zeros. One of the test cases of the presented approach implements a network with 3861 buses, which includes detailed models for 990 three phase-electrical generators, 594 switches (diodes), 990 three-phase transformers, etc. It should be noted that the presented program for GPU-based EMT simulation is capable of modelling larger networks than the above. The duration of simulation in all of the test cases was taken to be 10 seconds and the simulation time-step used in this work was $50\mu s$. To quantify the improvements in performance, total clock times for GPU-based simulation of various test cases were compared with the total clock times for conventional CPU-based simulation of those test cases. It should be noted that CPU-based simulation means sequential, single core implementation of EMT simulation on the CPU. Similarly, GPU-based implementation means the parallel version of the EMT simulation that run on multiple processing cores on the GPU. To quantify the improvements in the GPU-based simulation process performance gain (β_{GPU}) defined by the equation below [37] was used:

$$\beta_{GPU} = \frac{\text{Total clock time for CPU only simulation}}{\text{Total clock time for simulation using CPU \& GPU}} \tag{5.1}$$

Details on the results of simulating various test cases (as presented earlier) using the described parallelized EMT simulation and sequential EMT simulation and performance gain using GPU-computing will be presented in this chapter.

5.2 Details on the workstation used

In this thesis a hybrid workstation consisting of GPUs and multi-core CPU was used. The operating system on the workstation used was Linux (distribution Fedora 14) [74]. Details of the hybrid platform are listed in Table 5.1. The CPU processors in the workstation were chosen to be more powerful (in terms of speed and cache memory) than those on the GPU. The CPUs of this workstation use a 'Sandy Bridge'

Table 5.1: Details of the hybrid simulation platform

Main Computer (CPU) details	
Type	Intel core i7 CPU 2600K
CPU Clock rate	3.40 GHz
Total RAM	16GB
GPU Details	
Type	NVIDIA GeForce GTX 590
Number of multiprocessors	16
Number of cores	512
GPU Clock rate	1.26 GHz
Global memory	1.5GB
Shared memory per block	64KB
Warp size	32
Max. No. of threads per block	1024

Figure 5.1: Pictorial view of the workstation used in this work (GPUs are shown connected to the motherboard).

architecture, which (at the time of writing this thesis) is one of the top ranked CPUs in the market in terms of overall performance, value, etc. [75]. On the other hand the GPU was chosen to be an ordinary one, with moderate rating in performance and speed [37]. The main reason behind choosing the less powerful GPU (compared to CPU) was to show the capability of an ordinary GPU to computationally outperform the most powerful CPUs. A labelled photograph of this workstation is shown in Fig. 5.1. In this workstation, two GPUs (Nvidia GTX GeForce 590) are mounted on the motherboard through the PCIe buses, which connect these GPUs to the Intel core i7 2600K CPU. The Intel core i7 2600K CPU is a 4-core CPU based on the above mentioned Sandy Bridge architecture, which has a clock speed of 3.40GHz and was installed with an external RAM of 16GB. Each of the Nvidia GTX GeForce 590 GPUs houses two GeForce 580 equivalent GPUs [37]. Each GPU (i.e. GeForce 580) has 512 processing cores, capable of performing floating point operations in parallel. This makes for a total of 2048 processing cores available in this workstation. There are 1.5 Giga-Bytes of DRAM on each of these GPUs (i.e. GeForce 580) and the clock rate of each processor on the GPU is 1.26 GHz. Other details of these GPUs are listed in Table 5.1.

5.3 Overview of the preliminary findings

The author of this thesis reported the very first work on the GPU-based EMT simulation in a conference in 2011 [33]. Further progress of this illustrated GPU-based EMT simulations have been published in subsequent years [33, 34, 35]. Most recently we submitted a journal paper [39] based on the the most advanced algorithmic changes (such as new algorithm for matrix-vector multiplication, optimized approaches for history current calculations, inclusion of parallelized model for various power system equipment, etc.) applied on the GPU-based EMT simulations, which is currently

under review. Before presenting the most recent results on accelerating EMT simulations using GPU computing (by using the algorithms and parallelization techniques presented in Chapter 4), a brief overview of the earlier published works of the author of this thesis will be presented.

A preliminary study on accelerating EMT simulation using random networks was published in [33]. Some important findings were reported in [33], which included the earlier discussed conclusion that GPU-based computations are not suitable for small networks (i.e. networks with few hundred buses). The GPU-based computations require the necessary data to be transferred to the GPU memory from the CPU memory before any EMT simulation related computations to be performed. When the network size is small, this transfer of data between the CPU and the GPU may require a longer time compared to the time required for the actual computations for the whole network. It was shown in [33] that a system of at least 55 nodes (appr.) is required to outperform CPU-based computations using GPU computing. Finally, simulation results using GPU-based EMT simulations were compared with those from commercial tool (PSCAD/EMTDC) [13].

The second paper was published in a conference in Montreal, Canada in 2012 [34]. In this paper the total computational times in simulating various parts of the EMT algorithm on the CPU and the GPU were reported. While parallelizing an algorithm it is critical to determine the core portion of the algorithm, where the bulk of the computations are performed. In this paper four different approaches were implemented to determine the most time consuming part of the EMT simulation, as follows:

1) **Implementation # 1: ALL/CPU**

In this case, all the computations related to the simulation of various electrical

networks were performed on the CPU in a sequential manner (i.e. a single core implementation of the algorithm). This approach was implemented to estimate the total computation time required by sequential simulations (as is the case in commercially available tools, such as PSCAD/EMTDC [13], which run the simulation in a sequential manner). This CPU-based sequential implementation was aimed to ensure homogeneity in the modelling. The exact modelling and simulation techniques of the commercially available sequential tools (such as PSCAD/EMTDC) were not available and it was very important to compare the performance improvements in simulating the exact same algorithm (implemented in serial on the CPU and in parallel on the GPU). Therefore, this paper implemented the same algorithm on the CPU in sequential and later on the GPU using the parallelization techniques. Total clock times for this implementation is used (later) in (5.1) to calculate the speed up using GPU computing. It should be noted that GPU is not used in this implementation (i.e. *Implementation # 1* is a CPU only implementation).

2) **Implementation # 2: MV/GPU**

MV/GPU stands for matrix-vector multiplication performed on the GPU. In this implementation, computations related to a portion of the EMT algorithm (i.e. the matrix-vector multiplication) were performed on the GPU, and the CPU was responsible for the rest of the computations related to the EMT simulation (such as history current calculation, generators related computations, etc.). It is to be noted that this matrix-vector multiplication related algorithm was in its primitive stage (i.e. parallelism was not explored efficiently and the number of parallel threads deployed were significantly less than the number of parallel threads deployed in the latest $2D$ version of matrix-vector multiplication as presented earlier) compared to the one presented in this thesis in Chapter 4. Even this relatively

primitive algorithm showed significant performance gain in the simulation process. The results in this implementations showed that matrix-vector multiplication consumes upto 90% of the total clock times for simulation.

3) **Implementation # 3: MV+H/GPU**

MV+H/GPU stands for matrix-vector multiplication and history currents computations performed on the GPU. In this implementation computations related to history currents computations and matrix-vector multiplications were performed on the GPU. It should be noted that the difference between **Implementation # 2** and **Implementation # 3** is the inclusion of history currents computations on the GPU. In this case, the CPU was responsible for updating the vector of source currents (which is the most sequential part of the EMT algorithm), storing output variable values, flow control of the simulation, etc. Performance gain in this case was higher compared to the previous case (i.e. *Implementation # 2*). Details on performance gain using this step can be found on [34].

4) **Implementation # 4: ALL/GPU**:

In this implementation all the computations related to the EMT algorithm, such as history currents computations, matrix-vector multiplications, updating the current vector, etc. were performed on the GPU. The difference between **Implementation # 3** and **Implementation # 4** is the inclusion of the current vector related computations on the GPU. In this case the CPU was only responsible for the flow control of the simulation and storing the output variables. As mentioned before, the time required to transfer various data between the CPU and the GPU acts as a delay in the simulation (worse for smaller networks being simulated). In this case, the CPU was responsible only for sending the control instructions regarding the

simulation, which ensures the minimum information transfer to control the simulation. In [34], it is shown that performing all the computations, including the highly sequential ones, related to EMT simulations on the GPU is more efficient than implementing them on the CPU. Therefore, in this thesis, all the computations related to EMT simulation were performed on the GPU.

Several test cases were simulated using the above mentioned cases on the CPU in a sequential manner and on the GPU in parallel. Based on the test results it was determined that matrix-vector multiplication is the most time consuming part in EMT simulation. It should be noted that the test cases used in [34] did not include transmission lines, generators, transformers, etc. Inclusion of these equipment models will increase parallelism in the network as well, as will be shown later. It was also shown that performing all the computations on the GPU actually accelerates the overall simulation. As discussed earlier, some tasks of EMT simulation may seem more suitable for implementation on the CPU, but implementing those algorithms on the GPU saves the time in moving the associated data between the CPU and GPU, which incurs significant delay.

In [35], transformers and generator models were included in the simulation. Details on modeling transformers were presented in Chapter 4. It was shown that inclusion of detailed models for such equipment in GPU-based implementations accelerated EMT simulations considerably. However, the results presented in [34] did not include switching effect, models for transmission lines and partitioning larger networks into smaller subsystems using transmission lines, etc. Additionally, the matrix-vector multiplication algorithm was not as efficient as the one described in this thesis. In all of the preliminary assessments the inverse of the admittance matrix was performed off-line using the commercially available tool *Matlab* and was inserted into the sim-

ulation at the beginning (as there was no switching in the network, the admittance matrix was constant). As introduced earlier, these inverse admittance matrix related computations are now entirely implemented on the CPU, based on the Gauss-Jordan elimination algorithm [57]. This makes the described GPU-based EMT simulation environment a complete one that does not require any external computing resources such as *Matlab*. More details on various implementations and test cases used in those approaches may be found in [33, 34, 35].

5.4 Performance improvements for simulation of test systems with low granularity

This section presents the computational performance improvements seen for the simulation of the test cases created by interconnecting several instances of *'building block 1'*, as presented earlier in Chapter 4. A typical test system with low granularity was presented schematically in Fig. 4.14. As presented earlier, the ratio of the number of interconnecting transmission lines between the subsystems inside the network to the size of the electrical network modelled on the GPU is defined as a measure of granularity. The test systems simulated in this section have an asymptotic value of the granularity equal to 0.154 as the network size increases towards infinity (i.e. $\lim_{N\to\infty}$granularity(N) = 0.154, where N is the size of the network).

Total clock times for simulating less granular test cases, created by interconnecting *building block* 1 system of Fig. 4.13, are presented in Table 5.2. It should be noted that the test systems of Table 5.2 have a smaller number of transmission lines with reference to the total number of busses in the network, compared to the more granular test systems, which were constructed using the smaller subsystem *building block* 2, as presented schematically in Fig. 4.16. This table presents the total clock times

Table 5.2: Total time for simulation and performance gain for test cases created by interconnecting *building block* 1 system of Fig. 4.13

No. of Buses	No. of T-Lines	Granularity	Without Sparsity			With Sparsity		
			GPU Only (s)	CPU Only (s)	Gain β_{GPU}	GPU Only (s)	CPU Only (s)	Gain β_{GPU}
39	0	0	11.890	22.280	1.874	11.250	11.560	1.028
78	3	0.0385	14.530	44.019	3.029	14.080	27.761	1.972
156	9	0.0577	14.530	89.331	6.148	14.650	55.747	3.805
273	21	0.0769	14.750	154.434	10.470	14.760	98.223	6.655
858	102	0.1189	23.060	491.358	21.307	19.790	310.882	15.709
936	114	0.12179	23.370	535.942	22.932	20.380	337.218	16.546
975	120	0.12308	23.850	561.537	23.544	20.780	354.224	17.046
1365	174	0.12747	28.461	784.544	27.566	24.200	494.932	20.451
1599	204	0.12758	29.940	926.433	30.942	25.660	578.723	22.553
3471	474	0.13656	52.407	2013.161	38.414	42.879	1261.134	29.412
3861	534	0.13831	57.146	2249.380	39.362	46.888	1403.705	29.937

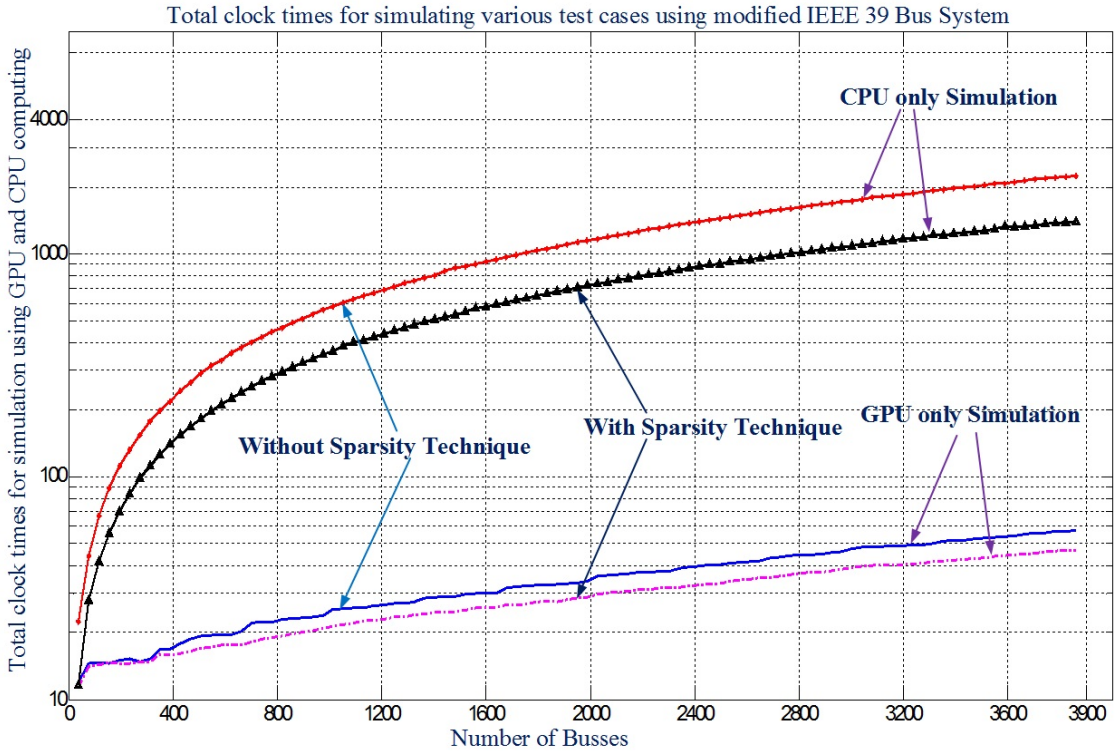


Figure 5.2: Total clock times for simulating various test cases with CPU and the parallelized GPU implementations created by interconnecting *building block* 1 (Fig. 4.13).

and performance gain for executing the programs (sequentially on the CPU and in parallel on multiple cores on the GPU) for dense matrix as well as for sparsity-based implementations. The number of buses, granularity, and the number of transmission lines for each system size are also indicated in Table 5.2. Various times are shown for simulation without and with the sparsity technique. The performance gain, β_{GPU} is also shown without and with the sparsity technique. Fig. 5.2 shows the total clock times for various simulations, as presented in Table 5.2, in graphical form by plotting total clock times for CPU and GPU-based computations for dense matrix based and sparsity-based simulations. It should be clear from Fig. 5.2 and Table 5.2 that GPU-based computations provide significant speedup over CPU-based implementations (even with the inclusion of sparsity based technique). As an example (in Fig. 5.2 and Table 5.2), for a system with 3861 busses and without sparsity, the total clock times for simulation are 57.146 s and 2249.380 s on the GPU and CPU, respectively. On the other hand, for the same system of 3861 busses and with sparsity, the total clock times for simulations become 46.888 s and 1403.705 s respectively.

In the CPU-based implementations, all the computations related to EMT simulation are performed sequentially, hence the total clock times to finish the simulation are longer. As can be seen in Table 5.2, the total clock time for simulation of the 3861-bus system on the CPU is more than half an hour. On the other hand parallel computation techniques are applied on the GPU-based implementations and therefore the total clock times for simulation are much less compared to the CPU-based implementations, as can be seen for the case of the 3861-bus system, the total clock time is less than a minute.

Application of the sparsity technique on the inverse admittance matrix also reduced the total computation times by eliminating the multiplications involving zero as can

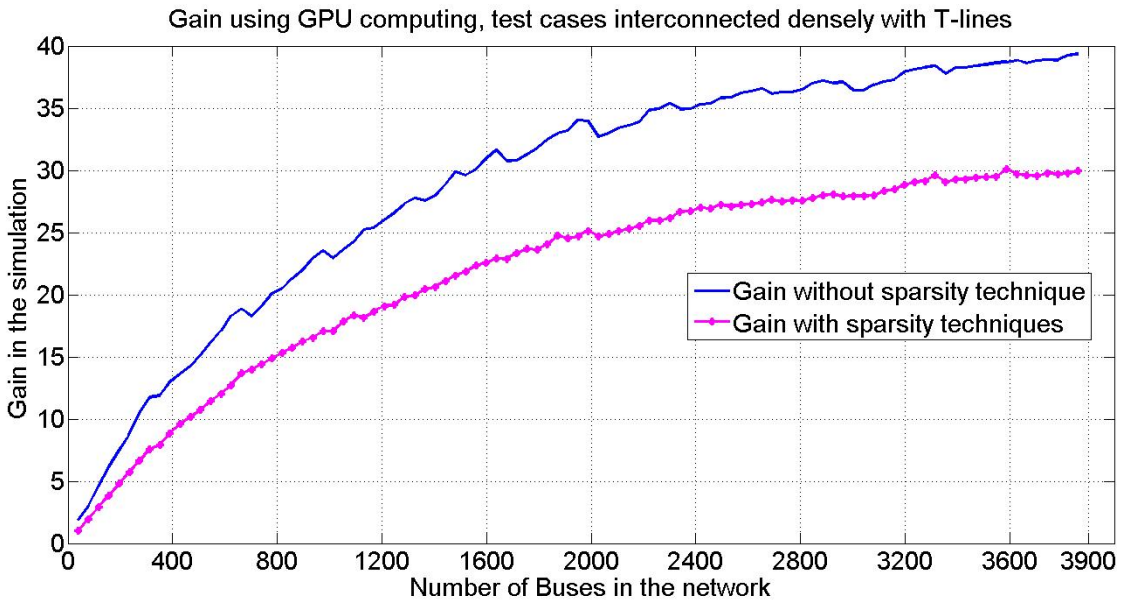


Figure 5.3: Computational performance gains with GPU computing, with and without sparsity (for total clock times presented in Table 5.2).

be seen from Fig. 5.2 and Table 5.2. The benefits from exploiting sparsity are significant for the CPU implementations, with a reduction in computation times from 2249.380 s to 1403.705 s, which means a speedup of 1.602. This is as expected because the elimination of unwanted computations involving zeros, greatly decreases the total number of sequential multiplications involving zero, on the CPU. However, there is only a smaller benefit (i.e. from 57.146 s to 46.888 s, which is a speedup of 1.193) on the GPU implementations, this is because the tasks have already been parallelized into smaller blocks involving parallel threads and the computation benefits from additional sparsity are thus reduced. Additionally, the exploitation of sparsity also reduces the total number of parallel threads to be deployed, which on the other hand is equivalent to increasing the total clock times for the simulation. This can be explained with an analogy. Let us consider a job of measuring customer level voltages for a community of 100-people. Henceforth, this job will be called *work1*. Lets consider there are four scenarios for *work1* as follows:

1) **Scenario** 1: Only one person is assigned to perform *work1*. This is equivalent of sequential implementation of the EMT simulation on the CPU.

2) **Scenario** 2: 20 people are assigned for *work1*. This may be considered as a parallel implementation of *work1*, which resembles the parallel implementation of EMT simulation on the GPU.

3) **Scenario** 3: In this case, the job was redefined with the discovery of redundancy in measuring the voltage for every other customer. Therefore, in this case *work1* has voltages are to be measured for only half of the people in the community (i.e. 50 people). In this case also, only one person is assigned. This is equivalent of implementing sparsity on the CPU in sequential.

4) **Scenario** 4: In this case for the redefined *work1*, only 10 persons are assigned instead of 20 as used in **Scenario** 2. Please note that total number of workforce is reduced to half in this case along with the redefinition of the job. It should be noted that this scenario is similar to the implementation of sparsity on the GPU (parallelism inside the job is reduced and the workforce is also reduced).

Now if we compare the total time required to perform *work1* in **Scenario** 1 with **Scenario** 2, it is obvious that **Scenario** 2 would be much less than the time for **Scenario** 1. It is also obvious that total time for **Scenario** 3 would be much less compared to **Scenario** 1 as the total number of customers was reduced with the discovery of the redundancy. But it should be noted that this savings in time was due to the discovery of the redundancy, which is equivalent of ignoring multiplications involving zeros in the computations. Additionally, the total time for **Scenario** 4 will not be significantly different than the time for **Scenario** 2. As mentioned earlier, in the case of **Scenario** 4 the job was redefined and the corresponding workforce was also reduced compared to **Scenario** 2. This is the situation in the case of the sparsity implementations for EMT-simulations. It reduces the total amount of computations

significantly on the CPU for sequential computations by eliminating the unnecessary multiplications involving zeros. On the other hand it reduces the number of parallel threads to be deployed on the GPU for parallel implementations (i.e. the case of redefining the job and the workforce, case of **Scenario** 4 in the above).

Fig. 5.3 graphically shows the performance gain in the simulation, β_{GPU}, without and with the sparsity technique as presented in Table 5.2. As can be seen, the speed gains for both of the cases are significant. For example, there is about a 40 times speedup for a system with 3861 buses without sparsity (as presented in Table 5.2) and with sparsity, the speedup for the same system size is 30. The programs in the described approach can simulate larger networks and the test cases so far did not reach the limit of the GPU resources. It is, however, clearly visible from Fig. 5.3 that the performance gain is about to reach saturation. Therefore, it is expected that the speed up will increase for larger test cases with more than 3861-buses, as the trend of speed gain is still increasing for the networks so far implemented.

5.5 Simulation-based tests on systems created using *building block* 2 (high granularity)

This section presents the total clock times for simulating test cases with higher granularity compared to those systems presented in the previous section. In this case the size of the basic subsystem (i.e. *building block* 2) is much smaller than the one (i.e. *building block* 1) used in the previous section. Therefore, the number of transmission lines interconnecting these smaller subsystems is increased compared to those presented above with less granularity. The asymptotic value for granularity is 1.33 for the test systems constructed using *building block* 2 (i.e. $\lim_{N\to\infty}$granularity(N) = 1.33, where N is the size of the network). This is in contrast with 0.154 for the systems

Table 5.3: Total time for simulation for various test cases created with the more granular *building block* 2 system of Fig. 4.15

No. of Buses	**Test cases using 3 bus system of Fig. 4.15**					**Test cases using IEEE 39 bus system**				
	No. of 3-phase T-lines	Granularity	CPU times (sec)	GPU times (sec)	Gain β_{GPU}	No. of 3-phase T-lines	Granularity	CPU times (sec)	GPU times (sec)	Gain β_{GPU}
39	32	0.82051	16.426	15.441	1.064	0	0	10.430	9.739	1.071
78	80	1.02564	34.011	23.484	1.448	3	0.03846	23.738	13.022	1.823
117	130	1.11111	51.318	32.611	1.574	6	0.05128	36.290	13.232	2.743
195	226	1.15897	87.761	49.335	1.779	12	0.06154	60.086	13.327	4.509
234	276	1.17949	103.348	58.480	1.767	15	0.06410	66.642	14.406	4.626
273	324	1.18681	123.364	67.356	1.832	21	0.07692	87.204	14.161	6.158

presented in the previous section. Table 5.3 presents the total clock times for simulation of the various test cases created by interconnecting the *building block* 2 system of Fig. 4.15 on page 67. These subsystems have only 3 buses and finer granularity compared to those systems created using *building block* 1 subsystem of Fig. 4.13 of page 65, and are interconnected with transmission lines as presented schematically in Fig. 4.16 on page 68. Table 5.3 also includes total clock times to simulate similar test cases created by interconnecting *building block* 1 system, without the power electronic subsystem (this is essentially the standard IEEE 39 bus system). It should be noted that the presence of switching in the subsystems does not affect the granularity. The granularity depends on the number of subsystems and the interconnecting transmission lines but does not depend on the content of a subsystem. Total clock times for simulating various test cases with larger granularity are presented in Table 5.3. Once again, the duration of each simulation run was 10s, which used a $50\mu s$ time-step. Table 5.3 lists the number of buses, number of transmission lines, granularity and total clock times for simulating the various test cases on the CPU and GPU. It is to be noted that the subsystem admittance matrices for various test cases created by interconnecting *building block* 2 are much smaller in size compared to those created

by interconnecting *building block* 1. Therefore, it was expected that the test cases with smaller subsystems sizes would run faster than the others. But the actual simulation results (Table 5.3) are to the contrary. For example, for the more granular system with 273 buses, constructed with the 3-bus *building block* 2, total simulation times are 123.364 s and 67.356 s respectively on the CPU and GPU. On the other hand, total simulation times were 87.204 s and 14.161 s respectively on the CPU and GPU for the less granular system, with the same number of buses constructed using the IEEE 39 bus *building block* 2. This is because the transmission lines increase the computation burden in the test systems, which effectively reduces the total amount of parallelism (as discussed earlier). Hence proper attention must be given to the number of interconnecting transmission lines and the size of the subsystems, while partitioning big networks into smaller subsystems. Initially, when only a few transmission lines are present in the network, their computational burden is much smaller than the computation time for the admittance matrix based network solution. However, increasing the number of transmission lines causes the computational burden of transmission lines to become comparable to the admittance matrix related computations. Therefore, partitioning a large power system into many smaller subsystems using transmission lines is not always efficient. For example, faster simulation may result if larger systems are modeled on the GPU (i.e. reduced granularity) for a system with extensive granularity.

5.6 Typical simulated wave-shapes of the proposed GPU based EMT simulation

The main focus of this thesis was to demonstrate the performance gain in EMT simulations using GPU computing. Therefore, simulated voltage and current wave-shapes for large networks were not presented so far. It should be noted that the compar-

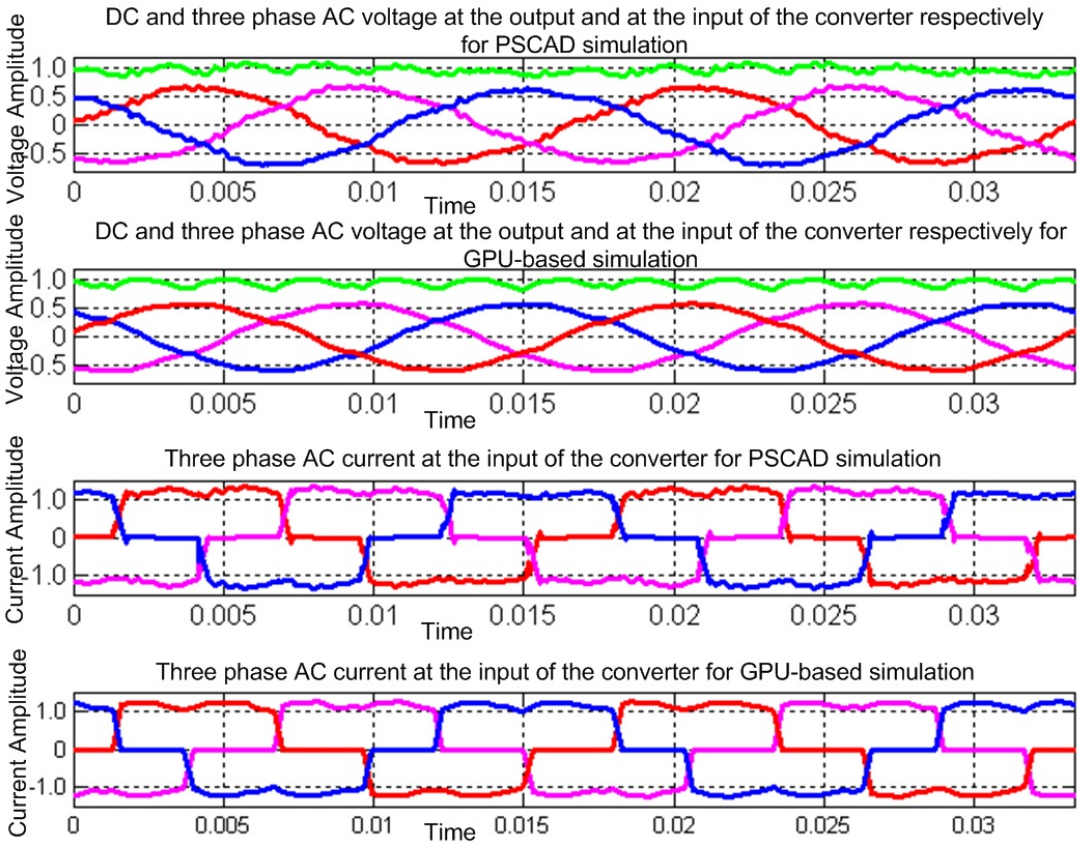


Figure 5.4: Various voltages (AC, DC) and AC currents entering the converter in a 312 Bus system (simulated using the proposed GPU-based EMT simulation and the commercial tool, PSCAD/EMTDC).

ison of GPU-based EMT simulation with a commercially available simulation tool (i.e. PSCAD/EMTDC) was presented in Chapter 3, with a typical circuit example. This section presents typical output voltage and current waveforms for a large network, for visual demonstration purposes (simulated using the proposed GPU-based EMT simulation and the commercial tool PSCAD/EMTDC [13]). Fig. 5.4 shows the DC voltage at the output of the converter, the AC voltages at the input of the converter, and the AC-currents entering the converter, simulated using the proposed GPU-based EMT simulation, and the PSCAD/EMTDC for a system of 312-bus. DC voltage is the rectified version of the ac voltage (due to the diode-bridge) at bus 277 of a 312 bus test system. The 312-bus test system was implemented by intercon-

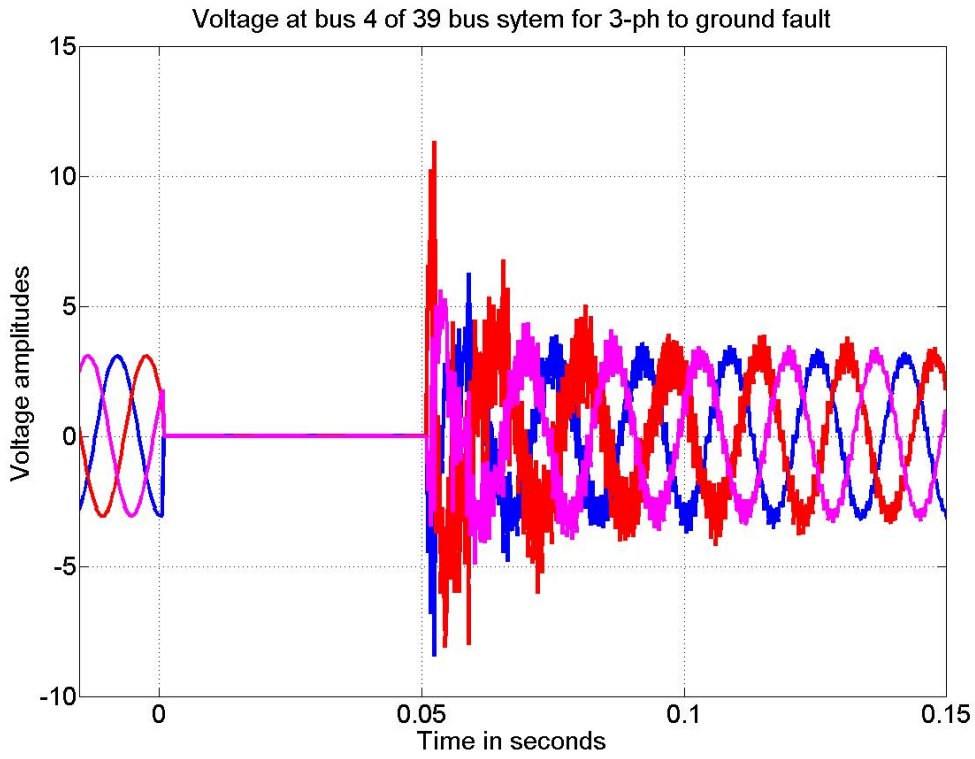


Figure 5.5: Voltage at bus 24 of a 39 bus system for a 3-phase to ground fault.

necting the *building block* 1 subsystem using transmission lines. Note that the diode bridge was included only to demonstrate that switching could be included in the formulation (as illustrated earlier). It is unlikely that such a bridge would be connected in any real power system. A better example would have been to include an HVDC transmission system. It should be noted that inclusion of HVDC systems (which will require additional control sequences to determine the desired switching instances) will not change the algorithms used to implement this basic power electronic switching. Additionally, the main focus of this research work was on the development of the simulation methods. Therefore, modelling and simulation of a more realistic power system with HVDC or FACTS devices is left for future research. It should be noted that switching operation changes the admittance matrix of the network, which requires a new inverse admittance matrix to be inserted in the simulation. Therefore,

implementation of switching requires additional logical operations in the algorithm to determine the exact switching instant, which ultimately increases the computational burden in the simulation. Therefore, switching was not implemented so far on the GPU-based EMT-simulations. Similarly, AC-voltage at the input of the converter (as shown in Fig. 5.4) is essentially the three phase AC-voltage from the source and the AC-currents (shown in Fig. 5.4) are the currents entering the converter during various switching instances. As can be seen from Fig. 5.4, GPU-computing produces exactly the same results as produced by commercial tool (explained and graphically proved earlier in Chapter 3). Similarly, Fig. 5.5 shows the voltage at bus 219, following a 3-phase to ground three cycle fault in a 234 bus test system. Again, this typical fault was applied to demonstrate the capability of the GPUs to correctly handle the computations related to a fault in the network. It should be noted that there is an instantaneous change of the admittance matrix during the fault due to the change in admittances in the particular bus in fault. The transients following recovery of the fault are visible and balanced operation of the network resumes after about 100 ms of fault clearance as can be seen in Fig. 5.5.

5.7 Chapter summary

This chapter presented the improvements in simulation speed achievable by implementing EMT simulations using the presented GPU-based parallel computing techniques. This chapter started with a brief overview of the earlier works on GPU-based EMT simulations published in the literature by the author of this thesis and demonstrated the proposed advanced parallelism techniques used in this thesis. It presented details about the workstation used to demonstrate the presented GPU-based EMT simulations. This chapter presented total clock times for simulating various test cases using two types of granularity in the network. Performance improvements using GPU-

based EMT simulation using the presented sparsity technique were also presented for various test cases with different granularity. Significant improvements in performance gain using the described GPU-based parallel EMT simulation were reported in various sections of this chapter. This chapter also demonstrated that dividing larger networks into smaller subsystems using interconnecting transmission lines may result in less performance gain on the GPU (if the granularity is high). Therefore, proper care must be taken while sub-dividing large networks into smaller subsystems. Finally this chapter presented simulated voltage (DC, AC) and AC current wave-shapes for the power electronic converter (simulated using the presented GPU-based EMT simulation and the commercial tool, PSCAD/EMTDC). The simulated AC voltage wave-shape in the event of a fault was also presented at the end. The conclusions of the thesis and future directions of this proposed GPU based EMT simulation are presented in the next chapter.

Chapter 6

Conclusions and future directions

The main purpose of this thesis was to explore the potential to speed up electromagnetic transients (EMT) simulation of large power systems using graphics processing unit (GPU) based computing. The entire algorithm for the EMT simulation was parallelized to adapt it for the proposed GPU-based simulations. Additionally, various component-related computations (such as, generators, transmission lines, etc.) were also parallelized so that those can be implemented on the GPU in parallel. Various techniques used in the process of parallelization have been discussed in the earlier chapters, which were ultimately used to ensure optimal performance gain in the simulation. Output waveforms produced by the described GPU-based simulation were compared to those produced by professional serial simulations to verify the correctness of the presented implementation. The presented algorithm was evaluated and improvements in performance gain were demonstrated using a workstation consisting of CPU and GPUs. Various test cases to verify the proposed algorithm were designed and simulated using the described GPU-based EMT algorithm.

6.1 The main contributions and conclusions of the thesis

i) A novel and potentially cost effective alternative (as most computers these days are equipped with powerful GPUs, which do not require further investment) to perform and speed up EMT simulation using GPUs is presented in this thesis.

   Various test cases with detailed models for different components have shown that GPU-based computations are significantly faster, even for dense matrix-vector multiplication. So far systems with detailed models of generators, transmission lines, transformers and certain power electronics components in the network have shown a speed gain of 40 for a network of 3870 buses (approx), using GPU computing. It should be noted that the proposed algorithm can handle larger networks than 3870 buses. The total clock times for simulation were approaching saturation at this size of the network (as shown earlier), hence additional test cases of larger size were not implemented. It should also be noted that the GPU-based simulation of 3870 bus system required less than a minute compared to more than half an hour on the CPU alone.

ii) The algorithm inside the EMT simulation was parallelized to adapt it to the GPU-based computations. Various parallelization techniques for different components were proposed and implemented on the GPU. These approaches showed significant performance improvement of the GPU based EMT simulation.

iii) A special algorithm to deploy parallel threads in $2D$-directions, was proposed and implemented on the GPU to speed up the matrix-vector multiplication, which is

required to solve the unknown nodal voltages.

This parallelized matrix-vector multiplication showed notable performance gain in the simulation process. It should be noted that the matrix-vector multiplication alone may consume up to 90% of the total clock time for simulation, as was noted in [34]. Therefore, its efficient parallelization is critical to obtain notable speedup in the simulation process.

iv) A sparsity algorithm for the inverse admittance matrix is introduced to avoid the unnecessary and expensive multiplications involving zeros.

It was observed that implementation of sparsity handling techniques, reduces the total clock times for the CPU-based computations significantly and for the GPU-based computations by a lesser but noteworthy amount. Inclusion of sparsity on the GPU has shown significant performance gain compared to conventional CPU-based simulations.

v) As discussed in Chapter 2, a CPU processor is significantly faster in instruction execution than a GPU-processor. However CPU processors can only perform computations in serial. On the other hand, although the GPU processors are more primitive than those of the CPU, the GPU's parallel processing capability eventually outperforms CPU-processors as the system size increases. The research investigated the minimum size of the network required to outperform CPU-based sequential implementations.

For the equipment used and test networks considered, simulation-based tests

demonstrated that a system with a minimum of 55 nodes is needed for the proposed GPU-based simulations to outperform CPU-based simulations.

vi) The research work also investigated the effect of building-block granularity on the performance gain using GPU computing. There is a tradeoff between subsystem size and the number of interconnecting transmission lines. In this case, the communication overhead for a large number of transmission lines cancels any advantage resulting from smaller matrix sizes. While dividing a large network into smaller subsystems, proper care must be given not to create excessive granularity that requires too many interconnecting transmission lines.

vii) A typical power electronic subsystem was implemented and included in the assessment of this work to demonstrate the capability of the GPU to perform EMT simulation of a real power network.

viii) Parallel implementation of computations related to various power system components (such as, transmission lines, generators, etc) on the GPU in parallel were developed and included in the simulation process.

ix) Finally an algorithm was developed and included in the simulation process to perform history current calculations on the GPU in parallel. This also contributed to the overall performance improvements using the GPU-based EMT simulations.

6.2 Future directions

The algorithms and methodologies developed and explained in this thesis have great potential for future research work. In this section major research directions taking this research as a basis are introduced.

i) The current simulation platform is equipped with only one GPU. Further work may be performed to apply multiple GPUs to simulate larger power systems than those used in this work or to perform multiple simulations at the same time. Multiple simulation at the same time may offer significant benefit to those simulations requiring multiple runs (as individual runs will take significantly less time and those runs will be in parallel themselves).

ii) A typical simulation based study on the effect of interconnection (using transmission lines) density was performed in this thesis. More detailed study of the effect of transmission line density on the simulation performance may open a new window to further speed up EMT simulation.

iii) GPUs lack some fundamental computations based on integer and associated operations (as presented earlier) such as bit-shifts and bitwise logical operations such as (AND, OR, XOR, NOT). EMT simulation especially with switching devices require lots of logical operations to implement switching operations. Therefore, future investigation is needed to perform the additional control related computations (to implement HVDC devices) on the GPU.

iv) In this work allocation of parallel threads for different part of the EMT sim-

ulation, was performed manually. Further work is needed to investigate the allocation of these jobs to the GPU processors automatically.

v) A standard electrical network with more buses and more detailed models for various realistic power systems equipment may be included in the simulation process. For example, simulation of a more realistic power system with HVDC and FACTS devices may open a new window (which may involve new technique to implement switching operations in parallel) for the GPU-based EMT-simulations.

vi) Another direction of future research work would be to investigate the possibility of applying these GPUs for real-time EMT-simulations.

vii) As mentioned earlier, further exploration of the parallel memory access scenario of the matrix-vector multiplication on the GPU may open a new direction for further speed up the simulation process.

viii) This thesis included a CPU version of the matrix inversion algorithm. Further work is needed to implement this matrix inversion on the GPU in parallel. Additionally, in the case of power electronic switches in the network, this matrix inversion has to be able to perform online (i.e. on the fly) computations. Future work may be performed to implement online GPU-based matrix inversion for EMT simulation.

ix) Finally, further work could be done in the direction of developing a dynamic simulation tool for GPU-based EMT simulations. A dynamic simulation tool will be able to partition the network automatically with the capability of choosing the optimal block-size and number of parallel threads per block for the GPUs.

REFERENCES

[1] Neville Watson and Jos Arrillaga. *Power Systems Electromagnetic Transients Simulation.* The Institution of Engineering and Technology (IET), London, United Kingdom, 2003.

[2] Hermann W. Dommel. Digital Computer Solution of Electromagnetic Transients in Single- and Multiphase Networks. *IEEE Transaction on Power Apparatus and Systems*, 44(4):388–399, April 1969.

[3] A.M. Gole. Simulation tools for system transients: an introduction. *IEEE Power Engineering Society Summer Meeting, Seattle, WA, USA*, 2:761–762, July 2000.

[4] Lou van der Sluis. *Transients in Power Systems.* John Wiley & sons Ltd, London, United Kingdom, 2001.

[5] José R. Marti and Luis R. Linares. Real-Time EMTP-Based Transient Simulation. *IEEE Transaction on Power Systems*, 9(3):1309–1317, August 1994.

[6] Hermann W. Dommel. Analysis of large power systems. available online at (accessed on March 19, 2011): http://www.archive.org/details/*nasa*_techdoc_19750021777.

[7] M.O. Faruque, Venkata Dinavahi, and Wilsun Xu. Algorithm for the accounting of multiple switching events in digital simulation of power electronic systems. *IEEE Transactions on Power Delivery*, 20(2):1157–1167, April 2005.

[8] Colin Adamson and A.M.S. El-Serafi. Simulation of the transient performance of synchronous machines on an a.c. network analyser. *Proceedings of the IEE - Part C: Monographs*, 104(6):323–331, September 1957.

[9] IEEE Working Group 15.08.09. Modeling and analysis of system transients using digital programs. *IEEE Power & Energy Society technical report PES-TR7 (Formerly TP133)*, pages 1–100, 1998.

[10] Willis Long, David Cotcher, Dan Ruiu, Philippe Adam, Sang Lee, and Rambabu Adapa. EMTP A Powerful Tool for Analyzing Power System Transients. *IEEE Computer Applications in Power*, 3(3):36–41, July 1990.

[11] R. Kuffel, J. Giesbrecht, T. Maguire, R.P. Wierckx, and P. McLaren. RTDS-A Fully Digital Power System Simulator Operating in Real Time. *IEEE Communications Power and Computing Conference, WESCANEX, Winnipeg, Manitoba, Canada.*, pages 300–305, May 1995.

[12] Alternative Transients Program. available online at (Last accessed on October 20, 2013): http://emtp.org/.

[13] PSCAD/EMTDC, Manitoba HVDC research center. available online at (Last accessed on March 20, 2013): https://pscad.com.

[14] A.M. Gole. Electromagnetic Transient Simulation of Power Electronic Equipment in Power Systems: Challenges and Solutions. *IEEE Power Engineering Society General Meeting, Montreal, Quebec, Canada.*, pages 1–6, June 2006.

[15] IEEE PES Task Force on Frequency Domain Methods for Transient Studies. z-Transform-Based Methods for Electromagnetic Transient Simulations. *IEEE Transactions on on Power Delivery*, 22(3):1799 – 1805, July 2007.

[16] P.E. Crouch, E. Brady, and D.J. Tylavsky. Frequency Domain Transient Stability Simulation of Power Systems: Implementation by supercomputer. *IEEE Transactions on Power Systems*, 6(1):51–58, February 1991.

[17] Prabha Kundur. *Power System Stability and Control*. McGraw-Hill, New York, United States, 1993.

[18] A.M. Gole, T.S. Sidhu, O.B. Nayak, and M.S. Sachdev. A Graphical Electromagnetic Simulation Laboratory for Power System Engineering Programs. *IEEE Transaction on Power Systems*, 11(2):599–606, May 1996.

[19] Aniruddha M. Gole, Shahin Filizadesh, Robert W. Menzies, and Paul L. Wilson. Optimization-enabled electromagnetic transient simulation. *IEEE Transactions on Power Delivery*, 20(1):512–518, January 2005.

[20] A.M. Gole, R.W. Menzies, H.M. Turanli and D.A. Woodford. Improved interfacing of electrical machine models to electromagnetic transients programs. *IEEE Transactions on Power Apparatus and Systems*, PAS-103(9):2446–2451, September 1984.

[21] Hermann W. Dommel and W. Scott Meyer. Computation of electromagnetic transients. *Proceedings of the IEEE*, 62(7):983–993, July 1994.

[22] Fernando L. Alvarado. Parallel Solution of Transient Problems by Trapezoidal Integration. *IEEE Transactions on Power Apparatus and Systems*, PAS-98(3):1080–1090, May 1979.

[23] M.A. Tomim, José R. Marti, and L. Wang. Parallel solution of large power system networks using the multi-area thévenin equivalents (MATE) algorithm. *ELSEVIER, Electrical Power and Energy Systems*, 31:497–503, 2009.

[24] IEEE PES Task Force on Computer and Analytical Methods. Parallel processing in power systems computation. *IEEE Transactions on Power Systems*, 7(2):629–638, May 1992.

[25] Jr. Franklin H. Branin. Computer methods of network analysis. *Proceedings of the IEEE*, 55(11):1787–1801, November 1967.

[26] Kai Hwang and Fayĕ Alayĕ Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, USA, 1984.

[27] C. Larose, S. Guerette, F. Guaya, A. Nolet, T. Yamamoto, H. Enomoto, Y. Kono, Y. Hasegawa and H. Taoka. A fully digital real-time power system simulator based on PC-cluster. *Mathematics and Computers in Simulation - Special issue: Modelling and simulation of electrical machines, converters and systems, Elsevier Science Publishers B. V. Amsterdam, The Netherlands*, 63(3-5):151–159, November 2003.

[28] Changyan Yue, Xiaoxin Zhou, and Ruomei Li. Node-splitting approach used for network partition and parallel processing in electromagnetic transient simulation. *IEEE International conference on power system technology-POWERCON, Singapore*, pages 425–430, November 2004.

[29] Djalma M. Falcáo, Eugenius Kaszkurewicz, and Heraldo L.S. Almedida. Application of Parallel Processing Techniques to the Simulation of Power System Electromagnetic Transients. *IEEE Transactions on Power Systems*, 8(1):90–96, February 1993.

[30] R. Singh, A.M. Gole, P. Graham, S. Filizadeh, C. Muller, and R. Jayasinghe. Grid-processing for optimization based design of power electronic equipment using electromagnetic transient simulation. *25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6, May 2012.

[31] David B. Kirk and Wen mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach* . Elsevier Inc, MA 01803, USA, 2010.

[32] Benjamin Block, Peter Virnau, and Tobias Preis. Multi-GPU accelerated multi-spin Monte Carlo simulation of the 2D Ising model. *ELSEVIER, Computer Physics Communications*, 181:1549–1556, 2010.

[33] J. Debnath, W. K. Fung, A. M. Gole, and Shaahin Filizadeh. Simulation of Large-Scale Electrical Power Networks on Graphics Processing Units. *IEEE EPEC, Winnipeg, MB, Canada*, pages 284–289, October 2011.

[34] J. Debnath, W. K. Fung, A. M. Gole, and Shaahin Filizadeh. Electromagnetic Transient Simulation of Large-Scale Electrical Power Networks Using Graphics Processing Units. *IEEE CCECE, Montreal, QB, Canada*, May 2012.

[35] J. Debnath, A. M. Gole, and W. K. Fung. Electromagnetic Transient Simulation of Large-Scale Electrical Power Networks Using Graphics Processing Units. *International Conference on Power Systems Transients (IPST), Vancouver, BC, Canada*, pages 1–4, July 2013.

[36] Zhiyin Zhou and Venkata Dinavahi. Parallel massive-thread electromagnetic transient simulation on gpu. *IEEE Transaction on Power Delivery*, 3(29):1045–1053, Jun 2014.

[37] NVIDIA. GPU computing: CUDA zone. available online at (Last accessed on March 20, 2013): http://www.nvidia.com.

[38] Vahid Jalili-Marandi and Venkata Dinavahi. SIMD-Based Large-Scale Transient Stability Simulation on the Graphics Processing Unit. *IEEE Transactions on Power Systems*, 25(3):1589–1599, August 2010.

[39] Jayanta K. Debnath and Aniruddha M. Gole and Wai-Keung Fung. Accelerating electromagnetic transients simulation of large electrical power systems using graphics processing units. *IEEE Transaction on Power Delivery (under review)*, Aug 2014.

[40] GPGPU forum. General-Purpose computation on Graphics Processing Units. website: http://gpgpu.org/.

[41] John D. Owens and Mike Houston and David Luebke and Simon Green and John E. Stone and James C. Phillips. GPU Computing: Graphics Processing Units–powerful, programmable, and highly parallel–are increasingly targeting general-purpose computing applications. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[42] Aaftab Munshi, Benedict Gaster, Timothy G. Mattson, James Fung, and Dan Ginsburg. *OpenCL Programming Guide*. Addison-Wesley Professional, Boston, MA, United States, 2011.

[43] Janusz Kowalik and Tadeusz Puźniakowski. *Using OpenCL: Programming Massively Parallel Computers*. IOS Press BV, Netherlands, 2012.

[44] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of cuda and opencl. *IEEE International Conference on Parallel Processing (ICPP), Taipei City*, pages 216 – 225, September 2011.

[45] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, Boston, MA 02116, USA, 2010.

[46] S. Pratap Vanka, Aaron F. Shinn, and Kirti C. Sahu. Computational fluid dynamics using graphics processing units: challenges and opportunities. *Proceedings of the ASME, International Mechanical Engineering Congress & Exposition, IMECE2011, Denver, Colorado, USA*, pages 1–9, November 2011.

[47] Bryan Schauer. Multicore Processors A Necessity. available online at (accessed on Aug 27, 2014): http://www.csa.com/discoveryguides/multicore/review.pdf.

[48] John D. Owens and David Luebke and Naga Govindaraju and Mark Harris and Jens Krüger and Aron E. Lefohn and Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *State of the art report in Eurographicc*, pages 22–51, August 2005.

[49] James Fung and Steve Mann. Using multiple graphics cards as a general purpose parallel computer: application to computer vision. *IEEE International Conference on Pattern Recognition (ICPR)*, 1:805–808, August 2004.

[50] Vahid Jalili-Marandi. Acceleration of Transient Stability Simulation for Large-Scale Power Systems on Parallel and Distributed Hardware. Phd Thesis at the Faculty of Graduate Studies, University of Alberta, Canada. Available online at (accessed on March 20, 2011): http://hdl.handle.net/10048/1266.

[51] Khronos Group. Khronos Releases SPIR 2.0 Provisional Specification. available online at (accessed on Aug 27, 2014): http://www.khronos.org/news/press/releases/khronos-group-releases-opencl-1-1-parallel-computing-standard/.

[52] Omprakash Nayak, Garth Irwin, and Arthur Neufeld. GUI enhances electromagnetic transients simulation tools. *IEEE Computer Applications in Power*, 8(1):17–22, January 1995.

[53] Microtran Power Systems Analysis Corporation. available online at (Last accessed on October 2, 2014): http://www.microtran.com/.

[54] Real Time Digital Simulator. available online at (Last accessed on October 2, 2014): http://www.rtds.com/index/index.html.

[55] EMTP-RV: the reference for power system transients. available online at (Last accessed on March 20, 2013): http://emtp.com/.

[56] Robert L. Boylestad. *Introductory Circuit Analysis*. Prentice Hall, 2002.

[57] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes, The Art of Scientific Computing*. Cambridge University Press, Cambridge CB2 8RU, UK, 2007.

[58] T.L. Maguire and A.M. Gole. Digital simulation of flexible topology power electronic apparatus in power systems. *IEEE Transaction on Power Delivery*, 6(4):1831–1840, October 1991.

[59] S.Y.R. Hui and Christos Christoulos. Modeling of non-linear power electronic circuits with the transmission-line modeling technique. *IEEE Transaction on Power Electronics*, 10(1):48–54, January 1995.

[60] Hermann W. Dommel. Nonlinear and time-varying elements in digital simulation of electromagnetic transients. *IEEE Transaction on Power Apparatus and Systems*, PAS-90(6):2561–2567, November 1971.

[61] A.M. Gole, Albert Keri, C. Nwankpa, E.W. Gunther, H.W. Dommel, I. Hassan, J.R. Marti, J.A. Martinez, K.G. Fehle, L. Tang(Chairman), M.F. McGranaghan, O.B. Nayak, P.F. Ribeiro, R. Iravani, and R. Lasseter. Guideline for modeling power electronics in electric power engineering applications. *IEEE Transaction on Power Delivery, Power Electronics Modeling Task Force & Digital Simulation Working Group*, 12(1):505–514, January 1997.

[62] Chung-Wen Ho, Albert E. Ruehli, and Pierce A. Brennan. The modified nodal approach to network analysis. *IEEE Transaction on Circuits and Systems*, CAS-22(6):504–509, June 1975.

[63] Shengtao Fan and Hui Ding. Time domain transformation method for accelerating emtp simulation of power system dynamics. *IEEE Transactions on Power Systems*, 27(4):1778–1787, November 2012.

[64] José R. Marti, L. R. Linares, J. Calviño, H. W. Dommel, and J. Lin. OVNI: An Object Approach to Real-Time Power System Simulators. *International Conference on Power System Technology, PowerCon'98, Beijing, China*, pages 1–5, August 1998.

[65] Gabriel Kron. Tensorial Analysis of Integrated Transmission Systems, Part III. The "Primitive" Division. *AIEE Transactions*, 71(3):814–821, 1952.

[66] D.A. Woodford, A.M. Gole and R.W. Menzies. Digital simulation of dc links and ac machines. *IEEE Power engineering Review*, pages 36–36, June 1983.

[67] A.J. Gruodis and C.S. Chang. Coupled Lossy Transmission Line Characterizagtion and Simulation. *IBM Journal of Research and Develepment*, 25(1):25–41, January 1981.

[68] C.W. Ho. Theory and Computer-aided Analysis of Lossless Transmission Lines. *IBM Journal of Research and Develepment*, 17(3):249–255, May 1973.

[69] Hermann W. Dommel. *EMTP Theory Book*. Microtran Power System Analysi Corporation, 2nd Edition, Vancouver, 1992.

[70] Louis Bergeron. *Du Coup de Belier en Hydraulique au Coup de Foudre en Electricite (Waterhammer in hydraulics and wave surges in electricity)*. Paris: Dunod (in French), (English translation by ASME Committee, New York: John Wiley & Sons, 1961), 1950.

[71] O. Alsac, B. Stott, and W. F. Tinney. Sparsity-oriented compensation methods for modified network solutions. *IEEE Transaction on Power Apparatus and Systems*, PAS-102(5):1050–1060, May 1983.

[72] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematic, Philadelphia, PA 19104-2688 USA, 2003.

[73] Timothy A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms)*. Society for Industrial and Applied Mathematic, Philadelphia, PA 19104-2688 USA, 2006.

[74] GNU Operating System. available online at (Last accessed on March 20, 2013): http://www.gnu.org/.

[75] Intel® Core™ i7-2600K Processor review. available online at (Last accessed on August 20, 2014): http://ark.intel.com/products/52214.

Appendix A

Schematic algorithm of various *kernel*s used in this thesis

This chapter presents schematic of some of the *kernel*-functions as used in this thesis. As mentioned earlier, matrix-vector multiplication is the most time consuming part in the EMT simulation. The schematic presented in Algorithm 1, explains the algorithm used in performing matrix-vector multiplications. In this case, the whole matrix was divided into number of blocks in the x and y axis directions (as mentioned before in Fig. 4.6 of page 47) based on the programmer (or user) defined variable for the *Block-size*. Each block is assigned with a unique *id* and is accessible from the GPU. In this algorithm, number of threads per block is set during the compilation process. Number of threads in the x and y axis directions (for each block) are equal to the user defined parameter *Block-size*. The algorithm uses a loop counter to perform the computations for all the blocks in the x direction and computations related to the parallel blocks in the y-direction are performed according to their own *Id*s (for various *blocks* and *thread*s), in parallel on the GPU. In this case, all the y-direction *blocks* are created in parallel by the *kernel* call, the size of these *blocks* (as mentioned before) are equal to the parameter *Block-size*. The total number of *blocks* in the

y-axis may or may not be equal to the number of *blocks* in the x-axis. In case of a large test system, whole network is usually divided into smaller sub-systems using transmission lines and hence contain different number of *blocks* in the x and $y-$axis directions. For example, a 975 bus system, as presented in Fig. 4.14 on page 66, had 25-of those *building block* 1 system of Fig. 4.13 in page 65. In this case the admittance matrix had a dimension of 2925×117. If a block size of 3 was used in this particular system (of Fig. 4.13 of page 65), then the number of blocks in the x-direction would be equal to 39 and in the y-axis direction would be 975. A loop (as mentioned in Algorithm 1), moves the multiplication process of various blocks along the x-axis. Fig. A.1 shows the multiplication process in the very first iteration of the loop. In this case, multiplication for all the rows in each block are performed in parallel and also for all the blocks in that column (i.e. column of the admittance matrix in the y-axis direction created by the border of the first iteration loop, as shown in Fig. A.1). Each block contains equal number of threads in the x and y direction. As shown in Fig. A.1, all the threads in each blocks are assigned to perform the multiplication with the corresponding element in the vector. After the multiplication the result is added to the previous multiplication results. This processes continues for all the blocks in the x-axis direction. Finally, all the elements of one row of the temporary matrix containing results of those multiplications are are added together to form a vector and these results are copied to the corresponding block of the resultant vector of voltages (as shown details in Algorithm 1 and Fig. A.1).

The schematic algorithm for updating history currents are presented in Algorithm 2. As mentioned earlier, a table containing the values of inductive and capacitive branches with the information of interconnecting nodes are prepared at the beginning of the simulation process. During the simulation this whole table is divided into various suitable blocks, depending on the amount of branches. During the computations, if

the history current is related to capacitive branch it uses equation 3.2 of page 25 and in case of inductive branches it uses the equation 3.1 of page 25. There were two different *Kernel* functions to perform calculations for capacitive and inductive branch related history currents. Various parameters such as *Block* size, total number of *Threads*, etc are set (by the programmer/user) at the beginning of the simulation.

Algorithm 1 Schematic algorithm for matrix-vector multiplication

1: `Initialize:`
$N_y \leftarrow y-$dimention of the matrix,
$Bs \leftarrow$ Block size,
$Nb \leftarrow (N_y/Bs) \leftarrow$ Number of blocks in the $x-axis$ direction,
$Temp[Bs][Bs] \leftarrow$ Temporaray result storage matrix (set initial values to zero).

2: **for** `(i=1; i<=` Nb`; i++)` **do**
3: `Perform multiplication of each element of each of the rows of the current matrix-block with the corresponding element in the vector and sum the result with the previous result.`
4: **end for**
5: **for** `(i=1; i<=` Bs`; i++)` **do**
6: `Take summation of all the elements of each row of the resultant matrix and save in a vector.`
7: **end for**
8: `copy the results to the resultant vector of voltages`

Algorithm 3, shows the schematic algorithm for performing computations related to transmission lines. Schematic equivalent circuit of transmission lines as used in this thesis (i.e. Bergeron's model) is shown in Fig. 4.9 on page 55. Also, transmission lines related computations are presented in equation 4.3 on page 56. The algorithm starts with converting the sending and receiving end voltages of every transmission lines into modal domain using the modal transmission matrix (it also includes memory access calls for various nodes). Next this algorithm performs computations related to equation 4.3 of page 56 for all transmission lines in parallel (this includes memory read write of previous modal currents as well). Finally, resultant modal currents for

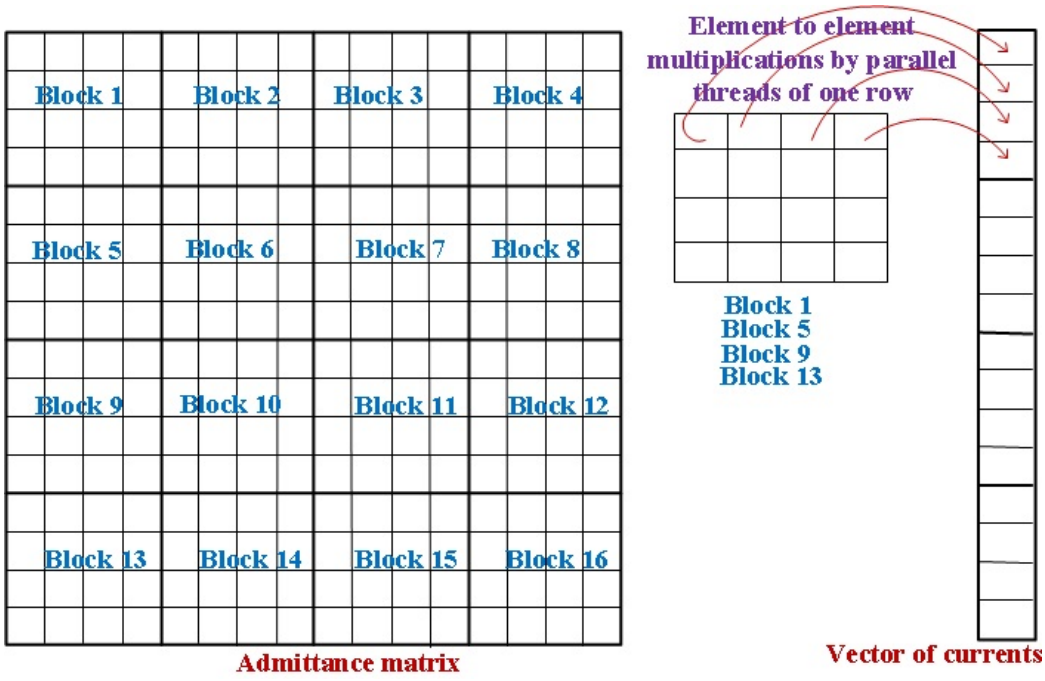


Figure A.1: Schematic of the matrix-vector multiplication on the GPU in one iteration of the inner loop shown in Algorithm 1.

every transmission lines are converted back into time domain using the inverse-modal transformation matrix (details are shown in Algorithm 3).

Next, Algorithm 4 and Algorithm 5 shows the schematic algorithms related to the implementation of electrical generators on the GPU. Schematic equivalent circuit of a synchronous generator with necessary mathematical equations are presented in Fig. 4.8 on page 52. As mentioned earlier, synchronous generators are interfaced with the main network using Norton equivalents. Generators related computations are performed separately in the *dq0*-domain. Algorithm 4 shows the schematic of the *kernel*-function used to re-arrange the generator terminal voltages into a matrix, which ultimately helps in faster access to the updated voltages on the generator terminals. Fig. A.2 shows the schematic of the reformatting of various terminal voltages using this *kernel*-function. As seen in the figure, before formatting the voltage vector contains all the generator voltages in an arbitrary format into a single

Algorithm 2 Schematic algorithm for history currents calculation

1: `Initialize:`

$bx \leftarrow x-$axis component for Block index,
$by \leftarrow y-$axis component for Block index,
$tx \leftarrow x-$axis component for thread index,
$N_{RL} \leftarrow$ Block size for the history current computations

2: `Perform computations related to the history currents using:`
3: *Case of Inductive Branch*:
4: `Equation 3.1 of page 25.`
5: *Case of Capacitive Branch*:
6: `Equation 3.2 of page 25.`
7: `Copy the results to the history current vector`

Algorithm 3 Schematic algorithm for Transmission line related computations

1: `Initialize:`

$Trmat \leftarrow$ modal transformation matrix,
$Trmat_{inv} \leftarrow$ inverse of the modal transformation matrix,
$bx \leftarrow x-$axis component for Block index,
$by \leftarrow y-$axis component for Block index,
$tx \leftarrow x-$axis component for thread index.

2: `Transform sending and receiving end voltages into modal domain using` $Trmat$
3: `Perform transmission lines related computations using equation 4.3 of page 56.`
4: `Convert the modal domain currents (sending and receiving end) back into time domain using` $Trmat_{inv}$

Algorithm 4 Schematic algorithm for the *kernel*-function to rearrange terminal voltages of synchronous generators into a matrix form

1: `Initialize:`

$bx \leftarrow x-$ axis component for Block index,
$by \leftarrow y-$axis component for Block index,
$tx \leftarrow x-$axis component for thread index,
$N_{gen} \leftarrow$ total number of generators in the network,
$Vid \leftarrow$ matrix containing various terminal voltages of the generators with dimension is $N_{gen} \times 3$.

2: `Copy the new magnitudes of the voltages at different nodes (connecting the generators) into the matrix` Vid `(according to the interconnection table containing the interconnection nodes of the generators).`

voltage vector. After reformatting, (as seen from Fig. A.2) the voltages are fit into a new matrix where various phases are assigned along the columns of the matrix and various generators are assigned along the rows of the matrix. This helps in faster access of various generator voltages using their corresponding *block* and *thread Id*s in parallel. Algorithm 5 shows the schematic of various steps used in the other *kernel*-function, which was used to calculate the values for current sources in the Norton equivalents of generators connected in the main network. As mentioned earlier each generators are assigned in a block with several threads to perform computations to perform various computations related to each generator.

Algorithm 5 Schematic algorithm for synchronous generator computations

1: Initialize:
2: $T(\theta)$ $\leftarrow$ time domain to *dq0*-domain transformation matrix, $T(\theta)^{-1}$ $\leftarrow$ *dq0*-domain to time domain transformation matrix, bx $\leftarrow$ $x-$axis component for Block index, by $\leftarrow$ $y-$axis component for Block index, tx $\leftarrow$ $x-$axis component for thread index
3: Transform generator terminal voltages into *dq0*-domain using $T(\theta)$
4: Perform generator related computations using equations shown in Fig. 4.8 on page 52.
5: Convert the equivalent currents (for Norton equivalents sources connected in the main network) into time domain using $T(\theta)$-1

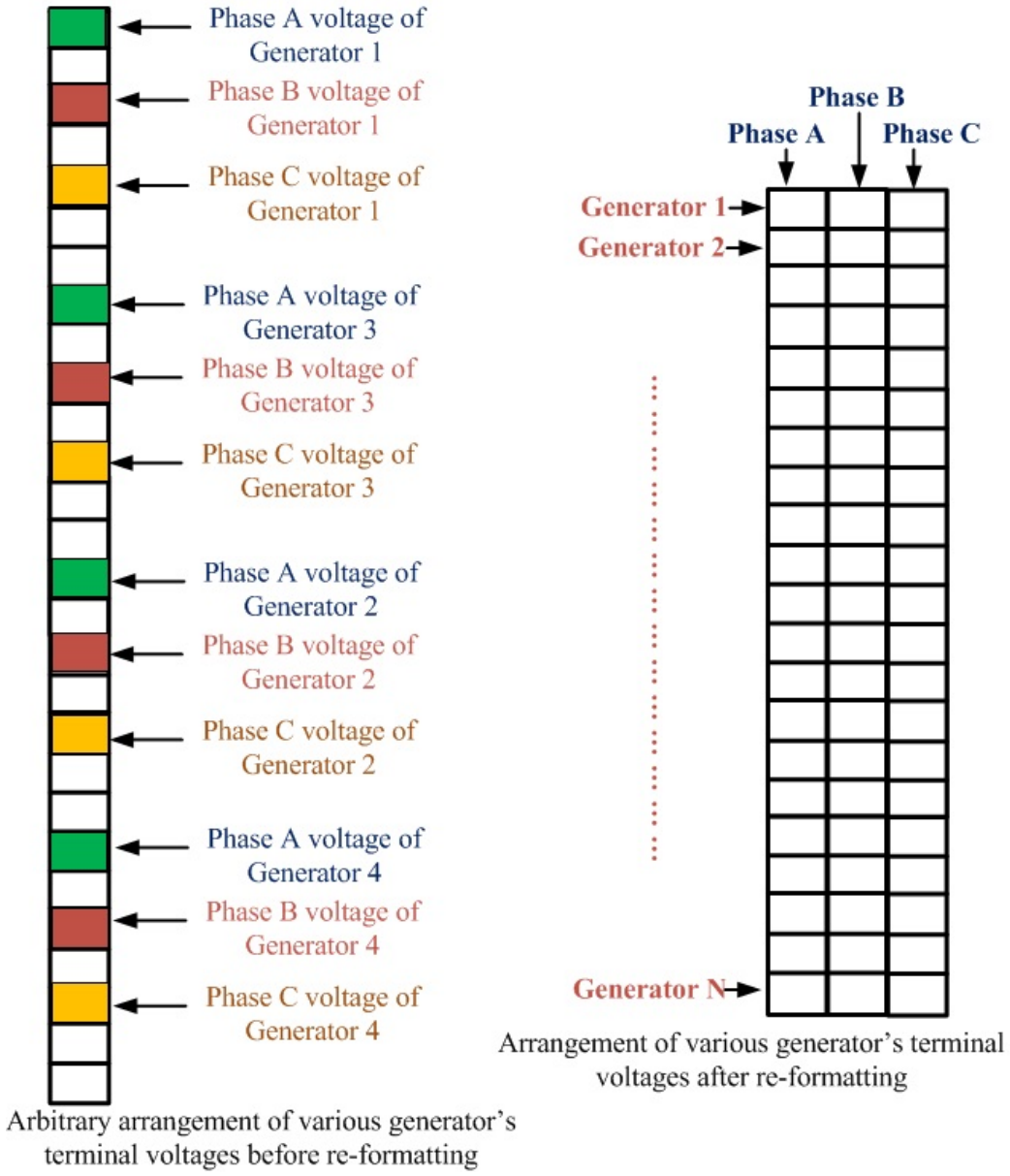


Figure A.2: Schematic of generator terminal voltage reformatting before and after the *kernel*-function execution.

Appendix B

Schematic of some of the test cases used in this work

This Chapter presents schematic of various test cases used in this thesis. These networks do not represent any real system but they may represent hypothetically large systems, which are used (in this thesis) to demonstrate the capability of the GPU to perform computations related to any real power system. Fig. B.1 shows the *building block* 1 test system consisting of an IEEE 39-bus system interconnected with a power electronic subsystem as mentioned earlier. This system was used in the simulation (details were presented before in section 4.5 on page 63) to create very large test cases to implement very large and realistic power systems.

Fig. B.2 shows the schematic of 78-bus test system implemented using the *building block* 1 system shown in Fig. B.1. In this case, two of those *building block* 1 systems were interconnected using transmission lines. Three transmission lines were used as link to interconnect two *building block* 1 systems. Fig. B.3 shows the schematic interconnection of a typical 585-bus test system, used in this work. In this case 15-of those *building block* 1 system were interconnected using transmission lines. Detailed

interconnection of a typical *building block* 1 with its neighbours is also shown in this figure. Similarly interconnected typical schematic for 195-bus and 390-Bus systems are shown in Fig. B.4 and Fig. B.5 respectively. Other larger test cases were created in the similar manner and a typical example of a 975 bus system was presented earlier in Fig. 4.14 on page 66.

Fig. B.6 shows the schematic details of a 78-bus test system created using the

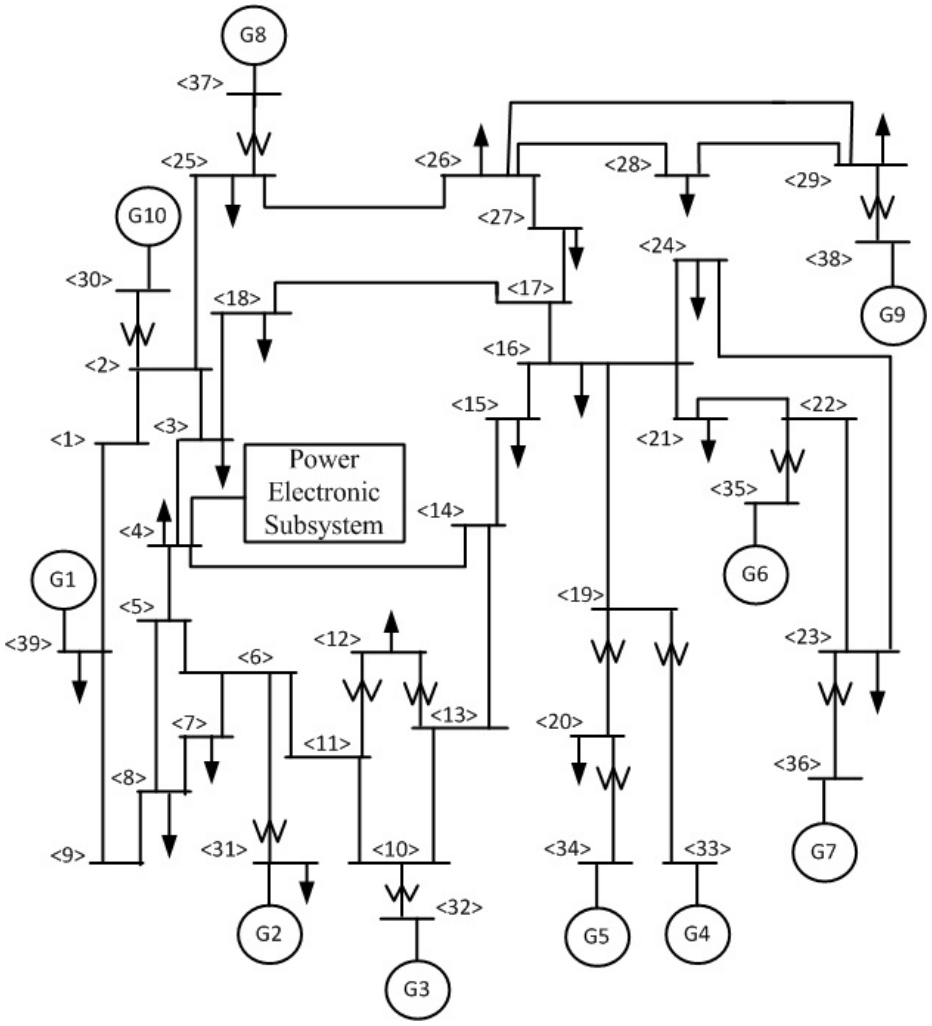


Figure B.1: Schematic of the 39 Bus test system interconnected with a power-electronic subsystem, as used in this work.(links between various buses are implemented using Π-equivalents)

building block 2 system of Fig. 4.15 of page 67. As presented earlier, this *building block* 2 system is a 3-bus system used to demonstrate the effect of communication

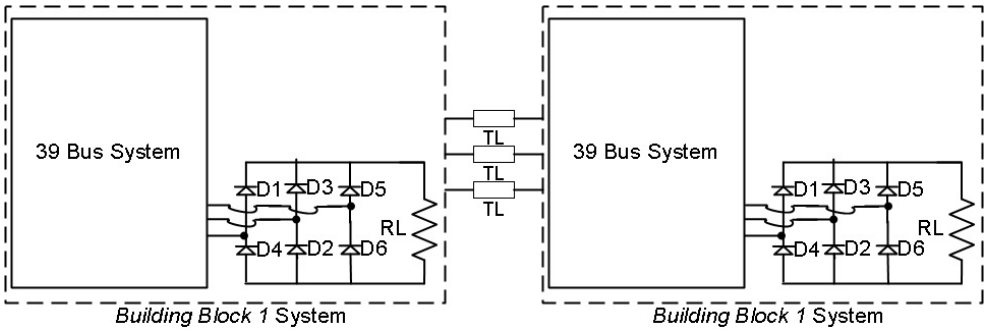


Figure B.2: Schematic of the 78 Bus test system used in this work created by interconnecting *building block* 1 system of Fig. 4.13 of page 65.

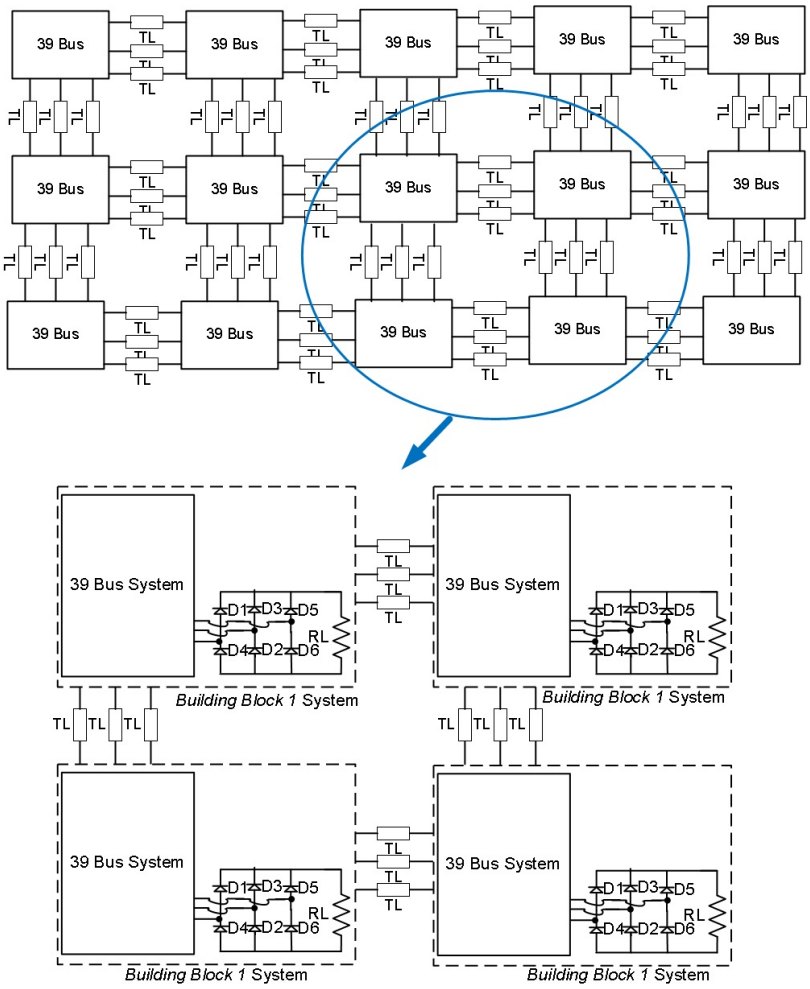


Figure B.3: Schematic of details interconnection of the 585 Bus test system used in this work created by interconnecting *building block* 1 system of Fig. 4.13 of page 65.

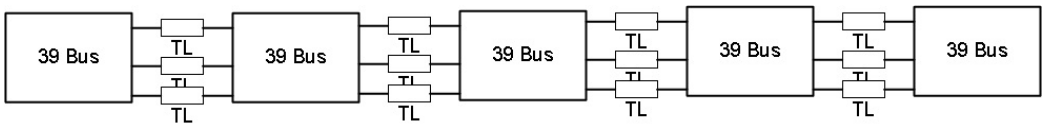


Figure B.4: Schematic interconnection of 195-Bus system as used in this work created using the *building block* 1 system of Fig. 4.13 of page 65.

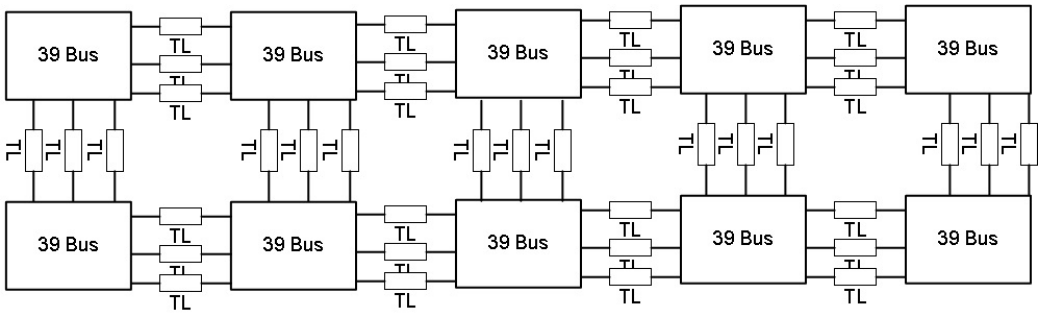


Figure B.5: Schematic interconnection of 390-Bus system as used in this work created using the *building block* 1 system of Fig. 4.13 of page 65.

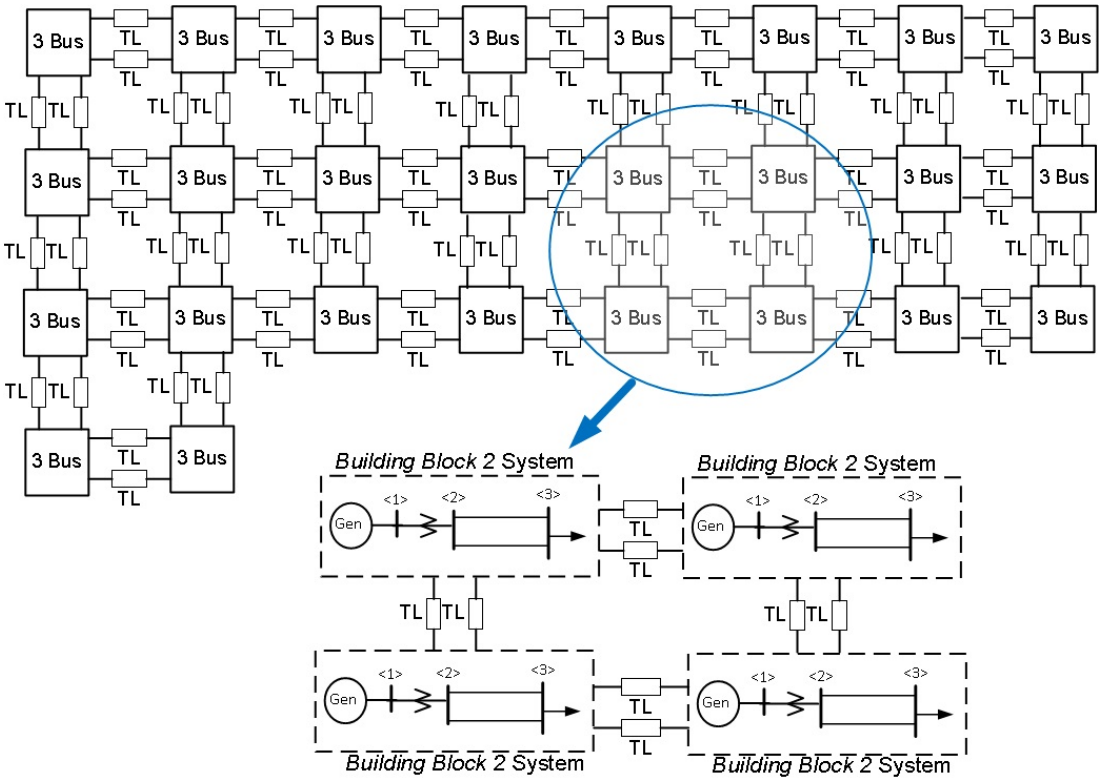


Figure B.6: Schematic of details interconnection of the 78 Bus test system used in this work created by interconnecting *building block* 2 system of Fig. 4.15 of page 67.

bottleneck in the EMT-simulation. Fig. B.6 shows interconnection of 26 of those *building block* 2 systems using transmission lines. This figure also shows the detailed interconnection of a particular *building block* 2 system with its neighbouring blocks using transmission lines. It is to be noted that these systems does not represent any real power system but are hypothetical networks created to demonstrate the effect of having increased number of transmission lines in the network. In this case also larger test cases were created by interconnecting these *building block* 2 systems in the similar manner as shown in Fig. B.6.

Fig. B.7 shows the schematic flow chart of a typical EMT-simulation. It starts with reading the initial conditions for various equipment with a time, $t = 0$ (if initial time is not specified). Then it reads the current admittance matrix and perform computations related to inverse admittance matrix. Then it enters into the main loop. It update the vector of injected currents, determines the unknown nodal voltages. Then it performs computations related to updating history currents for various inductive and capacitive branches. It increases the total time, t by Δt. Then it checks if there was any switching instances in the network or not. In case there was a switching instant it will return to update the current admittance matrix, otherwise it will continue the loop with a time step of Δt until the $STOP$-time of the simulation is reached.

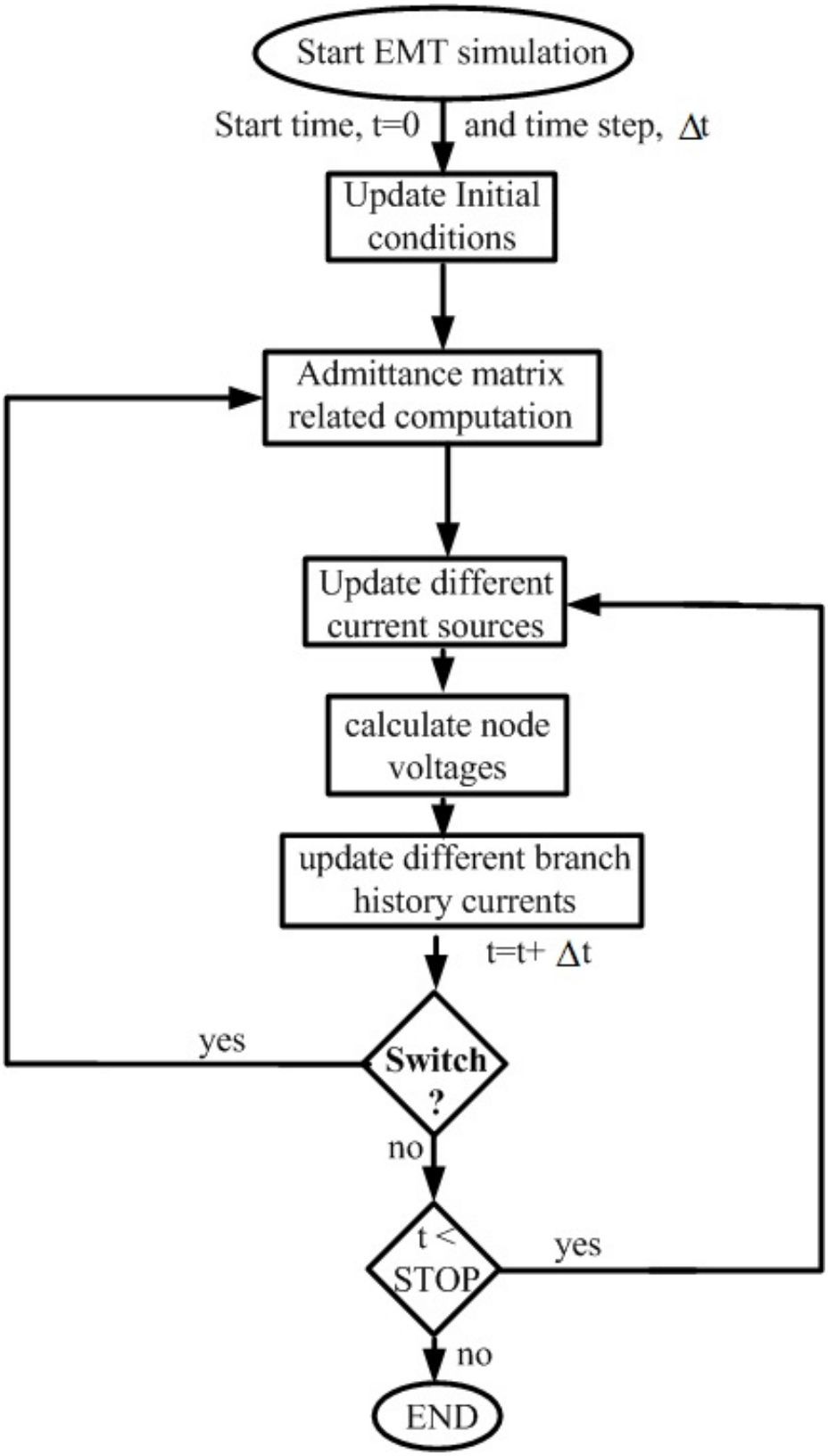


Figure B.7: Schematic flow-chart for EMT-simulation.