Parallel Dynamic State Estimation of Large-scale Cyber-physical Power Systems

by

Hadis Karimipour

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Energy Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Growing system size and complexity along with the large amount of data provided by phasor measurement units (PMUs) are the drivers for accurate state estimation algorithms for online monitoring and operation of power grids. State estimation is the process of estimating unknown state variables in a power grid, which are then used for energy management system (EMS) functions in the control centre. Even with modern computing power, the large volume of computation in the state estimation process is highly time and memory intensive, and a serious bottleneck for online operation and control of the grid.

Furthermore, the deployment of new smart grid technologies in communication and smart metering technologies brings new challenges to the state estimation problem. In addition to failure of physical infrastructure, the smart grid is also sensitive to cyber-attacks on its communication layer. Intelligent cyber-attacks can be designed to be unobservable by the traditional bad data detector. Such attacks can significantly impact the decision making process in control centres and can result in potentially catastrophic outcomes.

The focus of this research is to design parallel computational algorithms to accelerate dynamic state estimation for both static and dynamic states in large-scale power networks. Moreover, a stochastic model is proposed to guard the system against intelligent cyber-attacks. Utilizing the massively parallel architecture of graphic processors and using detailed models of power system components, the proposed research achieved the required accuracy as well as the computational speed-up in the results.

Preface

The material presented in this thesis is based on original work by Hadis Karimipour. As detailed in the following, material from some chapters of this thesis has been published in conference proceedings, and as journal articles under the supervision of Dr. Venkata Dinavahi in concept formation and by providing comments and corrections to the article manuscript.

Chapter 3 includes the results published in following papers:

- H. Karimipour, V. Dinavahi, "Accelerated parallel WLS state estimation for large-scale power systems on GPU", *NAPS*, pp. 1-6, 2013.
- H. Karimipour, V. Dinavahi, "Extended Kalman filter based massively parallel dynamic state estimation", *IEEE Trans. in Smart Grid*, vol. 6, no. 3, pp. 1539-1549, May 2015.

The materials presented in Chapter 4 are published/under publication in following papers:

- H. Karimipour, V. Dinavahi, "On detailed synchronous generator modeling for massively parallel dynamic state estimation", *NAPS*, pp. 1-6, 2014.
- H. Karimipour, V. Dinavahi, "Parallel domain decomposition based distributed state estimation for large-scale power systems", *IEEE Trans. on Ind. App.*, pp. 1-5, Aug. 2015.
- H. Karimipour, V. Dinavahi, "Parallel relaxation based joint dynamic state estimation of large-scale power system", *IET Gen., Trans. and Dist.*, vol. 10, no. 2, pp. 452-459, Feb. 2016.

Chapter 5 has been submitted for peer review as follows:

- H. Karimipour, V. Dinavahi, "Robust parallel dynamic state estimation against cyber-physical attack", *Submitted to IEEE Trans. on Ind. Inf.*, pp. 1-9, Nov. 2015, (TII-15-1635).

To my husband, Amin
who is always a constant source of support and encouragement
and to my parents for their unconditional love.

Acknowledgements

I would like to express my deepest appreciation to my supervisor *Prof. Venkata Dinavahi* for his supportive attitude, encouragement, and guidance through my research at the University of Alberta. Undoubtedly, without his constant help and supervision, this dissertation would not have been possible.

It is an honor for me to extend my gratitude to all my Ph.D. committee members *Dr.Sahar Pirooz Azad*, *Dr. Hai Jiang*, *Dr. Qing Zhao* and the external examiner *Prof. Chi Yung Chung* from University of Saskatchewan for reviewing my thesis and providing thoughtful comments to improve it. And my special thanks go to my colleagues and friends at the RTX-Lab with whom I had a wonderful time during my Ph.D. program.

I would like to thank my husband Amin for his great understanding and support throughout my research.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AC	Alternating Current
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSE	Dynamic State Estimation
DC	Direct Current
EKF	Extended Kalman Filter
GPU	Graphics Processing Unit
IR	Instantaneous Relaxation
MIMD	Multiple-Instruction-Multiple-Data
PMU	Phasor Measurement Unit
SCADA	Supervisory Control and Data Acquisition
SIMD	Single-Instruction-Multiple-Data
SM	Streaming Multiprocessor
SP	Stream Processor
SSE	Static State Estimation

1 Introduction

Continuous growth in electricity demand and complexity of the power systems brings up new challenges in online monitoring and state estimation of large-scale power system. Therefore, power engineers are always exploring methods for fast and efficient solutions for these problems. In addition, the evolution of power systems toward the new smart grid era is increasing the size and complexity of the power grids. Compared with the traditional state estimation, state estimation of smart grids has a lot of key differences. One of the main differences is that, the power grid can be distributed in a variety of environments. The traditional centralized state estimation is not scalable enough to process the huge amount of data generated all over the grid. In addition, the traditional state estimation updates much slower than the measurement cycle, so it is not fast enough to predict the real-time behavior of power grids and respond to emergencies [1], such as the 2003 US Canada Blackout [2].

Accurate state estimation with increasing uncertainties is another grand challenge to the smart grid. As the grid evolves, supervisory control and data acquisition (SCADA) systems will inter-operate with smart meters and communication devices, networks [3] and sensors such as phasor measurement units (PMUs) [4]. This fact results in higher speed at reduced costs; however, it increases the grid's vulnerability to IP based cyber-attacks. The existing bad data identification methods can detect basic attacks, but they may fail in the presence of more intelligent and unobservable attacks.

State estimation is a fundamental problem in monitoring and control of large-scale power systems. There are several key functions in power networks which are developed based on state estimation such as contingency analysis, optimal power flow and economic

dispatch [5]. Therefore a fast and accurate state estimation is invaluable for secure and economical operation of complex power systems.

A state estimator provides accurate estimate of the state variables, which usually includes static states like bus voltages and phase angles or dynamic states such as generator rotor angles and speed. The process is done using remotely captured and periodically collected measurements by SCADA through remote terminal units (RTU) in substations.

The overall state estimation process consists of the following steps:

1. Data acquisition
2. Network topology processing
3. Observability analysis
4. Estimation of the state vector
5. Detection/identification of bad data

The block diagram on Fig. 1.1 shows the components of a state estimator. RTU updates the system regarding the current status of the power system and encode measurement transducer outputs into digital signals. A central master station, located at the control center, gathers information through the SCADA system. Typical measurements include power flows (both active and reactive), power injections, voltage magnitude, phase angles and current magnitude. In addition, direct measurement of voltage phase angle is available through PMUs which are equipped with a GPS receiver allows for synchronization of measurements, yielding accurately measured and time-stamped voltage phase angles.

The network topology processor determines the topology of the network from the telemetered status of circuit breakers. The updated electrical model of the power transmission system is sent to the state estimator program together with the analog measurements.

The state estimator processes all data before being used by other programs, except the analog measurements of generator outputs, which are used directly, by the Automatic Generation Control (AGC) program.

The output of the state estimator consists of all bus voltage magnitudes, phase angles, power injections and power flows. The bad data is also identified, detected and, if possible, eliminated by the estimator. The output data together with the electrical model, developed by the network topology program, provides the basis for the economic dispatch program and contingency analysis program.

Many studies have been conducted online state estimation of large-scale power system. However, parallel dynamic state estimation, considering detailed modeling of generators,

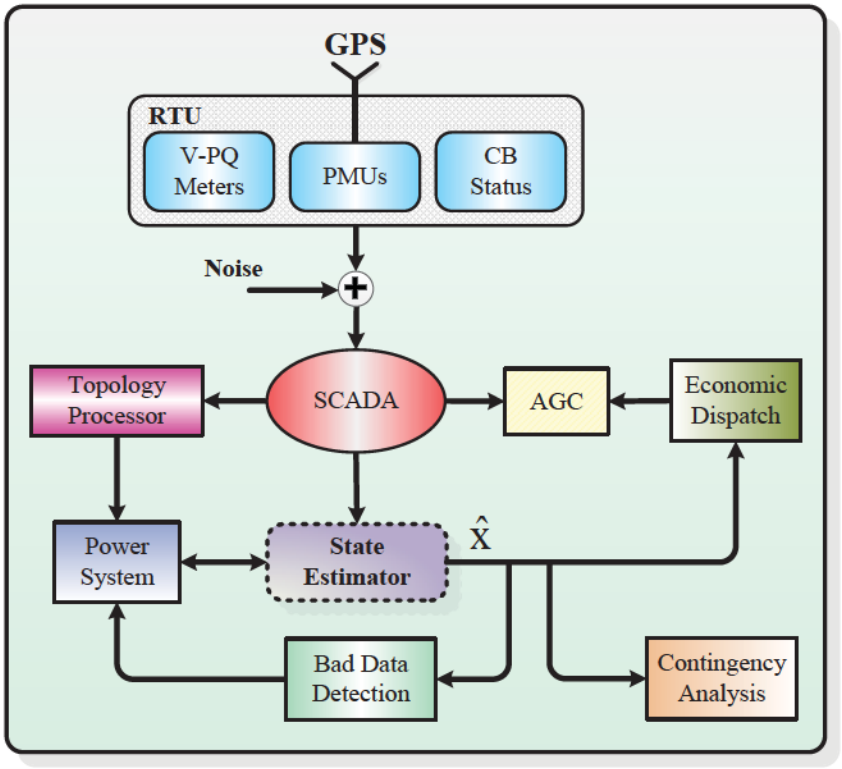


Figure 1.1: State estimation process block diagram.

has so far gained less attention. In this research, the main focus is to develop parallel algorithms for both static and dynamic states estimation on massively parallel graphic processing units (GPU). The popularity of the GPUs in the field of high-performance computing is due to their ability to provide computational power for massively parallel problems at a reduced cost. Using GPUs separate task will be assigned to individual computational core. Therefore, all tasks can be off-loaded and executed in parallel utilizing thousands of threads, thereby accelerating the process of state estimation significantly.

In addition, to maintain both the accuracy and speed, SCADA and PMU measurements will be used simultaneously. The numerical algorithms includes Extended Kalman Filter (EKF) method for state prediction, and Weighted Least Square (WLS) for state estimation. Furthermore, the relaxation based method will be employed for parallel and distributed state estimation of large-scale systems. Moreover, in contrast with the existing state estimation algorithms, which neglect the effect of cyber-attacks in state estimation modeling, we will use a more complex model by considering the stochastic nature of cyber-attacks which offers a robust state estimation method against false data injection attack.

1.1 General Terms and Definitions

In this section the important terms used in this thesis are defined to clearly identify the scope of work done in this research.

1.1.1 Static State Estimation

Static state estimation (SSE) relies on the fact that under normal operation condition, the power system is considered as a quasi-static system [6]. It mainly tries to estimate static parameters of the power system, such as voltage magnitude and phase angle, using the measurers provided by PMUs and SCADA system.

1.1.2 Dynamic State Estimation

Unlike the former method which only uses the current measurements to estimate the network state, dynamic state estimation (DSE) employs the previous states to predict the state one step ahead of the time and then correct it using current measurements [7]. The term dynamic state estimation also implies to the estimation of dynamic states of the synchronous generator.

1.1.3 Supervisory Control and Data Acquisition System

In general, supervisory control and data acquisition system (SCADA) implies to data acquisition systems with data transmission systems and human machine interface software which provide a centralized monitoring and control system for numerous process inputs and outputs. SCADA systems are designed to collect field information, transfer it to a central computer facility, and display the information to the operator graphically or textually, thereby allows the operator to monitor or control an entire system from a central location in real time [8].

1.1.4 Phasor Measurement Unit

A phasor measurement unit (PMU) provides measurements by sampling the AC voltage and current waveforms collected at secondaries of instrument transformers (CT and PT) while synchronizing with a global positioning system (GPS) clock with the accuracy of 1μs. The voltage and current phasors are then transmitted to the SCADA server via phasor data concentrators (PDCs). Compared to SCADA measurements, which are usually updated every 3-5 seconds, PMU measurements are more accurate and can deliver up to 50 measurements per second [9].

1.1.5 Graphics Processing Units

The especial architecture of the graphic processing units (GPUs) made it a successful accelerator/processor in particular applications where a large amount of computations is required to be performed on data-parallel structure elements. A data-parallel application consists of large streams of data elements in the form of matrices and vectors that identical computation codes (kernels) are applied to them [10].

1.1.6 Online Simulation

While dealing with power system state estimation, online simulation implies that the results of state estimation should be available at specific time intervals, sufficiently fast enough to track the system dynamics. In the context of DSE, online simulation means that the results are available before the next set of measurements arrives. Currently the update rate is every 30-60 seconds which is not fast enough to track the dynamic behaviour of the system.

1.2 Literature Review

This section will review the milestones and previous studies in areas that must be focused on this research.

1.2.1 Static State Estimation

Since power system SSE was formulated by Schweppe in 1969 [11–13], it has remained an extremely active and contentious area. Of the many criteria used to develop a robust state estimator, the following three are regarded as the most common:

- Maximum Likelihood: maximizes the probability that the estimated state variable is near the true value.
- Weighted Least-Squares (WLS): minimizes the sum of the squared weighted residuals between the estimated and actual measurements.
- Minimum Variance: minimizes the expected value of the sum of the squared residuals between components of the estimated state variable and the true state variable.

Schweppe used the WLS algorithm which has been the most popular method and fundamental for all other algorithms. He constructed an objective function based one the weighted sum of squares of the measurement residuals to be minimized.

The growing size and complexity of the power systems motivated researchers to investigate faster and numerically more stable methods. After the advent of various algorithmic

enhancements such as parallel computation, distributed processing, hierarchical methods and etc., variety of techniques were examined to apply these methods in the state estimation algorithm for better computational speed [14, 15].

In hierarchical state estimation, the power system is split into smaller interconnected observable areas. Each area has its own state estimator which estimates the states locally. A global estimator coordinates the local estimates considering the correlations among subsystems due to tie-line measurements [16,17]. The main drawback of this kind of estimator is the communication overhead between subsystems and the delay caused by coordination stage. Besides that, all of the subsystems need to be observable which may not be feasible in real time applications. In Chapter 4 the proposed domain decomposition and relaxation method reduced the communication between subsystems by decomposing the system into fully independent subsystems using coherency characteristic of the generators. Also, since the system of equations are decomposed instead of the physical system there is no need for observability of all subsystems.

In decoupled state estimation, the interconnection between different subsystems was neglected which resulted in a simpler model [18, 19]. This method requires less memory and saves a significant fraction of the computations. However, it has the same problems associated with hierarchical sate estimation. These problems motivated researchers toward distributed state estimation [20, 21]. In this approach state estimators are physically distributed across the subsystems. Each substation performs its own state estimation without transferring data to control centre. Even in distributed state estimation it is necessary to accelerate state estimation for online monitoring of the system. In this research different types of state estimation are implemented on the massively parallel architecture of the GPU to reduce the overall execution time.

In addition to intelligent methods, parallel programming techniques have also been exploited to accelerate the state estimation. Parallel execution deals with the running of the estimation process simultaneously in different subsystems which significantly speed up the computational process [22–24]. In the above techniques, the state vector for each instant of time is obtained from the measurement set at the same instant, which usually cannot follow the dynamic behaviour of the power system. In Chapter 3 using EKF and the state prediction technique we capture the dynamic nature of the power system. Also, fine-grained parallelization offered by the GPU significantly reduced the execution time of our proposed method compared with parallel programming technique on CPU.

1.2.2 Dynamic State Estimation

DSE possesses the ability to predict the system states in advance and within a short time interval. By modeling the time varying nature of the system DSE alleviates losses under drastic changes during load fluctuations or network switching. The idea of using DSE was examined shortly after SSE [27]. Many of DSE techniques follow a prediction correction method which gives the operator a rough estimate of the states one step ahead of the time.

Existing DSE paradigms mainly focus on complexity reduction using partial measurements, hierarchical and decoupled methods [28–30]. These approaches compromise the accuracy for speed, and are not fast enough to predict the real-time behavior of the system. Other methods that focus on estimation accuracy by increasing either modeling or algorithmic complexity are computationally onerous limiting their practical applicability to small scale systems [31–34]. For nonlinear systems, subject to Gaussian noise, EKF is the most popular method which is computationally very demanding specially combined with the complexity of detailed power system component model [35,36]. To alleviate computational burden one approach tried reduce the computational complexity by describing an equivalent reduced order state-space system with lower dimensional measurement data [37, 38]. Utilizing the massively parallel architecture of the GPU in this work we accelerate state estimation without reducing the complexity of the model. In order to be able to implement the proposed method for realistic application we tried to apply methods which are applicable for large-scale systems and are suitable for SIMD architecture of the GPU to reduce the computational burden.

Most of the approaches dealing with DSE tried to improve the computational performance of the steady-state estimation process [34, 39–41] which only provides a series of snapshots of system conditions, where the dynamic transition between the snapshots is overlooked. However, few researchers focused on the dynamic parameters of the synchronous generator which plays a vital role in a power system [31, 33, 42–44]. Detailed representation of the synchronous generator in online DSE would allow system operators to accurately assess system condition and take rapid control actions following major disturbances.

There is limited research on the state estimation of the synchronous generator. An unscented Kalman filter based state estimation method for third-order generator model assuming the rotor angle as a measurable signal was represented in [45–47]. Also proposed in [33] is DSE with unknown inputs for fourth-order generator model considering the exciter output voltage as unknown input. Using the same generator model, [47] proposed an optimal state estimation of both generator internal dynamic and algebraic states. In all of the previous works the number of synchronous generators in the state estimation

process was limited to a single generator which precluded the study of computational performance of large-scale systems. In Chapter 4 we used a ninth-order generator model for DSE. Furthermore, instead of a single generator, DSE was simulated for all generators in the network.

1.2.3 Application of PMUs in State Estimation

PMUs provide measurements by sampling instantaneous waveforms can deliver up to 50/60 measurements per second. The implementation of synchronized measurement technology in the modern power networks has improved the capability of system operators to monitor the real-time condition of power systems [48, 49].

For large-scale power networks, installing enough PMUs for full network observation may be impractical. Several state estimation methods that incorporated only PMU measurements are discussed in [33, 50–52]. A more realistic and feasible deployment of the PMUs is achievable using a hybrid estimator, in which both conventional and synchronized measurements are included in the algorithm [53, 54]. Some approaches used mixed measurements by transforming the PMU to traditional format which may cause ill-conditioning in case of poor transformation or neglecting the time synchronisation [55, 56]. Others proposed a hierarchical scheme which used PMU and SCADA in different levels [57, 58]. However, none of these methods can track system dynamics during fault or sudden change in load.

It should be considered that the PMU and SCADA measurements are different in terms of accuracy, refresh rate and transmission delay. The differences in measurement accuracy affect data compatibility. Data refresh rate and transmission delay can be combined into a unified time synchronization issue [59]. To overcome above mentioning drawbacks, in [61] authors interpolated the nodal voltage of PMU unobservable buses. They considered a sensitivity matrix added to admittance matrix in case of significant changes in PMU measurements. However, this assumption may not be always true since not all buses in the network fallow the same changes. Another approach decoupled the system into PMU observable areas and SCADA observable area to estimate the states locally [28, 62]. This type of estimator requires the system to be completely observable through PMU which is not practically possible. In Chapter 3 we proposed a new data collation technique to combine the measurement input from both PMU and SCADA. Considering different accuracy for each set of the measurements, extrapolation of the SCADA measurements for time synchronization, and transformation of PMU data from polar to cartesian format provides a uniform measurement set for online DSE.

1.2.4 Smart Grids

Smart grids are large distributed systems which utilize enhanced information and communication technologies coupled with advanced control algorithms to improve efficiency and reliability of the power system [63]. With the addition of communication and smart metering technologies, state estimation is now feasible at all levels of the power system. However, new generations of digital devices, such as PMUs, distributed generation and smart meters provide huge amount of additional information which increase the computational burden and significant data storage resulting in new challenges [64].

Considering complexity associated with the large-scale estimation problems, researchers proposed network reduction [65], and domain decomposition [66, 67] to scale down the problem. A survey of multi-area and distributed state estimation can be found in [68]. However, as the number of measurements and sampling rate increase, these approaches may suffer from communication bottleneck and computational reliability issues inherent in system architecture.

To alleviate the drawbacks of above methods, distributed state estimation approach was investigated [64,69,70] which, both reduces the size of the problem and does not suffer from long latency caused by communication between different levels. Recent research in power system state estimation within the smart grid context is focused on the following subjects:

- Distributed state estimation by domain decomposition to scale down the problem [65, 67, 70].
- Combined parameter and dynamic state estimation on practical data [42].
- Vulnerability analysis of power grids as a cardinality minimization problem [71].
- Secure state estimation by identification and protection of critical measurements [72, 73].

While the above methods address important issues, they overlook the computational efficiency aspect and speed of the state estimation process which are vital for online monitoring and control of the grid. In addition, the proposed approaches are mainly tested on CPU-based hardware for small system sizes which behave somewhat differently from large-scale power systems in terms of computational complexity. In this work parallel implementation of DSE on GPU significantly reduces the computational burden. The proposed methods are tested on large-scale systems which guarantee its efficiency for realistic applications.

1.2.5 Cyber-Physical Attacks on State Estimation

Since the beginning of state estimation development, it was necessary to validate the measurements obtained from SCADA, as they are known to contain measurements with various types of errors. Many researchers have considered the problem of bad data detection (BDD) in the power systems [17,75–78], however conventional BDD approaches usually fail when the network malfunction is intentionally caused by an attacker [4,79,80].

Although, the advancement of cyber technologies in sensing, communication and smart measurement devices significantly enhanced the power system security and reliability, it increased the vulnerability of smart grids to cyber-attack [81,82]. Coordinated cyber-attacks can be designed to be unobservable for BDD algorithms. Such attacks may impose significant errors in the state estimation algorithms, and mislead system operators into making potentially catastrophic decisions [83]. Vulnerabilities of power system to cyber-attacks can be classified into three main categories [84]:

- *Data integrity analysis*- this research area investigates the possibility of the attack from attacker's point of view by exploiting weaknesses in BDD techniques [85–87].
- *Consequence analysis*- the effect of false data attack within different functions of energy management systems such as optimal power flow, congestion analysis, automatic generation control and energy pricing is investigated in this research area [88–90].
- *Attack prevention analysis*- specifically this research area is interested in finding the critical measurements and protecting them by improving the security of the communication system [91–94].

One important fact which is neglected in above works is that the cyber-security analysis should be performed in a timely manner in order to efficiently solve the data attack construction problem. Otherwise it will slow down the process of state estimation and control of the system behaviour. Another main concern related to most of the above approaches is that they are not tested on large-scale power systems. So the complexity and efficiency of the proposed approaches in practical system is unclear.

In Chapter 5, considering the fact that the power system is a nonlinear system which is subjected to many random events, a method based on the stochastic nature of the power system is proposed. The power system can be best modeled as a stochastic hybrid dynamical system where the stochastic availability of generation and state/structure interaction is explicitly included [95,96]. Using Markov chain theory and Euclidian distance we proposed a detection technique which can reduce the chance of a successful cyber-attack. The proposed method is implemented on the GPU and tested on large-scale systems.

1.2.6 Relaxation Methods

Relaxation approach is not restricted to a particular system structure. In this method the system is partitioned into a number of subsystems to reduce the computational complexity of the system solution. Each subsystem uses the previous results of other subsystems as an initial guesses for its new iteration. After each iteration results are exchanged between subsystems, and this process is repeated until convergence [98].

The relaxation method was first introduced in the power system area in [100]. In [101] the simulation time between the sequential relaxation method and direct method has been compared for some study cases. The useful outcome resulted from both sequential [101] and parallel [102] implementation of the relaxation method is getting a higher efficiency for the larger systems. Applying the Gauss-Seidel relaxation technique to the large linear matrix solutions confirmed that the present method can reduce the CPU cost without losing the numerical accuracy and stability [103]. The application of relaxation for cyber-attack analysis [104] and for real-time transient stability simulation of power systems [105] was also investigated. In chapter 4, this method is employed for parallel DSE on GPU.

1.2.7 Domain Decomposition

Domain decomposition techniques are primarily partitioning methods which try to split a system into several subsystems that can be solved individually [106, 107]. The main advantage of decomposition techniques is that they are suitable for parallel application on multi-processors since independent subsystems can be solved simultaneously.

Generally, from a power system point of view, domain decomposition methods reduce the problem size by dividing the network into several sub networks which results in less computation effort in each individual sub network. Most of the approaches in distributed state estimation partitioned the system either randomly or based on the geographical distances by assigning equal numbers of generators and buses among the subsystems. Most of these approaches ignore overlapping and boundary buses which is not an efficient method because the network buses have different connectivity which leads to load balancing problem and inaccuracy [70, 109, 110].

Other methods which tried to overcome the load balancing problem based on graph theory approaches are mainly too complicated for online applications [111–113]. Another option would be to split the computation burden among processors based on the total number of equations. However, this approach will increase both the programming and communication complexity [22, 24].

In the proposed decomposition technique the combination of coherency characteristic with equal computational load balancing in Chapter 4 increased the efficiency of the parallel programming while reducing the execution time with no changes in the complexity of the system.

1.3 Motivation of this work

Conventional methods to improve the cycle time for state estimation mainly include the following strategies:

- Complexity reduction
- Hierarchical state estimation
- Distributed state estimation
- Parallel state estimation

Although above approaches tried to improve the process of state estimation, they all have their own drawbacks which are discussed in literature review. Moreover, in almost all of the aforementioned works, DSE has been performed for either a single machine or a small power system. However, in large-scale power systems with detailed modeling of grid components, a key issue is the requirement for high-performance computation resources. It was one of the major motivation in this work to look for faster techniques for DSE in Chapter 3 and Chapter 4. For online monitoring of the power system, we need to preserve accuracy and speed simultaneously to prevent catastrophic events like blackouts.

From computing source point-of-view, supercomputers, multiprocessor networks, various types of parallel processing architectures, and distributed memory have already been employed for power system analysis. All of these approaches accelerate the simulation process to some extent; however, they are limited by some important factors such as cost, system size programmability, and communication issues. Advancements in GPU technology along with the increasing deployment of high-speed time-synchronized PMUs in wide areas provide the opportunity for online monitoring of the power system which was another important motivation of this research. The GPU offers significant speed-up with the lowest possible cost on properly designed parallel algorithm. Moreover, it can be used in existing power network with minimum changes in the system.

In addition, historically researchers focused on the effect of faults on the physical infrastructure; however, faults in the cyber components may cause even more damages than the physical failures. Cyber attacks are intelligently designed to manipulate the operator so that they result in irreparable damages in the system which, are not comparable with

random bad data. Therefore, new solutions must be developed to assess sources of vulnerabilities and detect cyber-attacks. The primary effect of including cyber-attacks in the system modeling is that it increases the size of the system under study. As a result, faster algorithms need to be investigated to guarantee online system performance by detecting the attack in the smallest possible time which gives the operator enough time to take preventive actions. Utilizing the GPU and considering the stochastic nature of the power system in Markov chain modeling the proposed method in Chapter 5 reduces the chance of successful cyber-attack in the power system.

Overall, the desire to accelerate the state estimation process for large-scale power systems is the main motivation of this thesis. The speed of state estimation can be improved by three approaches:

- Developing new algorithmic methods
- Exploiting parallel and distributed processors
- Utilizing faster processors

This thesis aggregates all three aforementioned approaches to accelerate the state estimation process for large-scale power systems. A novel algorithmic method is proposed and implemented on two different types of processors. One is a general purpose CPU, and the other is the massively parallel GPU. GPU offers a viable alternative computational engine for speeding up online simulators for endless growing complexity and increasing accuracy demand of a system model.

1.4 Thesis Objectives

The objective of the proposed research is to develop parallel algorithms for dynamic state estimation of large-scale power systems. Utilizing the massively parallel architecture of GPUs, the proposed approach is implemented in both single-GPU and multi-GPU computing systems. Main contributions of this study are as follows:

- Two-level state estimation
  In order to capture the dynamic nature of the power systems, a WLS state estimation method applying EKF on PMU measurements is proposed. Due to the high cost of PMU synchronization, it is not possible to have them all over the grid, so a hybrid of PMU and SCADA measurements is used. The SCADA measurements in the buses without PMU installation are calculated based on the historical data considering the delay in the PMU measurements to take advantages of the PMU's faster refresh rate. Using EKF the entire states are predicted one step ahead of the time. The PMU and

estimated SCADA measurements are used for state correction. In normal operation condition the exponential moving average method is used to estimate the PMU measurements. To consider the effect of load changes, in case of significant difference in PMU measurements, a correction term is added to the results of prediction. The time synchronization, which is the most important reason for ill-conditioning, is guaranteed since the SCADA measurements are estimated at the same time as the PMU measurements are sent.

- Massively parallel implementation on GPUs
  In this research, we do not compromise on the system model complexity. GPU thread-level parallelism is used for large-scale power system state estimation for the purpose of online monitoring and control. Taking advantages of the massively parallel architecture of the GPU, computationally intensive sections of the state estimation program converted into a CUDA kernel (functional program which generates a large number of threads for data parallelism).
  Via thousands of threads the entire task executed in parallel, which significantly accelerates the process of state estimation. Three types of parallelism are used in this research; algorithm-parallelism, task-parallelism and data parallelism. The first one find the parallelism inherent in the problem. In the second method the traditional serial algorithm is converted into various smaller and independent tasks which may be solved in parallel. The last method is the most fine-grained type of parallelism that can be used on the SIMD-based architecture of the GPUs.

- Numerical solution methods
  The numerical methods for parallel computations is an important subject for a better system performance. Both direct and iterative sparse linear solvers are exploited. The LU decomposition and Conjugate Gradient solver are applied for state estimation using static states. Relaxation methods is employed for the nonlinear system of dynamic states using detailed generator model. Relaxation based solver is an iterative method whose underlying structure is parallelized in nature. Therefore, each subsystem is solved independently to fit in the parallel hardware architecture of the GPU.

- System partitioning
  Another important factor for exploiting parallelism is to partition the large system into smaller subsystems. Partitioning can be done in two different ways. The problem can be divided into many small tasks to be solved independently which result in intensive the communication between the processors. Or it can be split into a few

but large tasks or subsystems which impose less communication on the processors. In our research, a combination of both approach is employed using the thread-level parallelism on GPU and applying the relaxation technique. For partitioning a load balancing technique considering coherency characteristic of the generator is used to distributed equal load work among processors.

- Cyber-attack analysis
  To overcome the effect of cyber-attacks, the stochastic nature of the system disturbances is considered and a cyber-physical model of the power system utilizing the Markov chain is proposed. Considering the previous results of state estimation, a set of possible states along with the probability of each state is provided. A Markov chain based on these states is defined. After each estimation process all states are checked on the Markov chain. If the estimated state is close to a value with low probability or out of the Markov chain, the possibility of the cyber-attack is high. In this situation, critical measurements are identified and updated based on the on-line power flow analysis. The state estimation run again using updated measurements. Changing the critical measurements using updated information decreases the chance of successful cyber-attack.

1.5 Thesis Outline

This thesis consists of six chapters and is organized as follows:

- **Chapter 1: Introduction** - The general terms used in this thesis are described in this chapter to highlight the scope of the research. The background work done in this area since several decades ago is also summarized.
- **Chapter 2: Overview of Multi-Core and Many-Core Architectures** - This chapter presents an introduction about CPU and GPU technology and its architecture.
- **Chapter 3: Massively Parallel Static/Dynamic State Estimation: Single GPU Implementation:** - This chapter presets the results of parallel state estimation for both static and dynamic states implemented in single GPU architecture. By exploiting the parallelism inherent in the state estimation problem, a parallel algorithm is devised to maximize the computational efficiency of the GPU.
- **Chapter 4: Relaxation-based Dynamic State Estimation: Multi-GPU Implementation** - The Relaxation method is introduced and implemented in this chapter. The objective is to revisit the application of relaxation-based approaches to the state estimation problem using coherency based domain decomposition.

- **Chapter 5: Robust Dynamic State Estimation Against Cyber-attack** - Considering the stochastic nature of the power system and using Markov chain model a robust state estimation against false data injection attack is presented in this chapter.
- **Chapter 6: Conclusions and Future Works** - The contribution of this research and the future works are summarized in this chapter.

2

Overview of Multi-Core and Many-Core Architectures

2.1 Introduction

A sequential computer with one central processing unit (CPU) includes only one control instruction unit. Apart from its limitation to single instruction execution at any time, there are two main obstacles with this technology: slow memory access and fundamental limitations such as overheating with compact circuits. These issues limited the achievable speed of serial computers even with the growth of the hardware technology. Therefore, the parallel processing techniques were seriously taken into account as the main alternative approach.

Recently, GPU which was originally developed for video game programming has become programmable as a general purpose programming platform. Due to the low cost and huge computational power of the recent GPUs, they are getting a lot of attention in different type of power system analysis. GPU-based techniques as an alternative for CPU-based parallel programming have a major role in acceleration of simulations which need highly intensive computational power.

The application of parallel processing in power system analysis is motivated by the desire for faster computation. The popularity of the GPUs in the field of high performance computing is due to their ability to provide computational power for massively parallel problems at a reduced cost.

2.2 Parallel Processing

Parallel processing is an information processing technique in which two or more processors work simultaneously on the solution of a problem. The application of parallel processing in power systems analysis is motivated by the desire for faster computation and not because of the structure of problems. Parallel processing techniques attracted considerable research interest in different types of power system computations such as state estimation [22, 24], power flow analysis [114–116] and transient stability [117, 118].

2.2.1 Massively Parallel Processing on the GPU

Parallel processing is a type of computation where many calculations are performed simultaneously. This method of computation is based on the fact that large problems can be divided into smaller pieces and then solved concurrently. In parallel processing the single CPU is replaced by multiple CPUs (even if they are individually slower than the presumed single CPU) whose overall parallel performance accelerates the simulation. Fig. 2.1 shows the four main steps in creating a parallel program which includes: 1) decomposition of computation in tasks, 2) Assigning tasks to processors, 3) arrangement of data access, communication and synchronization, and 4) mapping processes to processors.

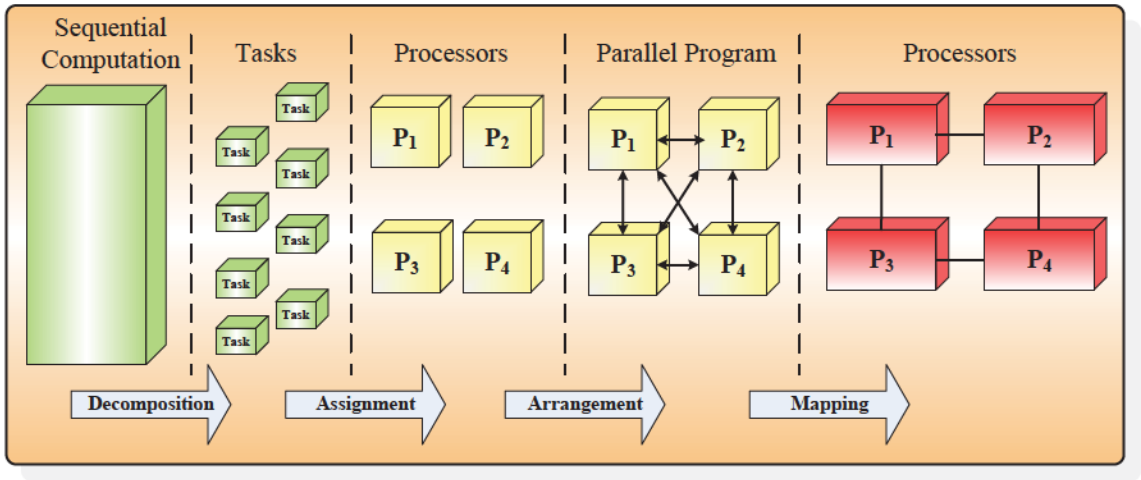


Figure 2.1: Steps of parallel algorithm generation.

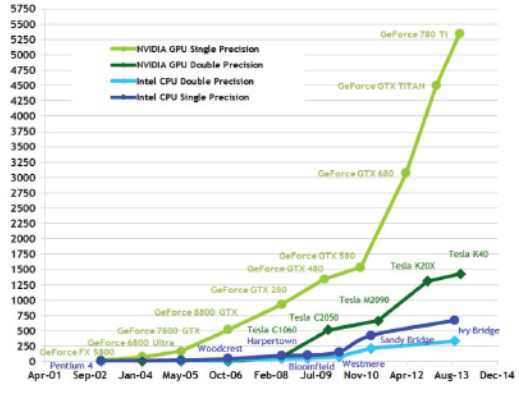
Chronologically, there are two main classification for parallel processing architecture hardware. In the first one computing machines are characterized by the number of simultaneously active instruction and data streams which practically grouped as single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) architectures [119]. In a SIMD-based technology the parallelism is exploited by performing the same operation

concurrently on many pieces of data, while in the MIMD architecture different operations may be performed simultaneously on many pieces of data. The SIMD model works best on a certain set of problems such as image processing, and MIMD is suitable for general purpose computation. Vector processors and array processors are examples of the SIMD-based architecture, multi-processor and PC clusters have an MIMD architecture.

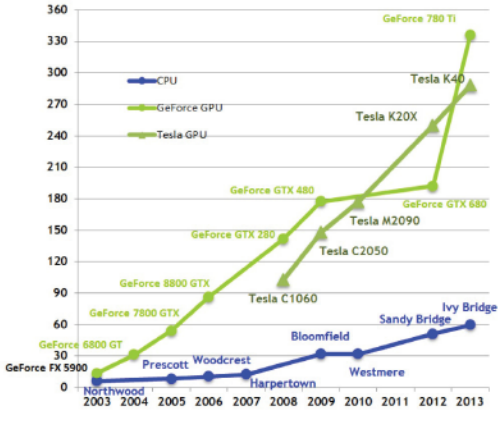
In the other category the focus is on the relationship between processing elements and memory modules [120]. There are two classes of parallel processing architectures: distributed memory, and shared memory. In the former, there is no memory in the system other than the local memory on each processing element, and the processors communicate with each other by sending and receiving messages in a network. In the shared memory processors, however there is a central memory accessible from any of the processing units, regardless of existing local memory on each processing units. The common memory is used to make communications between processors in shared memory architecture.

2.3 CPU and GPU Architecture

There exist two major processor architectures: the multi-core CPU and the many cores GPU. The CPU, which is the processor in most of the computers, is composed of only a few cores that can handle a few software threads at a time. In contrast, the GPU which is an energy-efficient processor on the market is composed of hundreds of cores known as stream processors (SP) that can simultaneously handle thousands of threads. Fig. 2.2 shows a comparison between the GPU and CPU computation capabilities and bandwidth, where it can be seen that, as of 2011, the peak performance and bandwidth of the GPU is about four times the performance and bandwidth of an Intel CPU [121].



(a) Floating-point operation per second for the CPU and the GPU.



(b) Memory bandwidth for the CPU and the GPU.

Figure 2.2: Comparison between CPU and GPU [121].

The modern GPU consists of multiprocessors which map the data elements to the parallel processing threads. A multi-core CPU has 6 to 7 times larger cache than the GPUs cache system. On the other hand, the GPU has many more cores than the CPU. Owing to the fact that more transistors are devoted to data processing than caching and flow control, the GPU has significantly larger computational power compared to a multi-core CPU [122]. In order to highlight the differences between CPU and GPU, Fig. 2.3 is provided which shows the physical and abstracted resources in a CPU and a GPU. CUDA^{TM} and OpenMP® which are used for parallel programming will be explained later.

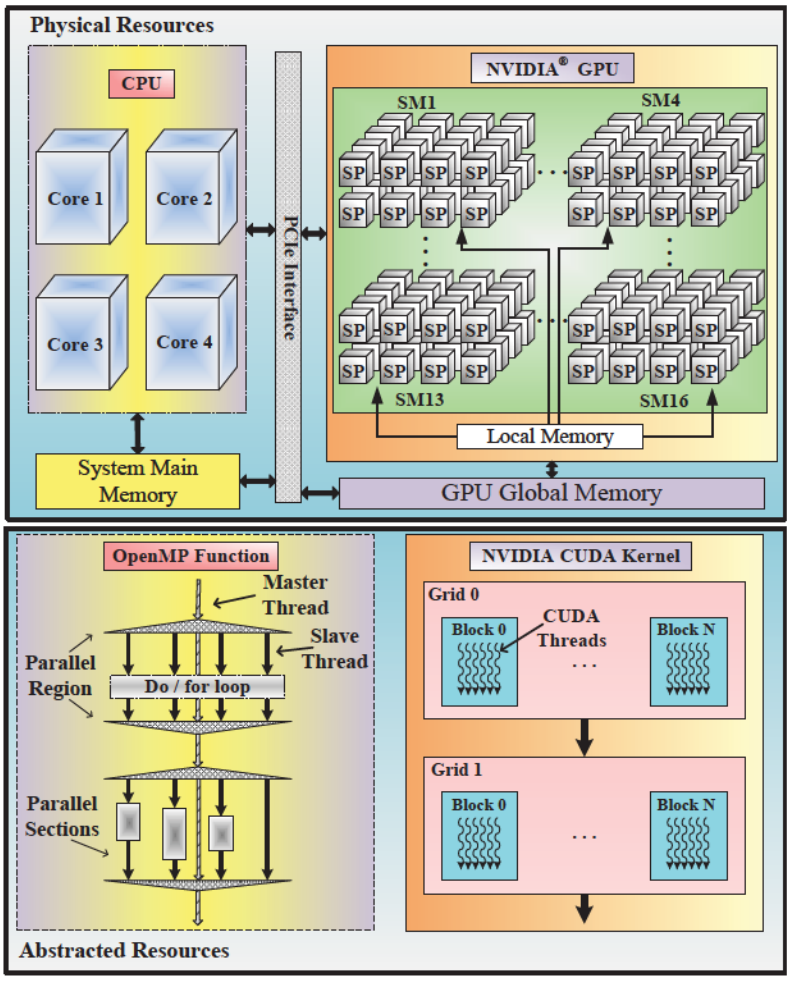


Figure 2.3: CPU, GPU, CUDA^{TM} and OpenMP® resources.

2.3.1 Multi-Core CPU Architecture

The term multi-core refers to a multiple core processor that is simply an integrated circuit where two or more processors have been attached for increased performance via parallel processing. The multiprocessor creates, manages, and executes concurrent threads in hard-

ware with zero scheduling overhead. To manage hundreds of threads the multiprocessors map each thread to one scaler processor core, and each scaler thread executes independently with its own instruction. There is a global device memory that all the multiprocessors can have access to. Also, each multiprocessor has its own on-chip memory that is accessible individually.

2.3.2 OpenMP

OpenMP® is a standard application programming interface (API) for multi-core CPUs which does not require major code reformation for parallelization [123]. It supports shared memory, however it is limited to a couple hundred core due to thread management overheads and cache coherence hardware requirements .

OpenMP® also includes directives, library routines, and environment variables to facilitate scheduling and parallelism at runtime with high level of portability. It provide support for sharing and privatizing data using extension of the C and C++ languages with tasking constructs, device constructs, work sharing constructs, and synchronization constructs. The program begins as a single process called a master thread which executes sequentially. The master thread creates a group of slave threads within the parallel construct (Fig. 2.3). At the end of the construct only the master thread remains while the rest of the slave threads synchronize and terminate.

Simple Example

An example is provided here to illustrate how an algorithm may be parallelized in a shared memory programming model using OpenMP®. Consider a function that takes two N $\times$ N matrices A and B and multiply them in a third matrix C. On the CPU, three *for* loops are used over all array elements as follows:

$$
\begin{aligned}
&for\ \ (i = 1\ :\ N)\\
&\quad for\ \ (j = 1\ :\ N)\\
&\qquad for\ \ (k = 1\ :\ N)\\
&\qquad\quad C[i][j] = A[i][k] * B[k][j] + C[i][j]\\
&\qquad end\\
&\quad end\\
&end
\end{aligned}
$$

There are two level of active computation in each loop: outside the loop and inside the loop. Outside the loop, the loop counter is increasing and compared with the length of N, while inside the loop, the actual computation is performed on arrays at a fixed position determined by the loop counter. The calculations performed on each data element in the matrices are independent of each other.

The OpenMP® version of the same function can be written as:

$$
\begin{aligned}
&\#pragma\ omp\ parallel\ num\_thread(n)\\
&\quad \#pragma\ omp\ parallel\ for\ private\ (j,k), shared(--)\\
&\qquad for\ (i=0;\ i<N,\ i++)\{\\
&\qquad\quad for\ (j=0;\ j<N,\ j++)\{\\
&\qquad\qquad sum=0;\\
&\qquad\qquad \#pragma\ omp\ parallel\ reduction\ (+:sum)\\
&\qquad\qquad for\ (k=0;\ k<N,\ k++)\{\\
&\qquad\qquad\quad sum=A[i][k]*B[k][j]+sum;\\
&\qquad\qquad \}\\
&\qquad\qquad C[i][j]=sum;\\
&\qquad\quad \}\\
&\qquad \}
\end{aligned}
$$

Using *parallel* construct a team of n threads which are defined by $num\_thread(n)$ is formed to start parallel execution of the corresponding code. $num\_thread(n)$ is an internal control variable which defined the maximum number of threads in the team executing the parallel region. Appropriate iterations are assigned to the individual threads by compiler. The data environment consists of:

- *Private variable*- which determine private variables in the thread executing region.
- *Shared variable*- which determine variables that are shared among the team of threads executing the parallel region.
- *Reduction variable*- which identifies which shared value and operation will be used.

This model of execution is referred to master/slave model where multiple threads of execution perform tasks defined implicitly or explicitly by OpenMP® directives. Given a sequential program, the master/slave makes it easy to get loop level parallelism in an incremental fashion which take advantages of a multiprocessor system.

2.3.3 Many-Core GPU Architecture

The GPU thrives on data-parallel applications with large computational requirement. A data-parallel application consists of large streams of data elements in the form of matrices and vectors that run the same computation code. To efficiently use and program a GPU it is instructive to learn the GPUs internal architecture. The following section briefly explains the architecture and hardware specifications that we require for programming purposes.

2.3.4 CUDA Program Structure

The first general-purpose programming model for the GPU hardware was compute unified device architecture (CUDA) which provide a C-like syntax to execute and manage computations on the GPU as a data-parallel computing device. A CUDA program consists of multiple phases that are executed on either the CPU (host) or the GPU (device). The sequential parts are implemented on the host, and intensive parallel phases are performed on the device.

The executable code which runs on the device is known as a CUDA kernel. Kernels are functions designed to run in parallel on multiple streaming multiprocessors (SMs) of the GPU. The GPU runs its own kernel independently under the CPU's control. The execution starts with the host and moves to the device after a kernel function is invoked. All the kernels have their own unique coordinates to distinguish themselves, and to identify the specific portion of the data to process. Typically, calling a kernel create a large number of threads to execute the same task all in parallel [121].

2.3.5 CUDA Hierarchy

Threads are the smallest computing element in the GPU that can be scheduled to run on SPs. When a kernel is invoked, blocks of threads are defined to assign one thread to each data element. All threads associated with a particular kernel are allocated in a grid running the same instruction on one SM. The grid is subdivided into a set of thread blocks, where each block has the same number of threads.

Each block within a grid, and each thread within a block are identified by individual indices *blockId* and *threadId* that make them accessible via the built-in variables in CUDA. All blocks in a grid have the same dimensions and share the same *blockId* values. Grids can be one- or two-dimensional with a maximum of 65535 blocks in each dimension. Each block within the grid is composed of up to 1024 threads. These threads are basically general purpose arithmetic units, each having their own floating point unit (FPU) which can be organized in one, two, or three dimensions arrays with a maximum of 1024 threads in

the 1st and 2nd dimensions and a maximum of 64 threads in the 3rd dimension [124]. Fig. 2.4 shows a high level view of CUDA in a grid of four blocks.

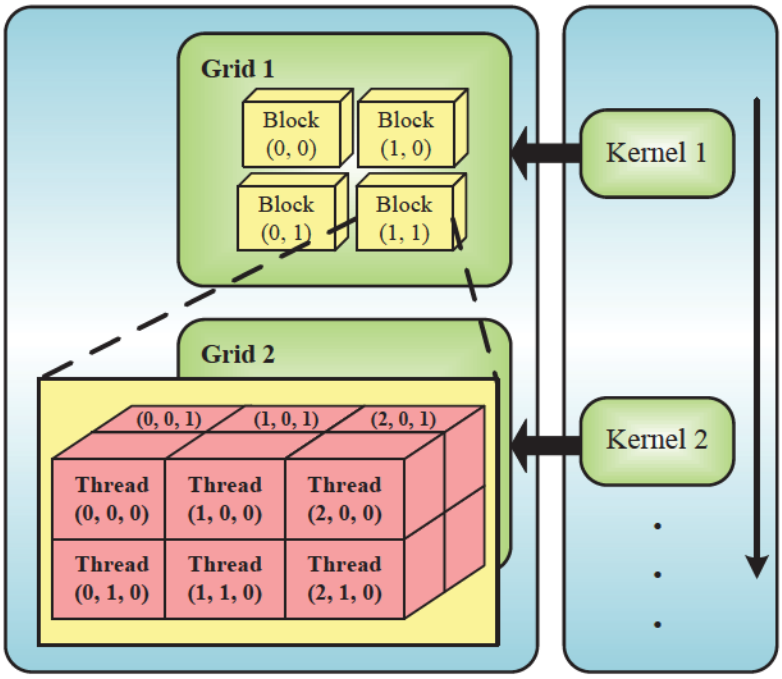


Figure 2.4: CUDA thread organization for data-parallel processing.

When a block of threads is assigned to a SM, the instruction unit splits and creates groups of 32 parallel threads called warps. Each of these warps contains the same number of threads, called the warp size, and is executed by the multiprocessor in a SIMD fashion. Because threads in a given block execute within the same SM, they have access to the shared memory in the SM, as well as to a global memory in the GPU.

To ensure all threads in a block have completed a stage of their execution, threads have to stop until every thread in the block reached the same point in the set of kernel instructions before continuing with execution. Excessive synchronization calls can result in wasted time where the SPs are waiting rather than working. Once all the threads have finished executing, the host can copy results from GPU memory to host memory and begin execution of a different kernel.

Simple GPU Kernel

In contrast with CPU which executes one instruction at a time, and each instruction does one thing, GPU can executes several instructions at the same time. To illustrate how the CUDA works, consider the same example provided in Section 2.3.2.

The computation on the GPU is performed by separating the outer loop from the inner calculations. First of all, enough memory space on device memory should be allocated for each matrix using $CudaMalloc$ commands:

$$CudaMalloc((void**)\&d_A, (sizeof(float) * N * N));$$
$$CudaMalloc((void**)\&d_B, (sizeof(float) * N * N));$$

$$CudaMalloc((void**)\&d_C, (sizeof(float) * N * N));$$

d_A and d_B specifies the location of the matrix A and B in device memory. The size of allocated memory is defined by measuring the size of the variable (float in our case) and multiplying it by the size of matrix A.

The next step is to transfer data to the GPU by executing the following command:

$$cudaMemcpy(d_A, h_A, (sizeof(float) * N * N), cudaMemcpyHostToDevice);$$
$$cudaMemcpy(d_B, h_B, (sizeof(float) * N * N), cudaMemcpyHostToDevice);$$

h_A and h_B specify the location of matrix on host memory. Arrays of the matrices will be stored in vector format in the GPU. The kernel code to perform the operation can be written as:

$$\_\_global\_\_\ void\ MatrixMultiplication(float\ *A, float\ *B, float\ *C, int\ N)$$
$$\{$$
$$float\ \ sum = 0;$$
$$int\ id.x = blockIdx.x * blockDim.x + threadIdx.x;$$
$$int\ id.y = blockIdx.y * blockDim.y + threadIdx.y;$$
$$if(id.x < N\ ||\ id.y < N)$$
$$for(int\ i = 0;\ i < N;\ ++i)$$
$$sum+ = A[id.y * N + i] + B[i * N + id.x];$$
$$C[id.y * N + id.x] = sum;$$
$$\_\_syncthreads();$$
$$\}$$

The global qualifier $\_\_global\_\_$ specifies that the kernel is callable from the CPU and will be executed on the GPU. Instead of the frits two *for* loops new parameters, called $id.x$ and $id.y$, are defined to control the execution of the kernel. Each thread will perform on the

vector elements specified by $id.x$. $blockDim.x$ returns the number of threads in each block. Every thread in a block and every block in a grid has a unique index which is accessible through the $threadIdx$ and $blockIdx$, respectively. The next line performs the dot product between rows of first matrix and column of second one. Once the results are added up it will be saved on matrix C. The number of threads have to be equal or more than the number of elements in matrices to ensure that calculation is performed on all elements. The $\_syncthreads()$ call ensures that all threads are synchronized. To invoke this kernel from a CPU-based code we need to add a syntax as below:

$$MatrixMultiplication <<< grid, block >>> (d_A, d_B, d_c, N);$$

The grid dimensions and the block dimension in execution configuration (<<< >>>) are defined by *grid* and *block*, respectively. At the end the result can be transferred to host memory using the $cudaMemcpy$ command.

2.3.6 Memory Hierarchy

At the fundamental level, each of the threads has their own set of local memory and a set of registers. Threads in a block have collective access to shared memory which is local to that block. All the threads also, have access to a global memory which is the main memory of the GPU with the largest size and slowest access speed. Constant memory is a fast but read-only memory and texture memory is usually used for graphics rendering applications.

In addition, a CUDA also contains constant memory, which is very fast but read-only memory. Texture memory which is usually used for graphics rendering applications is also available. Finally, the GPU also has access to the host's main memory, but not direct access [121]. Fig. 2.5 shows the memory hierarchy in a grid of four blocks described above.

2.4 Hardware Setup

The hardware used in this work is one unit of Tesla^{TM}S2050 GPU from NVIDIA® with 148 GB/s memory bandwidth. Each Fermi^{TM} GPU inside Tesla^{TM}S2050 has 448 cores which deliver up to 515 Gigaflops of double-precision peak performance. This device contains 16 streaming multiprocessors (SMs), each with 32 streaming processors (SPs), an instruction unit, and on-chip memory. Each SM has 4 special function units (SFUs) execute transcendental instructions such as sin, cosine, and 16 load/store units, allowing source and destination addresses to be calculated for sixteen threads per clock [121]. Fig. 2.6 shows the

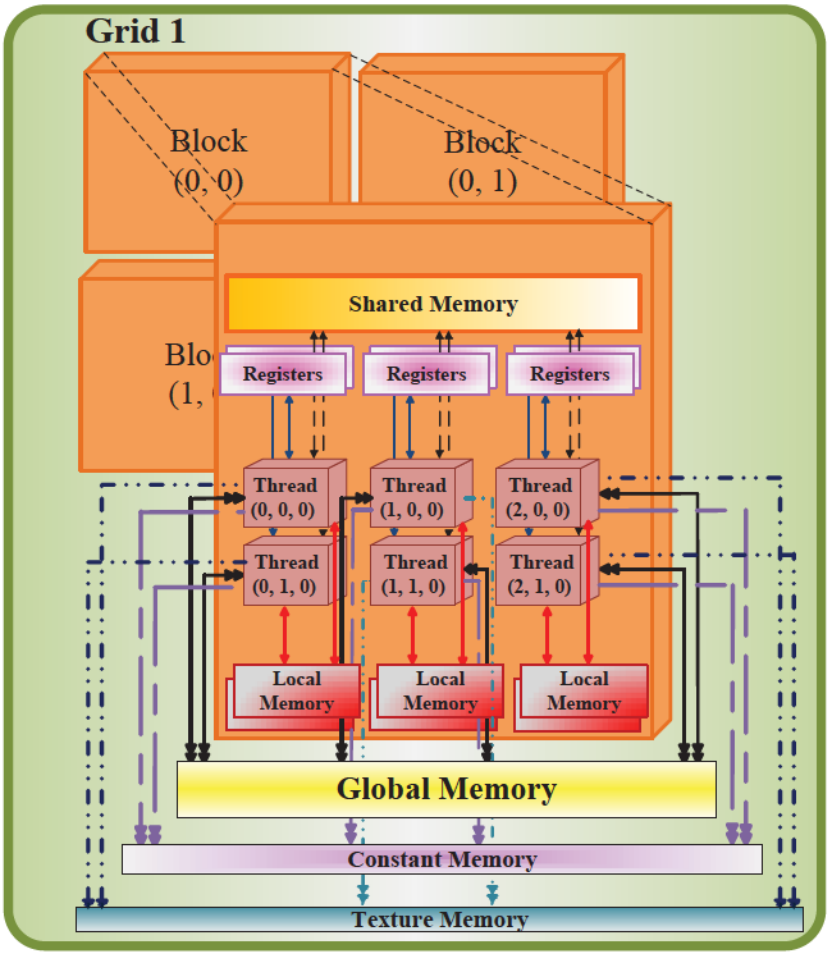


Figure 2.5: CUDA memory hierarchy and thread organization for data-parallel processing.

architecture of the Fermi^{TM} GPU.

The Tesla S2050 is a CUDA-enabled device which can execute multiple kernels (4) simultaneously. Thus to have four GPUs working in parallel, in a Tesla S2050 or any multi-GPU architecture, we require the same number of CPU cores to manage and control the GPUs simultaneously to minimizes the overhead that occurs in data copying and kernel invocation. The CPU is the quad-core Intel® Xeon^{TM} E5-2620 with 2.0 GHz core clock and 32 GB memory with 42.6 GB/s memory bandwidth, running 64-bit Windows 7® operating system. Fig. 2.7 shows the inside architecture of Tesla^{TM} S2050 computing system connected to CPU.

In this work, the C-run-time library and the Win32 API was used to have a full control on the synchronization of GPU data-transfer. CUDA version 5.0 with compute ca-

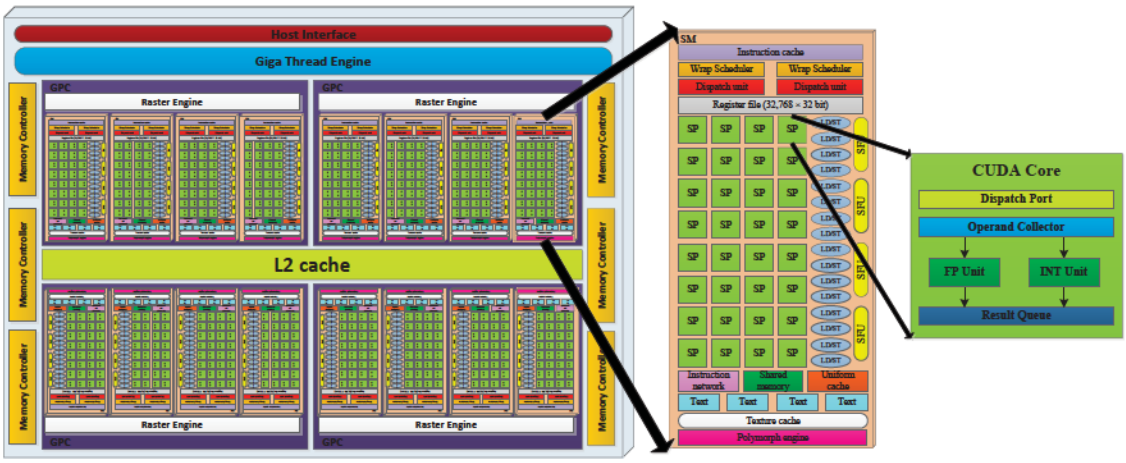


Figure 2.6: Fermi^{TM} GPU architecture.

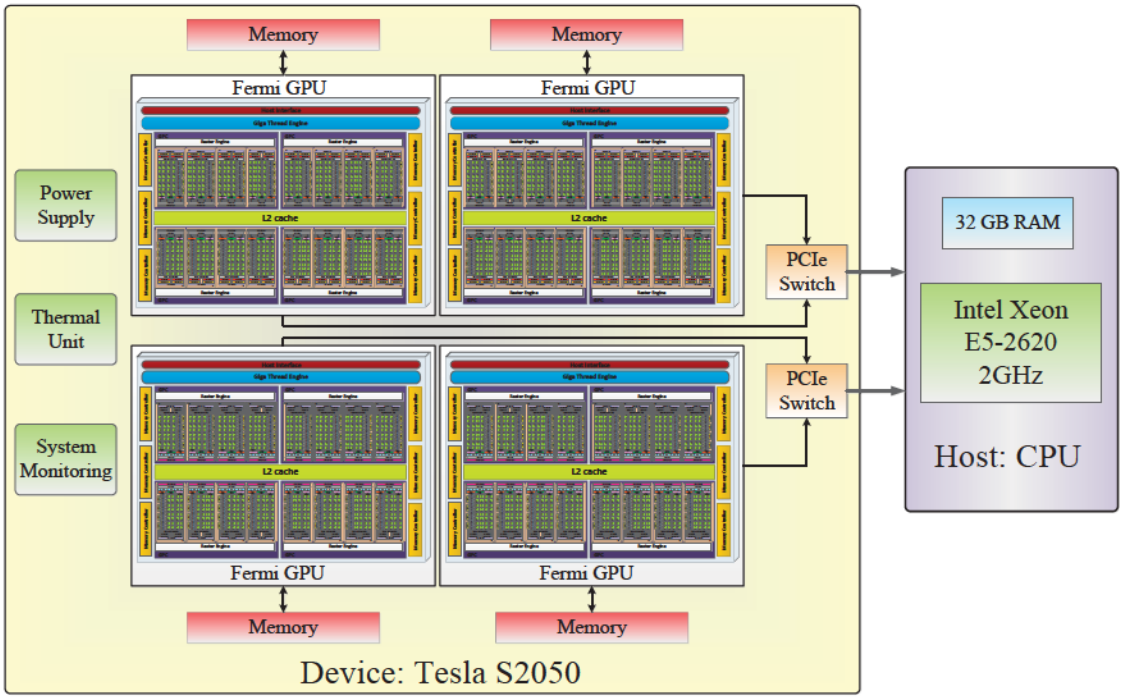


Figure 2.7: Tesla^{TM} S2050 computing system architecture.

pability 2.0 is used for programming. While a generic programming language such as OpenCL [125] could have been used to program GPU, CUDA's advantages in providing advanced debugging, better performance, and higher level of abstraction are the main reason for its adoption in this work.

2.5 Type of parallelism used in this work

Overall, three types of parallelism used in this thesis as follows:

- *Algorithm Parallelization*- this is a coarse-grain parallelism which happens at the first step and before any numerical method starts solving the system of equations. The main objective is to find the parallelism inherent in the overall algorithm. Implementation of relaxation method on state estimation problem is an example of algorithm parallelism.
- *Task Parallelization*- this type of parallelism the convert the traditional serial algorithm into several smaller independent tasks which can be solved concurrently. Decomposition of Gain matrix in state estimation problem into smaller matrices that can be solved in parallel falls under this type of parallelism. The space parallelism methods which will be explained later also falls into this category.
- *Data Parallelization*- this type of parallelism can be used on the SIMD architecture of GPU depends on the capability of the problem. This type of parallelism also called as fine-grained parallelism which can be used in both of the previous methods.

2.6 Discussion

It is shown that both CPU and GPU are capable of parallel programming. However, considering the SIMD architecture of the GPU it is a lot more efficient than CPU specially when the problem deals with huge amount of computation. Besides that the number of cores available in GPU's hardware are hundred times more than CPU which makes it significantly more powerful in fine-grained parallelism.

Based on Amdahl's law [129], even with many-core processors such as GPU, the maximum achievable speed-up utilizing parallel processing is limited by the time needed for the sequential part of the program. Gustafson's law which is a refined version of Amdahl's law argues that as the size of the problem increases, the inherently serial part of the program takes less portion in the overall problem. Since almost all the steps of our program is running on GPU, as the size of the system increases the parallel portion of GPU code expands faster than the serial portion.

Based on Gustafson's law if X is a non-parallel fraction of a program, the highest possible speed-up using N processors is given as [130]:

$$S_p = N - X(N-1), \tag{2.1}$$

X can be calculated using the measured speed-up (S_m) on a specific number of processors (N_p) using

$$X = \frac{S_m - N_p}{(1 - N_p)S_m} \tag{2.2}$$

Above law prove that parallelization using GPU is not limited and it is possible to achieve higher speed-up by a different programming structure and more cores unlike CPU which can offer a limited speed-u with parallelization.

2.7 Summary

This chapter presents the fundamentals of GPU architecture and parallel processing. The issues involved in CPU-based parallelization are shortly explained. Taking the advantage of parallel architecture of GPU, efficient parallel programming with high speed, improved data throughput, and optimized hardware resource utilization is possible.

Overall, GPU have the following advantages over CPU clusters:

- *Parallelism*- parallelization using GPU is fine grained parallelization which is a lot different from coarse grained parallelization on CPU. In contrast to the CPU with a limited number of arithmetic cores, the GPU is composed of hundreds of cores known as stream processors (SPs) that can simultaneously handle thousands of threads.
- *Extensibility*- unlike CPU with limited achievable speed-up, the maximum achievable speed-up by massive parallelism in GPU is proportional to the number of cores.
- *Cost*- A GPU with hundreds of core is a lot cheaper than a system with hundred CPU cores. Basically, the GPU has enormous cost advantage, GFlops per dollar, in comparison with CPUs.

3

Massively Parallel Static/Dynamic State Estimation: Single GPU Implementation

3.1 Introduction

This chapter presents a unified framework for GPU-based static/dynamic state estimation of large-scale power system.

The evolution of power systems toward the new smart grid era is bringing unprecedented operational challenges toward online monitoring of networks. Traditional SSE and DSE are not scalable enough to process the large amount of data generated over the grid. Indeed, new approaches which are both fast and accurate are required for efficient monitoring of the system dynamic behaviour.

The novelty of the proposed method includes the parallel implementation of SSE on GPU, data collation method which is used to prepare measurement set, and the massive-thread parallel implementation of the DSE (MPDSE) on GPU which to the best of our knowledge is not reported yet. In this work, using an NVIDIA® GPU, separate tasks are assigned to individual compute unified device architecture (CUDA^{TM}) abstracted threads. Therefore, the computationally onerous tasks are off-loaded and executed in parallel utilizing thousands of threads, accelerating the process of state estimation significantly.

3.2 State Estimation Formulation

The weighted least squares method is a commonly used method for state estimation which tries to minimize the weighted sum of the squares of the residuals between the estimated and actual measurements [5].

3.2.1 Weighted Least Square Static State Estimation

Consider the measurement set vector m as:

$$\mathbf{m} = \begin{bmatrix} \mathrm{m_1} \\ \mathrm{m_2} \\ . \\ . \\ . \\ \mathrm{m_{2(l+n)+1}} \end{bmatrix} = \begin{bmatrix} \mathrm{h_1(x_1, ..., x_n)} \\ \mathrm{h_2(x_1, ..., x_n)} \\ . \\ . \\ . \\ \mathrm{h_{2(l+n)+1}(x_1, ..., x_n)} \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ . \\ \varepsilon_{2(l+n)+1} \end{bmatrix} = \mathbf{h(x)} + \boldsymbol{\varepsilon}, \tag{3.1}$$

where $\mathbf{m}$, $\mathbf{h(x)}$ and $\boldsymbol{\varepsilon}$, are the vectors of measurements, nonlinear measurement functions, and measurement errors, respectively. For a system with n buses and l lines, there are $2l + 2n + 1$ elements in each vector: $2l$ power flows, $2n$ power injections, and slack bus measurements. x is a vector of system states comprising of voltage magnitudes and phase angles. Since the phase angle in slack bus is considered 0, there are $2n - 1$ states to be estimated. For simplicity, it is assumed that:

- $\mathrm{Ex}[\varepsilon_\mathrm{i}] = 0 \quad i = 1, 2, ..., 2(l + n) + 1,$
- $\mathrm{Ex}[\varepsilon_\mathrm{i}\varepsilon_\mathrm{j}] = 0,$

Therefore $\mathrm{Cov}[\boldsymbol{\varepsilon}] = \mathbf{R} = \mathrm{Ex}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\mathrm{T}] = \mathrm{diag}(\sigma_1^2, ..., \sigma_{2(\mathrm{l+n})+1}^2)$, where σ_i is the standard deviation of measurement i.

Substituting the first-order Taylor's expansion of $\mathbf{h(x)}$ around $\mathbf{x_0}$ in (3.1), we obtain:

$$\mathbf{m} - \mathbf{h}(\mathbf{x_0}) = \mathbf{H}\Delta(\mathbf{x}) + \boldsymbol{\varepsilon}, \tag{3.2}$$

where $\Delta(\mathbf{x}) = \mathbf{x} - \mathbf{x_0}$ is the $(2n - 1) \times 1$ state mismatch vector and $\mathbf{H}$ is the $(2l + 2n + 1) \times (2n - 1)$ Jacobian matrix defined as:

$$\mathbf{H(x)} = \begin{bmatrix} \frac{\partial \mathrm{h_1(x)}}{\partial \mathrm{x_1}} & \cdots & \frac{\partial \mathrm{h_1(x)}}{\partial \mathrm{x_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{h_{2(l+n)+1}(x)}}{\partial \mathrm{x_1}} & \cdots & \frac{\partial \mathrm{h_{2(l+n)+1}(x)}}{\partial \mathrm{x_n}} \end{bmatrix} \tag{3.3}$$

The objective function $\mathrm{J(x)}$ to be minimized by the WLS formulation can be expressed as:

$$\mathbf{J(x)} = \sum_{k=1}^{2l+2n+1} (m_k - h_k(x))^2 R_{kk}^{-1} = [\mathbf{m} - \mathbf{h(x)}]^\mathrm{T} \mathbf{R}^{-1} [\mathbf{m} - \mathbf{h(x)}], \tag{3.4}$$

where R is the $(2l + 2n + 1) \times (2l + 2n + 1)$ covariance matrix. Index k refers to the k^{th} measurement. The following equation satisfies the first-order optimality condition at the minimum of J(x):

$$\mathbf{g}(\mathbf{x}) = \frac{\partial \mathbf{J}(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{H}^{\mathrm{T}}(\mathbf{x})\mathbf{R}^{-1}[\mathbf{m} - \mathbf{h}(\mathbf{x})] = 0, \tag{3.5}$$

where $\mathbf{g}(\mathbf{x})$ is the $(2n-1) \times 1$ matrix of gradient of the objective function. Substituting the first-order Taylor's expansion of $\mathbf{g}(\mathbf{x})$ in (4), the following equation is solved iteratively to find the solution which minimizes $\mathbf{J}(\mathbf{x})$:

$$\mathbf{G}(\mathbf{x})\Delta(\mathbf{x}) = \mathbf{H}^{\mathrm{T}}(\mathbf{x})\mathbf{R}^{-1}[\mathbf{m} - \mathbf{h}(\mathbf{x})], \tag{3.6}$$

where $\mathbf{G}(\mathbf{x}) = \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$ is $(2n-1) \times (2n-1)$ gain matrix. The WLS state estimation algorithm given by (4.20)-(A.3) can be solved iteratively until convergence of $\Delta(\mathbf{x})$. Fig. 3.1 shows the block diagram of state estimation process.

3.2.2 Extended Kalman Filter Dynamic State Estimation

Using the present and previous states of the network, DSE predicts the state vector one step ahead of the time. The generic power system for DSE can be described by:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k) + \mathbf{w}_k, \tag{3.7}$$

$$\mathbf{m}_{k+1} = \mathrm{h}(\mathbf{x}_{k+1}) + \varepsilon_{k+1}, \;\; \varepsilon_k \sim \mathrm{N}(0, \mathbf{R}_k), \tag{3.8}$$

where x is a vector of system states comprising of voltage magnitudes and phase angles at all buses except the slack bus where $\mathrm{V}_1 = 1\angle 0°$ p.u. $\mathbf{f}(\mathbf{x})$, $\mathbf{m}$ and $\mathbf{h}(\mathbf{x})$, are vectors of nonlinear system transition function, unified measurements, and nonlinear measurement functions, respectively. ε and $\mathbf{w}$ are measurements and system noises assuming normal distribution with zero mean, and covariance $\mathbf{R}$. (3.7) can be linearized as follows if the time frame is small enough:

$$\mathbf{x}_{k+1} = \mathbf{F}_k\mathbf{x}_k + \mathbf{a}_k + \omega_k, \;\; \omega_k \sim \mathrm{N}(0, \mathbf{Q}_k), \tag{3.9}$$

where $\mathbf{F}_k$ represents the $(2n-1) \times (2n-1)$ state transition matrix between two time frames, $\mathbf{a}_k$ is the vector of associated behavior of the state trajectory, and ω_k is the Gaussian noise vector with zero mean and covariance matrix $\mathbf{Q}_k$.

Generally, EKF is composed of three major steps: identification, prediction, and filtering which are explained in details as follows:

a) Parameter Identification

To evaluate the dynamic model, unknown parameters need to be calculated online. Holt's exponential smoothing technique [136] was used for identification of F_k and a_k. Based on

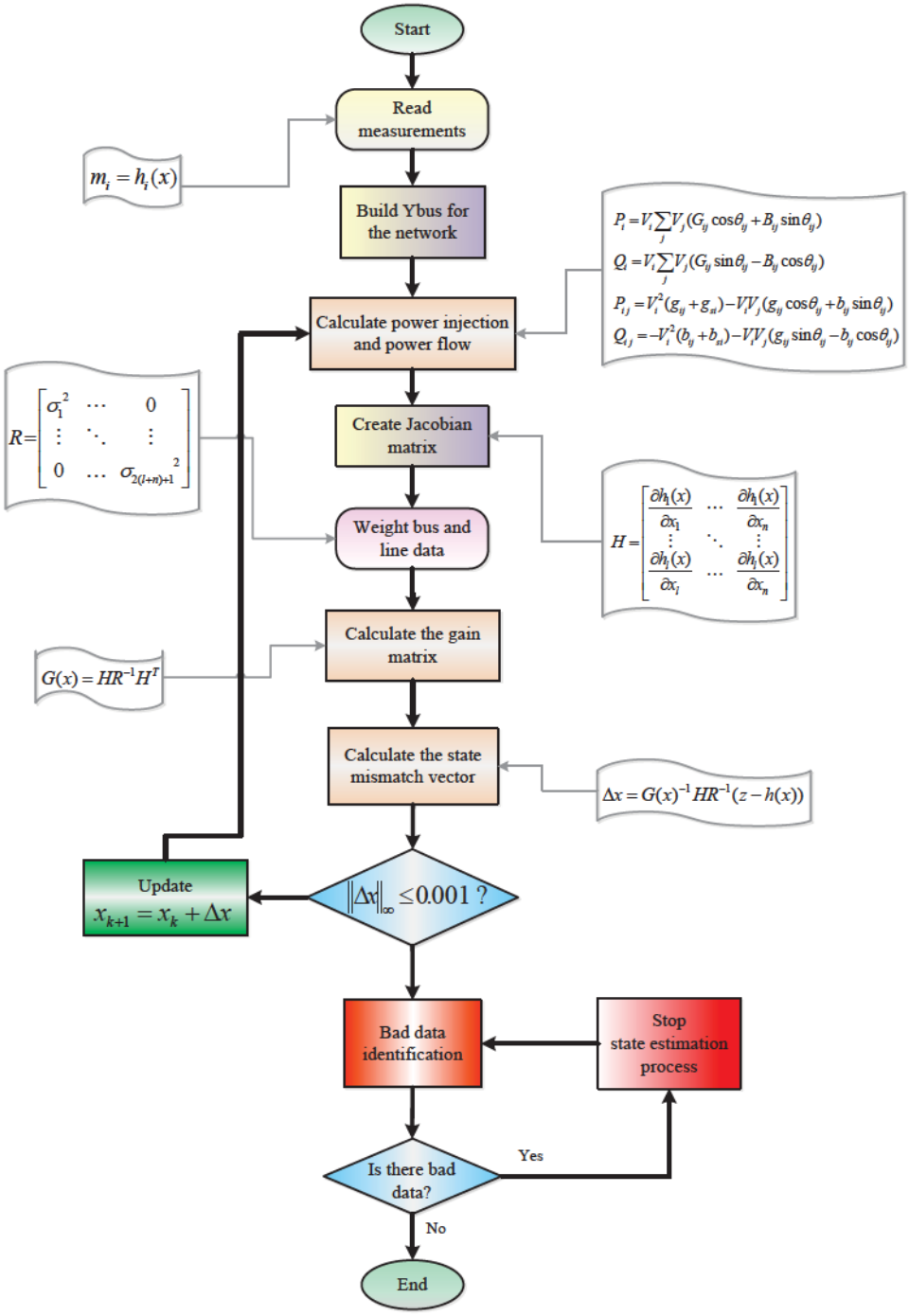


Figure 3.1: State estimation process block diagram.

this method F_k and a_k can be described as follows if $\tilde{x}$ and $\hat{x}$ represent the predicted and estimated value of the states, respectively:

$$\begin{aligned} \mathbf{F}_k &= \alpha(1+\beta)\mathbf{I}_{idn}, \;\; 0 < (\alpha, \beta) < 1, \\ \mathbf{a}_k &= (1+\beta)(1-\alpha)\tilde{\mathbf{x}}_k - \beta\gamma_{k-1} + (1+\beta)\xi_{k-1}, \\ \gamma_k &= \alpha\hat{\mathbf{x}}_k + (1-\alpha)\tilde{\mathbf{x}}_k, \\ \xi_k &= \beta(\gamma_k - \gamma_{k-1}) + (1-\beta)\xi_{k-1}, \end{aligned} \tag{3.10}$$

where α and β are smoothing parameters. Under normal operation conditions it is possible to adjust $\mathbf{F}_k$ and $\mathbf{a}_k$ such that $\mathbf{Q}_k$ remains constant. $\mathbf{I}_{idn}$ is the $(2n-1)\times(2n-1)$ identity matrix. However, considering the dynamic behaviour of the network neglecting the changes in $\mathbf{Q}_k$ may result in inaccurate prediction. Online estimation of $\mathbf{Q}_k$ can be formulated as [137]:

$$\hat{\mathbf{Q}}_{k+1} = \mathbf{Q}_k \sqrt{\frac{\text{trace}\{\mathbf{H}_{k+1}(\mathbf{F}_k \rho_k {\mathbf{F}_k}^T + \hat{\mathbf{Q}}_k){\mathbf{H}^T}_{k+1}\}}{\text{trace}\{\mathbf{H}_{k+1}(\mathbf{F}_k \rho_k {\mathbf{F}_k}^T + \mathbf{Q}_k){\mathbf{H}^T}_{k+1}\}}}, \tag{3.11}$$

b) State Prediction

Using the measurement and estimated states at the time instant k, the predicted value $\tilde{\mathrm{x}}_{k+1}$ can be formulated as:

$$\begin{aligned} \tilde{\mathbf{x}}_{k+1} &= \mathbf{F}_k\hat{\mathbf{x}}_k + \mathbf{a}_k, \;\; (\mathbf{x}_k - \hat{\mathbf{x}}_k) \sim \mathrm{N}(0, \rho_k), \\ \tilde{\rho}_{k+1} &= \mathbf{F}_k \rho_k {\mathbf{F}_k}^T + \mathbf{Q}_k, \;\; (\mathbf{x}_k - \tilde{\mathbf{x}}_k) \sim \mathrm{N}(0, \tilde{\rho}_k), \end{aligned} \tag{3.12}$$

where ρ and $\tilde{\rho}$ are $(2n-1)\times(2n-1)$ error covariance matrices for estimated and predicted values, respectively. The objective function $\mathbf{J}(\mathbf{x})$ was chosen to minimize both estimation and prediction errors:

$$\mathbf{J}(\mathbf{x}) = \underset{\mathrm{x}}{\text{argmin}} \; [\mathbf{m} - \mathbf{h}(\mathbf{x})]^T \mathbf{R}^{-1} [\mathbf{m} - \mathbf{h}(\mathbf{x})] + [\mathbf{x} - \tilde{\mathbf{x}}]^T \tilde{\rho}^{-1} [\mathbf{x} - \tilde{\mathbf{x}}], \tag{3.13}$$

The following equation satisfies the first-order optimality condition at the minimum of $\mathbf{J}(\mathbf{x})$:

$$\mathbf{g}(\mathbf{x}) = \mathbf{H}^T(\mathbf{x}) \mathrm{R}^{-1} [\mathbf{m} - \mathbf{h}(\mathbf{x})] - \tilde{\rho}^{-1} [\mathrm{x} - \tilde{\mathbf{x}}] = 0, \tag{3.14}$$

where $\mathbf{g}(\mathbf{x})$ is the $(2n-1)\times 1$ vector of gradient of the objective function, and $\mathbf{H} = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}$ is the $(2m+2n+1)\times(2n-1)$ Jacobian matrix. Using Taylor's expansion of $\mathbf{h}(\mathbf{x})$ around $\tilde{\mathbf{x}}_0$ (3.14) can be expressed as follows:

$$\begin{aligned} \mathbf{G}(\mathbf{x})\Delta(\mathbf{x}) &= \mathbf{H}^T(\tilde{\mathbf{x}}) \mathbf{R}^{-1} [\mathbf{m} - \mathbf{h}(\tilde{\mathbf{x}})], \\ \mathbf{G}(\mathbf{x}) &= \mathbf{H}^T(\tilde{\mathbf{x}}) \mathbf{R}^{-1} \mathbf{H}(\tilde{\mathbf{x}}) + \tilde{\rho}^{-1}, \end{aligned} \tag{3.15}$$

where $\Delta(\mathbf{x}) = \hat{\mathbf{x}} - \tilde{\mathbf{x}}_0$ is the $(2n-1)\times 1$ state mismatch vector and $\mathbf{G}(\mathbf{x}) = \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$ is the $(2n-1)\times(2n-1)$ gain matrix. The state estimation algorithm given by (3.13)-(3.15) can be solved iteratively until convergence of $\Delta(\mathbf{x})$ to a specified threshold.

c) State Filtering

This step updates the predicted values using the next set of measurements at the time instant *k+1*. The updated state through EKF can be written as:

$$\begin{aligned} \hat{\mathbf{x}}_{k+1} &= \tilde{\mathbf{x}}_{k+1} + \mathbf{K}_{k+1}(\mathbf{m}_{k+1} - \mathbf{h}(\tilde{\mathbf{x}}_{k+1})), \\ \mathbf{K}_{k+1} &= \tilde{\rho}_{k+1}\mathbf{H}^{T}{}_{k+1}[\mathbf{H}_{k+1}\tilde{\rho}_{k+1}\mathbf{H}^{T}{}_{k+1} + \mathbf{R}]^{-1}, \\ \rho_{k+1} &= \tilde{\rho}_{k+1} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\tilde{\rho}_{k+1}, \end{aligned} \tag{3.16}$$

where $\mathbf{K}$ is the $(2n-1)\times(2n-1)$ Kalman gain matrix. For the same reasons mentioned earlier, this step is also a good candidate for parallelization.

3.3 Measurement and Component Modeling

An individual transmission line is typically modeled as a single phase π circuit equivalent. An equivalent π model of a two bus (i,j) system is shown in Fig. 3.2. The measurements in an AC system are mainly of three types, bus power injection, line power flows and bus voltage magnitudes. These quantities can be expressed using the state variables.

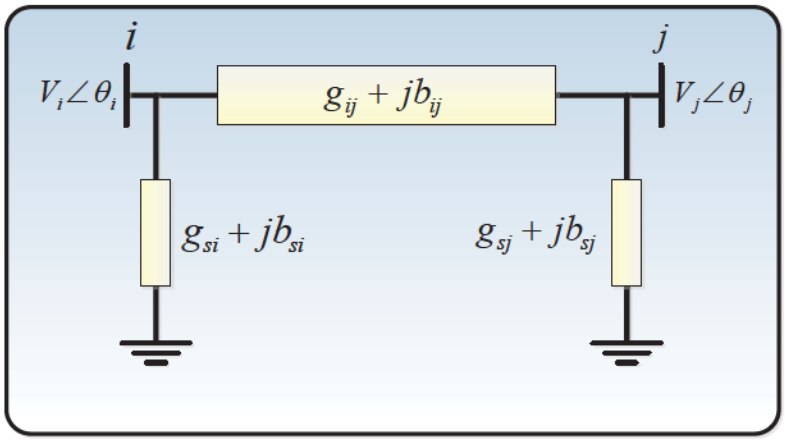


Figure 3.2: Standard transmission line π model.

The real and reactive power injection at a bus can be expressed as:

$$\begin{aligned} \mathrm{P_i} &= \mathrm{V_i} \sum_{\mathrm{j=1}}^{\mathrm{2l+2n+1}} \mathrm{V_j}(\mathrm{G_{ij}}\cos\theta_{\mathrm{ij}} + \mathrm{B_{ij}}\sin\theta_{\mathrm{ij}}), \\ \mathrm{Q_i} &= \mathrm{V_i} \sum_{\mathrm{j=1}}^{\mathrm{2l+2n+1}} \mathrm{V_j}(\mathrm{G_{ij}}\sin\theta_{\mathrm{ij}} - \mathrm{B_{ij}}\cos\theta_{\mathrm{ij}}), \end{aligned} \tag{3.17}$$

where $\mathrm{P_i}$ and $\mathrm{Q_i}$ are the real and reactive bus power injection at bus i, respectively. $\mathrm{V_i}$ and θ_{i} are the voltage magnitude and phase angle at bus i and $\theta_{\mathrm{ij}} = \theta_{\mathrm{i}} - \theta_{\mathrm{j}}$.

$(G_{ij} + jB_{ij})$ is the ij-th element of the complex bus admittance matrix.

The real and reactive power flow from bus i to bus j are expressed as follows:

$$\begin{aligned} P_{ij} &= V_i^2(g_{si} + g_{ij}) - V_j V_i(g_{ij}cos\theta_{ij} + b_{ij}sin\theta_{ij}), \\ Q_{ij} &= -V_i^2(b_{si} + b_{ij}) - V_j V_i(g_{ij}sin\theta_{ij} - b_{ij}cos\theta_{ij}), \end{aligned} \tag{3.18}$$

where P_{ij} and Q_{ij} are the real and reactive bus power flow from bus i to bus j , respectively.
$(g_{ij} + jb_{ij})$ is the admittance of the series branch connecting buses i and j.
$(g_{si} + jb_{si})$ is the admittance of the shunt branch connected at bus i.

To construct the Jacobian matrix (**H**), the partial derivative of line flows and bus power with respect to θ_i, θ_j, V_i and V_j should be computed:

$$\mathbf{H(x)} = \left[\begin{array}{ccc|ccc} \frac{\partial V_1}{\partial V_1} & \cdots & \frac{\partial V_1}{\partial V_n} & \frac{\partial V_1}{\partial \theta_2} & \cdots & \frac{\partial V_1}{\partial \theta_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial V_n}{\partial V_1} & \cdots & \frac{\partial V_n}{\partial V_n} & \frac{\partial V_n}{\partial \theta_2} & \cdots & \frac{\partial V_n}{\partial \theta_n} \\ & \vdots & & & \vdots & \\ \cdots & \frac{\partial P_{ij}}{\partial V_i}, \frac{\partial P_{ij}}{\partial V_j} & \cdots & \cdots & \frac{\partial P_{ij}}{\partial \theta_i}, \frac{\partial P_{ij}}{\partial \theta_j} & \cdots \\ & \vdots & & & \vdots & \\ & \vdots & & & \vdots & \\ \cdots & \frac{\partial Q_{ij}}{\partial V_i}, \frac{\partial Q_{ij}}{\partial V_j} & \cdots & \cdots & \frac{\partial Q_{ij}}{\partial \theta_i}, \frac{\partial Q_{ij}}{\partial \theta_j} & \cdots \\ & \vdots & & & \vdots & \\ \frac{\partial P_1}{\partial V_1} & \cdots & \frac{\partial P_1}{\partial V_n} & \frac{\partial P_1}{\partial \theta_2} & \cdots & \frac{\partial P_1}{\partial \theta_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_n}{\partial V_1} & \cdots & \frac{\partial P_n}{\partial V_n} & \frac{\partial P_n}{\partial \theta_2} & \cdots & \frac{\partial P_n}{\partial \theta_n} \\ \frac{\partial Q_1}{\partial V_1} & \cdots & \frac{\partial Q_1}{\partial V_n} & \frac{\partial Q_1}{\partial \theta_2} & \cdots & \frac{\partial Q_1}{\partial \theta_n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial V_1} & \cdots & \frac{\partial Q_n}{\partial V_n} & \frac{\partial Q_n}{\partial \theta_2} & \cdots & \frac{\partial Q_n}{\partial \theta_n} \end{array}\right] \tag{3.19}$$

3.4 Numerical Methods for Solving Linear Systems

Solution of a linear system is usually the most computationally expensive step in various power system analyses such as power flow and state estimation. In general, a linear system can be written as:

$$\mathbf{Ax} = \mathbf{b}, \tag{3.20}$$

where **A** is a $n \times n$ square matrix known as coefficient matrix, **b** is a $n \times 1$ vector and **x** is a $n \times 1$ vector.

Methods of solving linear systems fall into two general categories: direct methods and iterative methods. Direct methods obtain the exact solution of the system in a definite number of operations, whereas iterative methods calculate sequences of approximations that may or may not converge to the solution. While direct methods obtain an exact solution of the linear system, they require significantly more computations than iterative methods.

3.4.1 Direct Method

To solve the linear system $\mathbf{Ax} = \mathbf{b}$ several different algorithms can be used. One is to explicitly calculate the inverse of the coefficient matrix and multiply it by vector **b**. This method is computationally very expensive especially for large, sparse matrices, such as those encountered in power systems. Thus, various methods have been developed to solve a system of linear equations without explicitly calculating the inverse of the linear system. The most famous direct methods are LU-Decomposition and Cholesky decomposition [138].

Depending on the properties of the matrix A, different factorizations are used:

- For an $n \times n$ symmetric positive definite matrix, the Cholesky factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ is usually computed, where **L** is a lower triangular matrix.
- For a nn asymmetric matrix , its LU decomposition $\mathbf{A} = \mathbf{LU}$ is computed where **L** is a unit lower triangular matrix, and **U** is an upper triangular matrix.

LU Decomposition

The idea is to factor $\mathbf{A} = \mathbf{LU}$ where L is lower-triangular and **U** is upper-triangular. Then you solve the pair of equations as follows:

$$\underbrace{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & \cdots & a_{n,n} \end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & l_{n,n} \end{bmatrix}}_{\mathbf{L}} \underbrace{\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{n,n} \end{bmatrix}}_{\mathbf{U}} \tag{3.21}$$

Then the linear system of (3.20) becomes

$$\mathbf{L}\underbrace{\mathbf{Ux}}_{\mathbf{Z}} = \mathbf{b} \;\Rightarrow\; \mathrm{L}\mathbf{Z} = \mathbf{b}. \tag{3.22}$$

Fig. 3.3 shows the algorithm of LU decomposition.

Algorithm: LU decomposition

Initialize $U = A, \quad L = I$

for $i = 1 : n-1$

  for $j = i+1 : n$

    $L_{j,i} = U_{j,i} / U_{i,i}$

    for $k = i : n$

      $U_{i,k} = U_{j,k} - L_{j,i} U_{i,k}$

    end

  end

end

Figure 3.3: LU decomposition algorithm.

Cholesky Decomposition

The Cholesky factorization is a special LU factorization technique which decomposes the coefficient matrix into $\mathbf{L}\mathbf{L}^{\mathrm{T}}$. Where $\mathbf{L}$ is a lower triangular matrix with real and positive diagonal entries, and $\mathbf{L}^{\mathrm{T}}$ denotes the transpose of $\mathbf{L}$. For symmetric positive definite matrices, Cholesky factorization needs less computation and memory space, since only the elements $a_{i,j}, i = j, ..., n; j = 1, ..., n$ should be stored in memory. Cholesky decomposition algorithm is shown in Fig. 3.3.

3.4.2 Iterative Method

In contrast with direct methods, iterative methods construct a series of solution approximations such that it converges to the exact solution of a system. It starts with an approximation to the solution of (3.20) and improves this approximate solution in each iteration. The approximate solution may converge to the exact solution in an infinite or infinite number of iterations. The iterative method can be stopped whenever the desired accuracy in the solution is obtained. Some of the famous method in this category includes conjugate gradient (CG), Gauss-Seidel, successive over relaxation and Jacobi iterative method. Here we only focus on CG. Detailed description of the properties of iterative methods and their algorithms is provided in [139].

Conjugate Gradient

Starting from (3.20) and assuming A is a symmetric matrix, the conjugate gradient method was originally developed to minimize following quadratic function [140]:

Algorithm: Cholesky decomposition

Initialize $L = A$

for $i = 1 : n$

$\alpha = \beta = 0$

for $k = 1 : i - 1$

$\alpha = \alpha + L_{i,k}^{\ 2}$

$\beta = \beta + L_{i,k} L_{j,k}$

end

$L_{i,i} = L_{i,i} - sum$

for $j = i + 1 : n$

$L_{j,i} = (L_{j,i} - \beta) / L_{i,i}$

end

end

Figure 3.4: Cholesky decomposition algorithm.

$$\mathrm{f(x)} = \frac{1}{2}\mathrm{x^T Ax} - \mathrm{b^T x}, \tag{3.23}$$

The gradient vector points into the direction of the maximum increase of the f(x). The gradient of f(x) is defined as:

$$\mathbf{f_g(x)} \triangleq \left[\begin{array}{cccc} \frac{\partial \mathbf{f(x)}}{\partial \mathrm{x_1}} & \frac{\partial \mathbf{f(x)}}{\partial \mathrm{x_2}} & \cdots & \frac{\partial \mathbf{f(x)}}{\partial \mathrm{x_n}} \end{array} \right]^T, \tag{3.24}$$

$$\mathbf{f_g(x)} = \frac{1}{2}(\mathbf{A^T x} + \mathbf{Ax}) - \mathbf{b}, \tag{3.25}$$

Since $\mathbf{A}$ is symmetric, then $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$ and (3.25) can be rewritten as:

$$\mathbf{f_g(x)} = \mathbf{Ax} - \mathbf{b}, \tag{3.26}$$

Therefore, setting the gradient vector to zero and finding the critical point of $\mathbf{f(x)}$ is equal to solving the linear system $\mathbf{Ax} = \mathbf{b}$. The process start from a gauss about the solution and the method tries to reduce the residual in each iteration and get close to the exact solution as much as possible. The residual vectored is defined as $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$. If $\mathbf{r}$ is less than the threshold, or the iteration has exceeded the allowed maximum iterations, the algorithm will stop. Fig. 3.5 describe the conjugate gradient algorithm.

Algorithm: Conjugate gradient method

Choose an initial solution x_0

Calculate $d_0 = r_0 = b - Ax_0$

Set $k = 0$

while $r^{k+1} \geq \varepsilon$ ***do***

Calculate $\alpha_k = r_k r_k^T / d_k A d_k^T$

Update
$$x_{k+1} = x_k + \alpha_k d_k$$
$$r_{k+1} = r_k - \alpha_k d_k$$

Calculate
$$\beta_{k+1} = r_{k+1} r_{k+1}^T / r_k r_k^T$$
$$d_{k+1} = r_{k+1} + \beta_{k+1} d_k$$

Update $k = k+1$

end

Figure 3.5: Conjugate gradient algorithm.

Preconditioned Conjugate Gradient

Iterative solvers are mainly less robust compared to direct solvers. To combat this problem, preconditioned are developed to improve the performance by speeding the convergence rate which leads to less number of iterations and thus less run time. Preconditioning transforms the original linear system into one which has the same solution, but with a better condition number [141]. The condition number of the matrix is often used to quantify the eigenvalue spread of a matrix, and it is defined as:

$$\text{Cond}_\text{A} = \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}, \tag{3.27}$$

where Cond_A denotes the condition number, $\lambda_{\max}$ is the maximum eigenvalue and $\lambda_{\min}$ is the minimum eigenvalue. The system is said to be ill-conditioned, if the condition number is considerably more than unity.

Consider (3.20) which **A** is symmetric and positive-definite, let **M** be a preconditioner which approximates **A** in preserving the same solution. It is assumed that **M** is also symmetric positive definite. Then, the following preconditioned system could be solved:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}, \tag{3.28}$$

To preserve symmetry one can decompose M in its Cholesky factorization and split the preconditioner between left and right:

$$\mathbf{L}^{-1}\mathbf{A}\underbrace{\mathbf{L}^{-T}\mathbf{u}}_{\mathbf{x}} = \mathbf{L}^{-1}\mathbf{b}, \tag{3.29}$$

Considering the sparsity pattern and the structure of matrix **A**, Cholesky preconditioner was selected which has the lowest computational cost for symmetric positive definite matrices. The preconditioned conjugate gradient algorithm is described in Fig. 3.6.

Algorithm: Preconditioned conjugate gradient method

Choose an initial solution x_0

Calculate $r_0 = b - Ax_0$

Set $k = 0$

Calculate $P_0 = M^{-1}r_0$

while $r^{k+1} \geq \varepsilon$ ***do***

Calculate $\alpha_k = r_k^T M^{-1} r_k / P_k A P_k^T$

Update
$$x_{k+1} = x_k + \alpha_k P_k$$
$$r_{k+1} = r_k - \alpha_k A P_k$$

Calculate
$$\beta_k = r_{k+1}^T M^{-1} r_{k+1} / r_k^T M^{-1} r_k$$
$$P_{k+1} = M^{-1} r_{k+1} + \beta_k P_k$$

Update $k = k+1$

end

Figure 3.6: Preconditioned conjugate gradient algorithm.

3.5 Test Systems

In order to evaluate the efficiency of the proposed GPU-based method, large-scale systems were constructed by duplicating the IEEE 39-bus system. Details are available in Appendix C. The uniform set of measurements which are the input to the state estimation algorithms are obtained by corrupting online power flow results of the test power systems with Gaussian noise of zero mean and covariance **R**. Therefore, to assess the accuracy of the state estimator, the results are verified using bus voltage magnitudes and phase angles for all test cases modeled in PSS/E®. While it is possible to use a partial set of measurement, all of the measurements are included to make the problem as complicated as possible for the GPU. All results are verified by in terms of both accuracy and time efficiency.

3.6 GPU Implementation of Static WLS State Estimator

In steady state condition power system can be considered as a quasi static system with slow changes which update almost every 30 second. Therefore, for SSE it is sufficient to only consider SCADA measurements which provide new measurements every 2-5s. SSE can not monitor the dynamic behaviour of the system, it only provides a snapshot of the system changes which is useful for normal operation condition.

Fig. 3.7 illustrates the implementation of WLS state estimator on the GPU. Initialization was done in Stage 1. After transferring the data from CPU to GPU, all the other steps were executed in the GPU. Stage 2 contains the main parts of the parallel kernel. All the vector products for the computation of admittance matrix $\mathbf{Y}$, measurement function $\mathbf{h}(\mathbf{x})$ and Jacobian function $\mathbf{H}(\mathbf{x})$ were done in parallel utilizing CUDA kernels. The transpose of $\mathbf{H}(\mathbf{x})$ or the computation of residual $\mathbf{r}$ are not intensive tasks like matrix or vector products, however for large-scale systems it can take significant execution time which is here reduced using parallel implementation. To compute the gain matrix $\mathbf{G}(\mathbf{x})$ and the gradient of the objective function $\mathbf{g}(\mathbf{x})$, the matrix-matrix multiplication $\mathbf{H}^{T}\mathbf{R}^{-1}$ was partitioned into a series of independent operations $\mathbf{H}_{i}^{T}\mathrm{R}_{j}^{-1}$ where $\mathbf{H}_{i}^{T}$ and $\mathbf{R}_{j}^{-1}$ refer to the i^{th} row and j^{th} column in $\mathbf{H}^{T}$ and $\mathbf{R}^{-1}$ matrices, respectively. The vector inner product of $\mathbf{H}_{i}^{T}\mathrm{R}_{j}^{-1}$ was defined as the sum of the elements of $\mathbf{H}_{i_a}^{T}\mathbf{R}_{j_b}^{-1}$ where $\mathbf{H}_{i_a}^{T}$ and $\mathbf{R}_{j_b}^{-1}$ are a^{th} and b^{th} element of the $\mathbf{H}_{i}^{T}$ and $\mathbf{R}_{j}^{-1}$, respectively. These independent operations are simultaneously executed by individual CUDA threads. In total $w \times z$ single threads were needed for product of $\mathbf{H}^{T}$ with w rows and $\mathbf{R}^{-1}$ with z columns. Using CUDA basic linear algebra subroutines (CUBLAS) library, $\mathbf{G}(\mathbf{x})$ was decomposed in parallel. Code for LU decomposition and for solving $\Delta(\mathbf{x})$ was prepared utilizing *cublasSscal()*, *cublasSswap()* and *cublasStrsv()* functions. After updating $\Delta(\mathbf{x})$ in Stage 3, convergence check was done in Stage 4.

3.6.1 Parallel Kernel Structure

Generally, matrix-vector and matrix-matrix product is time consuming for large data-sets. In the WLS method, the computation of $\mathbf{H}^{T}(\mathbf{x})\mathbf{R}^{-1}[\mathbf{m} - \mathbf{h}(\mathbf{x})]$ and $\mathbf{H}^{T}(\mathbf{x})\mathbf{R}^{-1}\mathbf{H}$ can take a long time to complete even on CPU clusters. The matrix-vector multiplication to calculate $\mathbf{R}^{-1}[\mathbf{m} - \mathbf{h}(\mathbf{x})]$ takes about one order of magnitude less execution time than that of $\mathbf{H}^{T}(\mathbf{x})\mathrm{R}^{-1}[\mathbf{m} - \mathbf{h}(\mathbf{x})]$ or $\mathbf{H}^{T}(\mathbf{x})\mathbf{R}^{-1}\mathbf{H}$ but still can be a significant burden under emergency situations. Matrix-matrix and matrix-vector products contains several *for* loops in their implementation, which are the best candidates for parallelization utilizing GPU threads. Since all iterations of the loop can be executed in parallel, by assigning each iteration in a loop to individual CUDA threads, the task can be converted into a CUDA kernel.

In addition, solving $\mathbf{G}(\mathbf{x})\Delta(\mathbf{x}) = \mathbf{g}(\mathbf{x})$ by inversion of $\mathbf{G}(\mathbf{x})$ is considerably expensive due to the sheer size of the inverted matrix. As an alternative method, LU decomposition

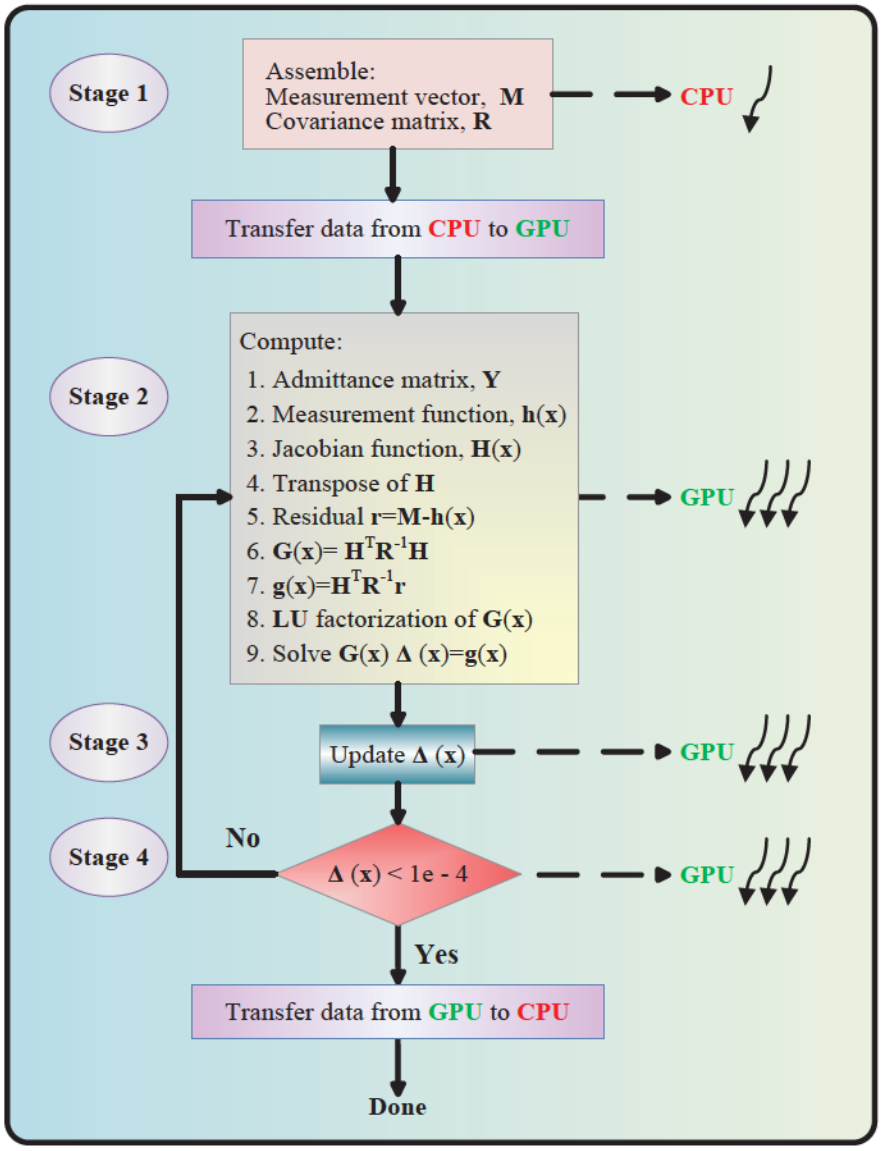


Figure 3.7: GPU implementation of the WLS algorithm.

is used in this work.

3.6.2 Experimental Results

To evaluate accuracy and efficiency of the parallel SSE algorithms, experiments were conducted based on two separate simulation codes: multi-thread CPU-based code in C++, and a massive-thread GPU-based code written in C++ and CUDA. The results of state estimation are compared with the CPU simulations. Since the matrices are highly sparse in state estimation, all matrices and vectors are stored in compressed sparse row format to reduce the computational burden.

Efficiency Evaluation of the GPU-based Static State Estimator

The simulations were done using the test data sets listed in Table 3.1, with a tolerance of 0.0001 for convergence of the estimated parameters. Fig. 3.8 show the results of comparison between the CPU and the GPU programs along with the speed-up of the parallel code.

Table 3.1: Summary of Simulation Results

Case	No. of Buses	No. of Meas.	Jacobian matrix H(x)	Gain matrix G(x)	Single thread CPU	GPU Comm.	GPU Comp.	Speed-up
1	39	171	171 × 77	77 × 77	0.031s	0.017s	0.033s	0.6
2	78	347	347 × 155	155 × 155	0.18s	0.041s	0.069s	1.6
3	156	699	699 × 311	311 × 311	0.39s	0.08s	0.11s	2.05
4	312	1421	1421 × 623	623 × 623	2.7s	0.13s	0.38s	5.3
5	624	2865	2865 × 1247	1247 × 1247	16.5s	0.34s	1.56s	8.7
6	1248	5825	5825 × 2495	2495 × 2495	59.1s	0.86s	4.34s	11.4
7	2496	11553	11553 × 4991	4991 × 4991	195s	2.81s	10.19s	15
8	4992	23151	23151 × 9983	9983 × 9983	1577s	6.51s	34.99s	38

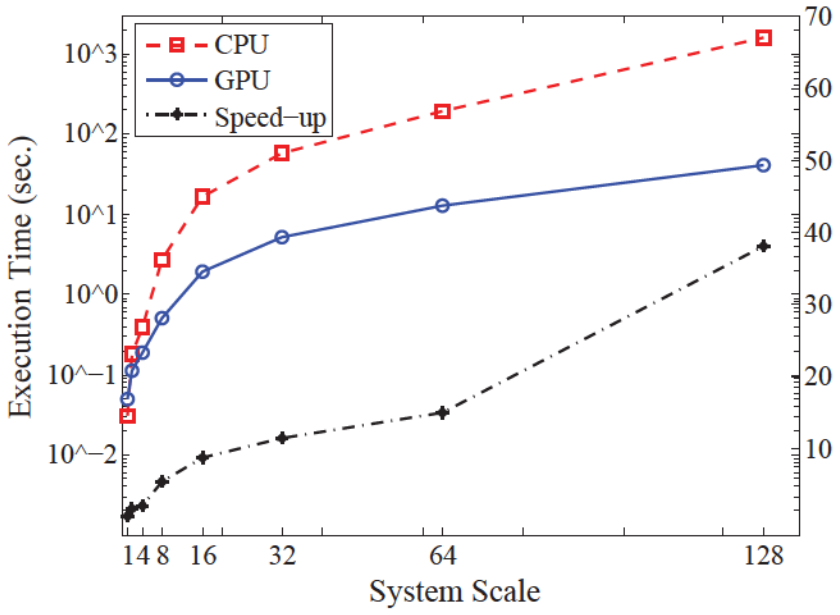


Figure 3.8: Execution time and speed-up for various case studies.

Fig. 3.9 illustrates the percentage of time taken by various steps in Stage 2 of Fig. 3.7 for all case studies. The results verify that for both the sequential CPU program and the data-parallel GPU program, the 3 most computationally demanding steps are the LU decomposition, computation of gain matrix $\mathbf{G}(\mathbf{x})$ and the solution of state mismatch vector $\Delta(\mathbf{x})$.

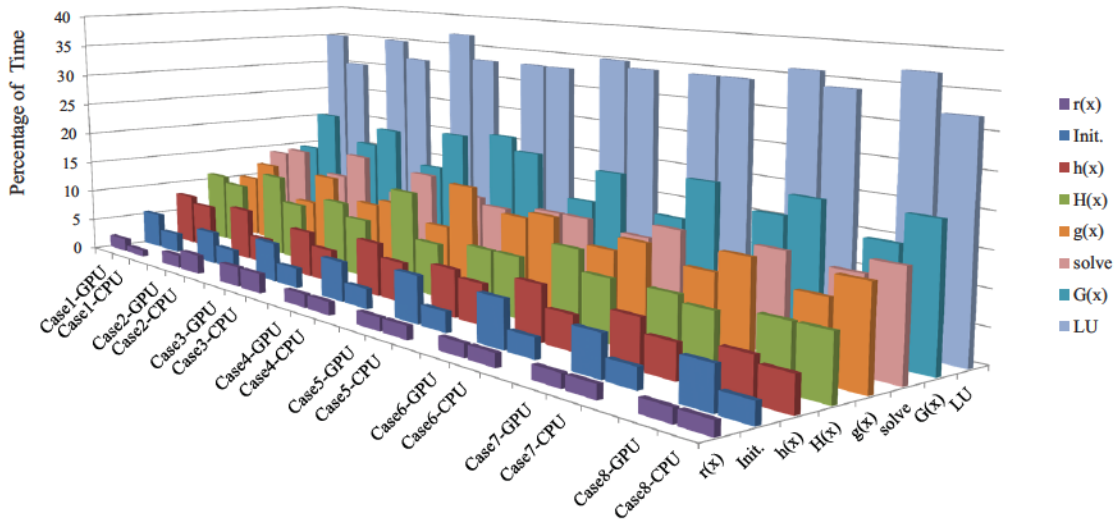


Figure 3.9: Percentage of time used for various steps in Stage 2 (Fig. 3.7).

Accuracy Analysis of the GPU-based Static State Estimator

The inputs to the GPU-based WLS state estimator are the power-flow results from PSS/E® corrupted with noise which are used as Pseudo-measurements. Therefore, to assess the accuracy of the state estimator, its output was compared with the original power-flow results from PSS/E®. The results of Case 1 in Fig. 3.10 and Fig. 3.11 verify the accuracy of the proposed method. There are small differences in the result which are justifiable considering the fact that the order of blocks execution in each grid is undefined in kernel definition. Therefore, it leads to slightly different results if different blocks perform calculations on overlapping portions of data. The estimation error for voltage magnitude and voltage angle in all of the case studies is less than 0.001 p.u. and 0.002 deg., respectively.

3.7 GPU Implementation of EKF-based Dynamic State Estimator

Since static state estimation consider the power system as a quasi-static system, it is enough to only use SCADA measurements during the estimation process. However, for dynamic state estimation it is necessary to have enough measurements to capture the dynamic of the system. So different type of measurements like PMUs are needed. PMUs provide measurements by sampling instantaneous waveforms and can deliver up to 50 measurements per second while SCADA measurements update every 2-5s. For large-scale networks, installing enough PMUs for full network observation may be expensive and impractical. A more realistic and feasible deployment of the PMU for dynamic state estimation is to use both conventional (SCADA) and synchronized measurements together.

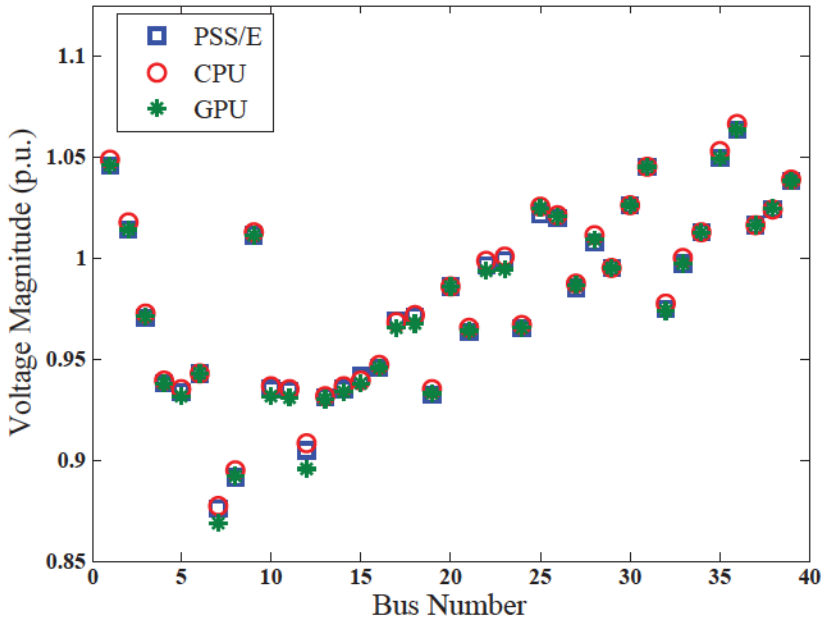


Figure 3.10: Voltage magnitudes for Case 1.

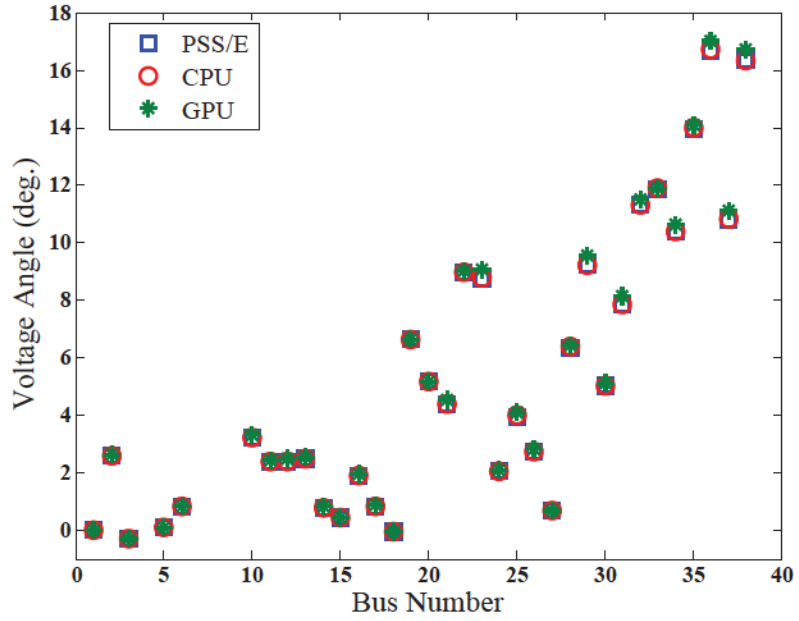


Figure 3.11: Voltage angles for Case 1.

3.7.1 Data Collation

The main goal in this section is to provide a coherent set of measurements for the MPDSE algorithm. Since it is not possible to make the whole system observable using PMUs due to the high cost of this technology, the proposed data collation method extrapolates SCADA

measurements to update them as fast as PMU measurements arrive. The process of data collation is done under the following assumptions:

- The network is observable with the existing SCADA measurements, and the PMU devices are installed at the generator buses.
- The number of PMU channels is limited to a maximum of two (one for voltage and one for current) for economical reasons.
- Since practical state estimation update every 30-60s, and SCADA measurements update every 2-5s [134, 135], SCADA and PMU refresh rate are considered as every 2s and 30/s, respectively.
- Based on the previous assumption, in between two SCADA measurements 60 PMU measurements are available. Since the changes for close measurements are very small, the buffer length of 6 is chosen to use the average of each 6 consecutive PMU measurements. So, in between two SCADA measurements there are 10 PMU measurements. However, under contingency scenarios, the actual refresh rate of the PMU measurements is used to decide the condition of the system.
- For simplicity the measurement uncertainty due to instrument transformers [142] is neglected, and PMU measurements are considered with higher accuracy than the SCADA measurements. To increase the accuracy of the results, higher weights are assigned to PMU measurements in error covariance matrix.
- SCADA data are recorded with local time stamps, so the time skew between SCADA and PMU measurements will be negligible considering the fact that quantities provided by SCADA measurements do not change significantly in a short time interval.
- The time step Δt is equal to 0.2s which is the time interval between two averaged PMU measurements.

Exponential Moving Average

The overall measurement set (m) is divided into two subsets: PMU measurements (m^P) and SCADA measurements (m^S). Since the refresh rate of m^P is a lot faster than m^S, there are more measurements available for the buses with PMU installations during the same time interval. To take advantage of all available measurements, missing SCADA measurements are extrapolated employing the exponential moving average method.

In general, for a set of observations $\boldsymbol{O}$ a moving average of order n, is the value of n consecutive observations.

$$\tilde{\mathbf{O}}_{t+1} = \frac{(\mathbf{O}_t + \mathbf{O}_{t-1} + \mathbf{O}_{t-} + ... + \mathbf{O}_{t-n+1})}{n}, \tag{3.30}$$

where t represent the time,n is the number of terms in the moving average, and $\tilde{O}$ shows forecast for the next period.

In contrast with moving average which assign the same weight to all data set, exponential moving average [143] assigns different weights to each measurements in a way that older measurements fade exponentially and new measurements have more effect on the result of prediction. Formally, the exponential smoothing equation can be presented as follows:

$$\tilde{\mathbf{O}}_{t+1} = \mu \mathbf{O}_t + (1-\mu)\tilde{\mathbf{O}}_t, \;\; 0 < \mu < 1, \tag{3.31}$$

where μ is the scalar smoothing constant.

Considering above explanation, extrapolated SCADA measurements can be formulated as:

$$\tilde{\mathbf{m}}^{E}_{k+1} = \mu \mathbf{m}^{S,E}_{k} + (1-\mu)\tilde{\mathbf{m}}^{E}_{k}, \;\; 0 < \mu < 1, \tag{3.32}$$

where $\mathbf{m}^{S,E}_{k}$ and $\tilde{\mathbf{m}}^{E}_{k}$ are previous measurements (including actual and extrapolated SCADA measurements) and extrapolated measurements, respectively. The indices $k = t$ and $k+1 = t+\Delta t$ are used for present time and one step in the future, respectively. μ is chosen to be 0.7 in this work by trial and error to get the best result.

To extrapolate a SCADA measurement, the last 10 available measurements which contain both measured SCADA and extrapolated SCADA are used. The algorithm, extrapolates all measurements even those that arrive exactly at the time the new set of SCADA measurements arrive. It should be noted that in every 10 extrapolated SCADA measurements, two of them will be available by actual measurement. Whenever new SCADA is available, extrapolated measurement is replaced by the actual measurement. To increase the accuracy, the average of last 10 estimation errors (difference between actual SCADA measurements and the extrapolated measurements) are added to the predicted value. Expanding $\tilde{m}^{E}_{k}$ with its components results in following:

$$\tilde{\mathbf{m}}^{E}_{k+1} = \mu \sum_{i=0}^{10} (1-\mu)^{i} m^{S,E}_{k-i} + (1-\mu)^{11}\tilde{m}^{E}_{k-11} + \frac{1}{10}\sum_{j=1}^{10}(\Delta e^{j}_{k})^{2}, \tag{3.33}$$

where $\Delta \mathbf{e}^{j}_{k}$ represents the estimation error at the time instant k. Indices i and j refer to the last 10 available measurements, and last 10 available estimation errors, respectively. An example of data collation for $t = 24.2s$ is shown in Fig 3.12.

Polar to Cartesian Transformation

In order to have a uniform set of measurements, the PMU measurements which are in polar format are transformed into cartesian format ($\mathbf{m}^{P}_{T}$ in Fig. 3.14). A Polar coordinate

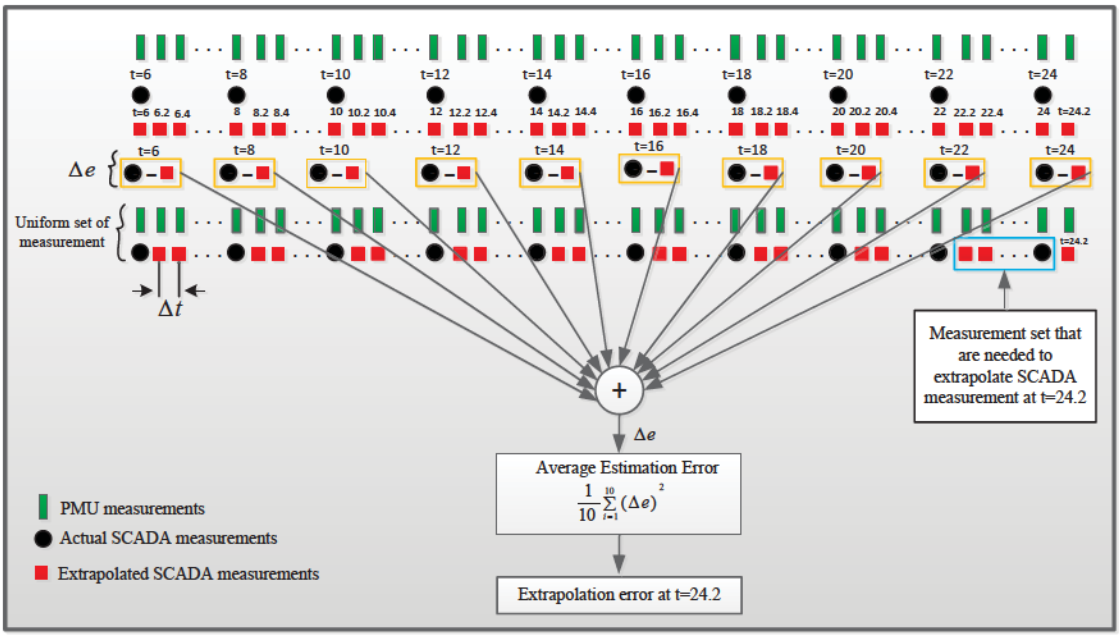


Figure 3.12: Example of data collation for MPDSE.

system is determined by a fixed point, a origin or pole, and a zero direction or axis as shown in Fig. 3.13. Each point is determined by an angle and a distance relative to the zero axis and the origin as follows:

$$\mathrm{r} = \sqrt{\mathrm{x}^2 + \mathrm{y}^2}, \quad \varphi = \arctan(\mathrm{y}/\mathrm{x}), \tag{3.34}$$

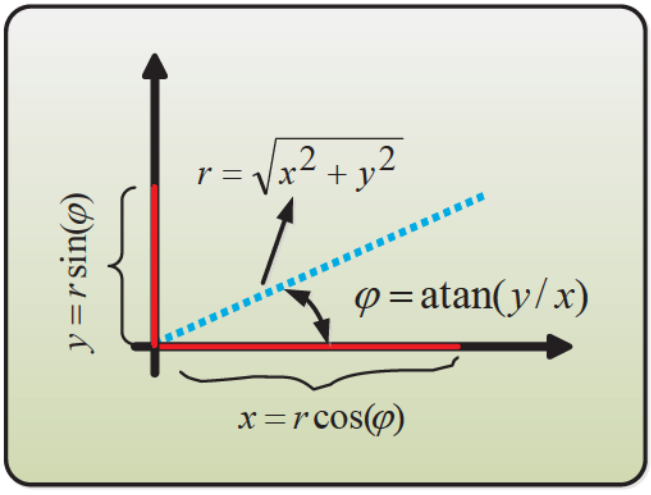


Figure 3.13: Polar and Cartesian representation.

where r is the distance from origin to the point, x represent Cartesian x-coordinate, and y represent Cartesian y-coordinate. φ is the relative angle to the zero axis. Cartesian

coordinates can be calculated from Polar coordinates:

$$\mathrm{x} = \mathrm{r}\cos(\varphi), \quad \mathrm{y} = \mathrm{r}\sin(\varphi), \tag{3.35}$$

For a given function $\mathrm{f}(\mathrm{r}, \varphi)$ in polar coordinates the relationship between derivatives in cartesian and polar coordinates is as follows:

$$\begin{aligned} \frac{\partial \mathrm{f}}{\partial \mathrm{x}} &= \frac{\partial \mathrm{f}}{\partial \mathrm{r}}\frac{\partial \mathrm{r}}{\partial \mathrm{x}} + \frac{\partial \mathrm{f}}{\partial \varphi}\frac{\partial \varphi}{\partial \mathrm{x}} = \cos(\varphi)\frac{\partial \mathrm{f}}{\partial \mathrm{r}} - \frac{\sin(\varphi)}{\mathrm{r}}\frac{\partial \mathrm{f}}{\partial \varphi}, \\ \frac{\partial \mathrm{f}}{\partial \mathrm{y}} &= \frac{\partial \mathrm{f}}{\partial \mathrm{r}}\frac{\partial \mathrm{r}}{\partial \mathrm{y}} + \frac{\partial \mathrm{f}}{\partial \varphi}\frac{\partial \varphi}{\partial \mathrm{y}} = \sin(\varphi)\frac{\partial \mathrm{f}}{\partial \mathrm{r}} + \frac{\cos(\varphi)}{\mathrm{r}}\frac{\partial \mathrm{f}}{\partial \varphi}. \end{aligned} \tag{3.36}$$

Defining $\partial f_x = \frac{\partial f}{\partial x}$ and the same for the rest of the variable, (3.36) can be rewritten as:

$$\begin{aligned} \partial \mathrm{f}_\mathrm{x} &= \cos(\varphi)\partial \mathrm{f}_\mathrm{r} - \frac{\sin(\varphi)}{\mathrm{r}}\partial \mathrm{f}_\varphi, \\ \partial \mathrm{f}_\mathrm{y} &= \sin(\varphi)\partial \mathrm{f}_\mathrm{r} + \frac{\cos(\varphi)}{\mathrm{r}}\partial \mathrm{f}_\varphi, \end{aligned} \tag{3.37}$$

which results in following transformation matrix:

$$\begin{bmatrix} \partial \mathrm{f}_\mathrm{x} \\ \partial \mathrm{f}_\mathrm{y} \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi)/\mathrm{r} \\ \sin(\varphi) & \cos(\varphi)/\mathrm{r} \end{bmatrix} \begin{bmatrix} \partial \mathrm{f}_\mathrm{r} \\ \partial \mathrm{f}_\varphi \end{bmatrix}. \tag{3.38}$$

The error covariance matrix R corresponding to PMU measurement errors in polar coordinates must also be transformed to cartesian coordinates. By definition of differentiability the incremental change $\Delta\mathrm{x}$ and $\Delta\mathrm{y}$ are given as:

$$\Delta \mathrm{x} = \mathrm{x}(\mathrm{r} + \Delta \mathrm{r}, \varphi + \Delta\varphi) - \mathrm{x}(\mathrm{r}, \varphi) \simeq \frac{\partial \mathrm{x}}{\partial \mathrm{r}}\Delta \mathrm{r} + \frac{\partial \mathrm{x}}{\partial \varphi}\Delta\varphi, \tag{3.39}$$

$$\Delta \mathrm{y} = \mathrm{y}(\mathrm{r} + \Delta \mathrm{r}, \varphi + \Delta\varphi) - \mathrm{y}(\mathrm{r}, \varphi) \simeq \frac{\partial \mathrm{y}}{\partial \mathrm{r}}\Delta \mathrm{r} + \frac{\partial \mathrm{y}}{\partial \varphi}\Delta\varphi, \tag{3.40}$$

resulting in the following general transformation:

$$\begin{bmatrix} \Delta \mathrm{x} \\ \Delta \mathrm{y} \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\mathrm{r}\sin(\varphi) \\ \sin(\varphi) & \mathrm{r}\cos(\varphi) \end{bmatrix} \begin{bmatrix} \Delta \mathrm{r} \\ \Delta\varphi \end{bmatrix} = [\mathrm{T}_\mathrm{r}] \begin{bmatrix} \Delta \mathrm{r} \\ \Delta\varphi \end{bmatrix}. \tag{3.41}$$

Based on definition of covariance matrix [144]:

$$\mathrm{R}(\mathbf{x}, \mathbf{y}) = \mathrm{Ex}[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{y} - \bar{\mathbf{y}})^\mathrm{T}], \tag{3.42}$$

where Ex indicate the expected value. $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$, represent the mean value of $\mathbf{x}$ and $\mathbf{y}$, respectively. Substituting (3.41) in (3.42) using the transformation matrix ($\mathbf{T}_\mathrm{r}$), the error covariance sub-matrix of PMU measurements in cartesian format can be driven as follows:

$$\begin{aligned} \mathbf{R}^\mathrm{C}_\mathrm{PMU} = \mathbf{R}(\Delta\mathbf{x}, \Delta\mathbf{y}) &= \mathrm{Ex}[(\mathbf{T}_\mathrm{r}\Delta\mathrm{r})(\mathbf{T}_\mathrm{r}\Delta\varphi)^\mathrm{T}] = \mathrm{Ex}[\mathbf{T}_\mathrm{r}\Delta\mathbf{r}\Delta\boldsymbol{\varphi}^\mathrm{T}\mathrm{T}_\mathrm{r}^\mathrm{T}] \\ &= \mathbf{T}_\mathrm{r}\mathrm{Ex}[\Delta\mathbf{r}\Delta\boldsymbol{\varphi}^\mathrm{T}]\mathbf{T}_\mathrm{r}^\mathrm{T} = \mathbf{T}_\mathrm{r}\mathbf{R}(\Delta\mathbf{r}\Delta\boldsymbol{\varphi})\mathbf{T}_\mathrm{r}^\mathrm{T} = \mathbf{T}_\mathrm{r}\mathbf{R}^\mathrm{P}_\mathrm{PMU}\mathbf{T}_\mathrm{r}^\mathrm{T}. \end{aligned} \tag{3.43}$$

$\bar{\Delta}\mathbf{x}$ and $\bar{\Delta}\mathbf{y}$ assumed to be zero.

Superscripts C and P refers to cartesian and polar format, respectively. Since the PMU measurements are assumed to have higher accuracy than SCADA measurements, more weight is assigned to them in measurement error covariance matrix:

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}^{\mathrm{C}}_{\mathrm{SCADA}} & 0 \\ 0 & \mathbf{R}^{\mathrm{C}}_{\mathrm{PMU}} \end{bmatrix}. \tag{3.44}$$

In case of sudden changes in the PMU measurements the accuracy of the estimation may degrade. By online tracking (as fast as measurement update rate) of the difference between two consecutive PMU measurements at the buses with PMU installations, the algorithm decides whether the network was in normal condition or not. The threshold for detecting contingency is set to 50% of previous measurement; it is assumed that if the next PMU measurement changes more than 50% of the previous value there is a contingency. The proposed method also can be applied for smaller thresholds. To handle the contingency effect on the state estimation process, a correction step is added to the algorithm. Nodal equations in a power network can be written as:

$$\begin{bmatrix} \mathbf{I}_{\mathrm{P}} \\ \mathbf{I}_{\mathrm{S}} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{\mathrm{PP}} & \mathbf{Y}_{\mathrm{PS}} \\ \mathbf{Y}_{\mathrm{SP}} & \mathbf{Y}_{\mathrm{SS}} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{\mathrm{P}} \\ \mathbf{V}_{\mathrm{S}} \end{bmatrix}, \tag{3.45}$$

where the subscripts P and S refer to buses with PMU measurement and SCADA measurement subsets, respectively. In general:

$$\Delta\mathbf{V}_{\mathrm{P}} = (\Delta\mathbf{Y}_{\mathrm{PP}})^{-1}(\Delta\mathbf{I}_{\mathrm{P}} - \Delta\mathbf{Y}_{\mathrm{PS}}\Delta\mathbf{V}_{\mathrm{S}}). \tag{3.46}$$

From (A.3), it can be concluded that changes in $\Delta\mathbf{V}_{\mathrm{S}}$ is proportional to changes in $\Delta\mathbf{V}_{\mathrm{P}}$. Therefore, in the case of sharp or sudden changes in the PMU measurements, only $\mathbf{Y}_{\mathrm{PS}}$ and $\mathbf{Y}_{\mathrm{PP}}$ needed to be updated using online power flow. These matrices are too small compare to $\mathbf{Y}_{\mathrm{SS}}$. Fig. 3.14 shows the block diagram of the entire data collation process.

3.7.2 Extraction of Parallelism

In the proposed MPDSE method several aspects of parallelism are combined to utilize the full capability of GPUs as efficiently as possible. First, all initialization and data collation are done on the CPU. After that all of the data are transferred to the GPU for executing the MPDSE algorithm. The following types of parallelism are used in this work:

- Task parallelism- in this level, the traditional serial algorithm is converted into various smaller and independent tasks which can be solved in parallel. All of the independent tasks in the three main steps of EKF are calculated in parallel to accelerate the algorithm. In the parameter identification, $\mathbf{a}_{\mathrm{k}}$, $\boldsymbol{\lambda}_{\mathrm{k}}$ and $\mathbf{Q}_{\mathrm{k}}$ do not rely on each other's result, so they are calculated in parallel to accelerate the algorithm. In the state prediction stage $\tilde{\mathbf{x}}_{\mathrm{k+1}}$, $\tilde{\rho}_{k+1}$, $\mathbf{G}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are parallelizable. Finally in the state filtering step, $\hat{\mathbf{x}}_{\mathrm{k+1}}$ and ρ_{k+1} can be calculated simultaneously.

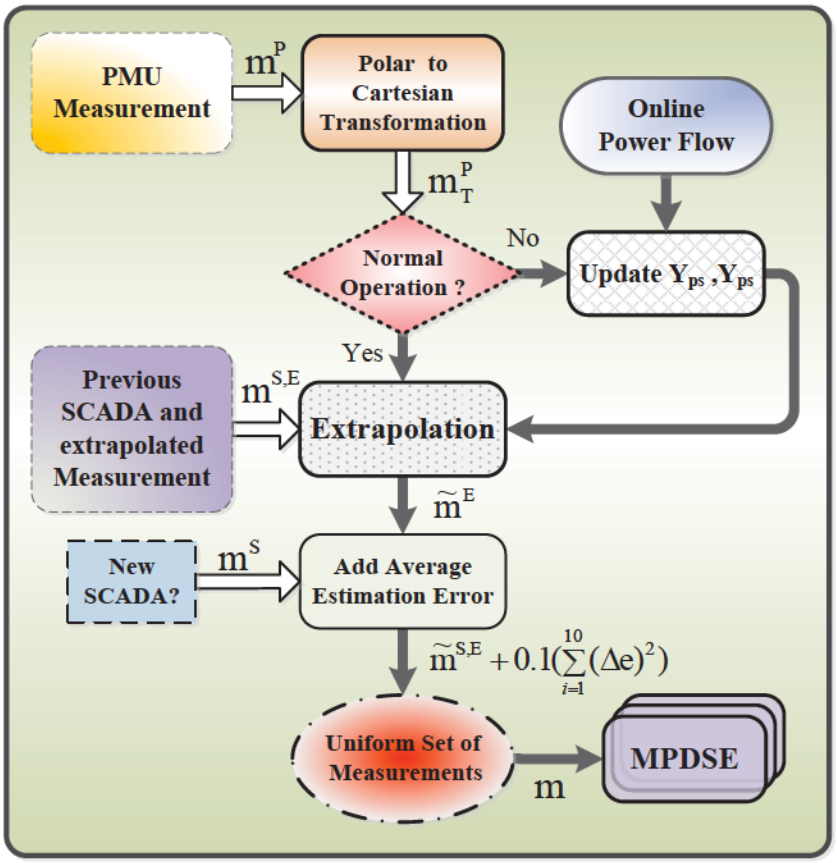


Figure 3.14: Data collation process flowchart.

- Data parallelism- this level employs the fine-grained type of parallelism that can be used on the SIMD-based architectures such as GPUs for the basic computations in the algorithm. Generally, matrix-vector and matrix-matrix products are time consuming for large data-sets. There are several independent *for* loops in the implementation of each matrix-matrix and matrix-vector products which make them the best candidates for parallelization utilizing GPU threads. Assigning each iteration in a loop to individual threads, the task can be executed in parallel by converting into a kernel. In the MPDSE algorithm, the computation of $\mathbf{a}_k$ for state prediction and $\mathbf{G}(\mathbf{x})$, $\mathbf{K}$, $\boldsymbol{\lambda}$, $\boldsymbol{\rho}$ and $\mathbf{H}^T(\mathbf{x})\mathbf{R}^{-1}\mathbf{H} + \tilde{\rho}^{-1}$ for state filtering can take a long time to complete even on CPU clusters. These separate tasks are composed of matrix-matrix and matrix-vector product or summations which can be assigned to an individual kernel to run in parallel. Each kernel is responsible for the calculation of that specific task. As the number of independent threads is a lot more than the CPU cores, this type of parallelization is not possible on the CPU.

- Parallelism in linear solver- Solution of $\Delta(\mathbf{x})$ by inversion of $\mathbf{G}(\mathbf{x})$ is considerably expensive due to the sheer size of the inverted matrix. Two alternatives, LU de-

composition as a direct method and Preconditioned Conjugate Gradient (PCG) as an iterative method were used in this work. For iterative solvers, the preconditioner is the most challenging part to parallelize. Coarse grained or task parallelism is difficult to achieve on LU and Cholesky factorization due to inherent sequentiality. However, underlying implementation of these algorithms (vector updates, inner products, matrix vector products) takes advantage of the data parallelism. So, these tasks are done as a combination of sequential and parallel computations.

Sparse matrix-vector multiplication and sparse triangular solve is used for GPU implementation using cuSPARSE library [156]. Fig. 3.15 shows the overall MPDSE flowchart, and Table 3.2 summarizes the sequential and parallel variables.

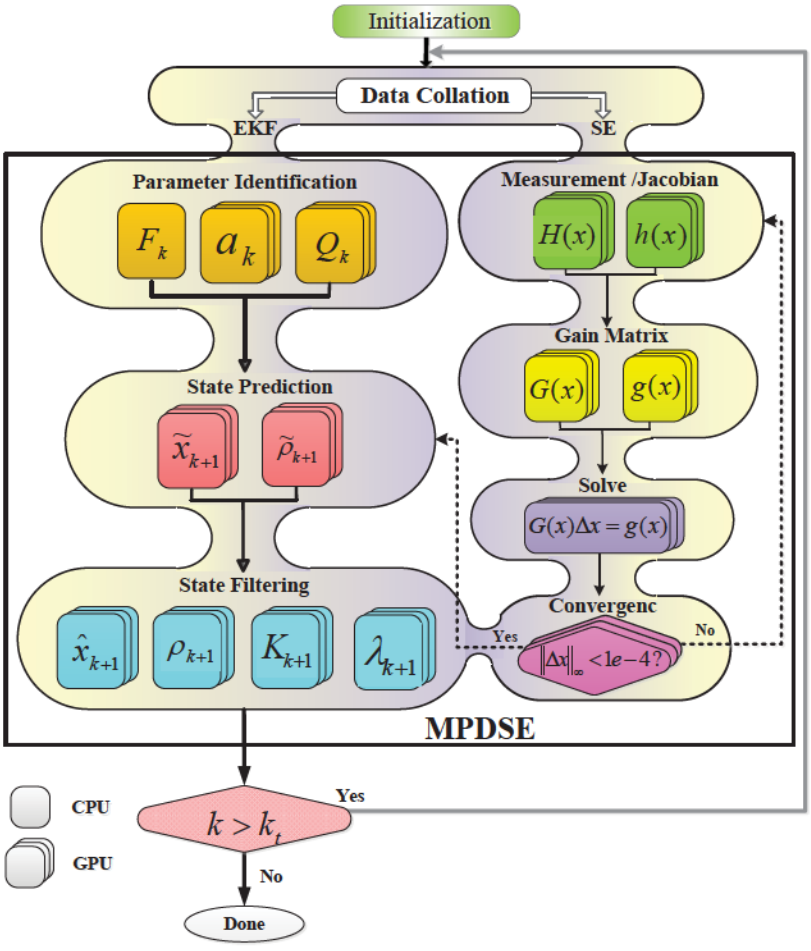


Figure 3.15: MPDSE operation flowchart.

Table 3.2: Summary of Sequential and Parallel Variable in MPDSE

Task	Sequential	Task Parallel	Data Parallel
Initialization	Y, R,...	–	–
Data Collation	$\tilde{m}^E$	–	–
Parameter Identification	F	a, λ, Q	a, λ, Q, ξ
State Prediction	H, h	$\tilde{x}$, $\tilde{\rho}$	H, h,$\tilde{x}$, $\tilde{\rho}$
State Filtering	–	$\hat{x}$, ρ	$\hat{x}$, ρ, λ K
Linear solver	LU, PCG	–	G, g, Δx, LU, PCG

3.7.3 Experimental Results

The same as SSE implementation, to evaluate accuracy and efficiency of the parallel DSE algorithms, experiments were conducted based on two separate simulation codes: multi-thread CPU-based code in C++, and a massive-thread GPU-based code. The results of state estimation were compared utilizing both iterative PCG and direct LU decomposition methods for the linear solver. Buses with PMU measurements and SCADA measurements in the IEEE 39-bus system are shown in Fig. 3.16. The block diagram for the MPDSE test procedure is shown in Fig. 3.17.

The simulations were done using the test data sets listed in Table 3.1, with a tolerance of 10^{-4} for convergence of the estimated parameters.

In the proposed GPU implementation most of the computational steps are moved to GPU to avoid any unnecessary communication between host and device. Fig. 3.21 illustrates the implementation of the MPDSE on the GPU. The OpenMP standard was utilized to develop the multi-core DSE code. All the *for* loops and parallel sections were assigned to separate threads and cores. Each thread is responsible for specific portion of the tasks to execute on that core. The entire DSE task was divided into several subsets to distribute an equal workload among the threads which equals the number of the cores. Fig. 3.22 shows a sample of the multi-thread CPU code.

Accuracy Analysis of the GPU-based MPDSE

The performance of the proposed MPDSE method, was evaluated under both normal and contingency conditions. The estimation error in Case 1 for a temporary fault at Bus 10 at t=10 min for a duration of 0.1 sec. are shown in is shown in Fig. 3.18. It is clear that the proposed MPDSE can accurately track the system dynamics. The average error of 0.07 p.u. for voltage magnitudes and 2.5 deg. for phase angles during the first 5 minutes was due to the fact that the estimator needed to predict missing SCADA measurements at the first level (Fig. 3.12).

Since at the beginning there was not enough history data to be used for prediction, the

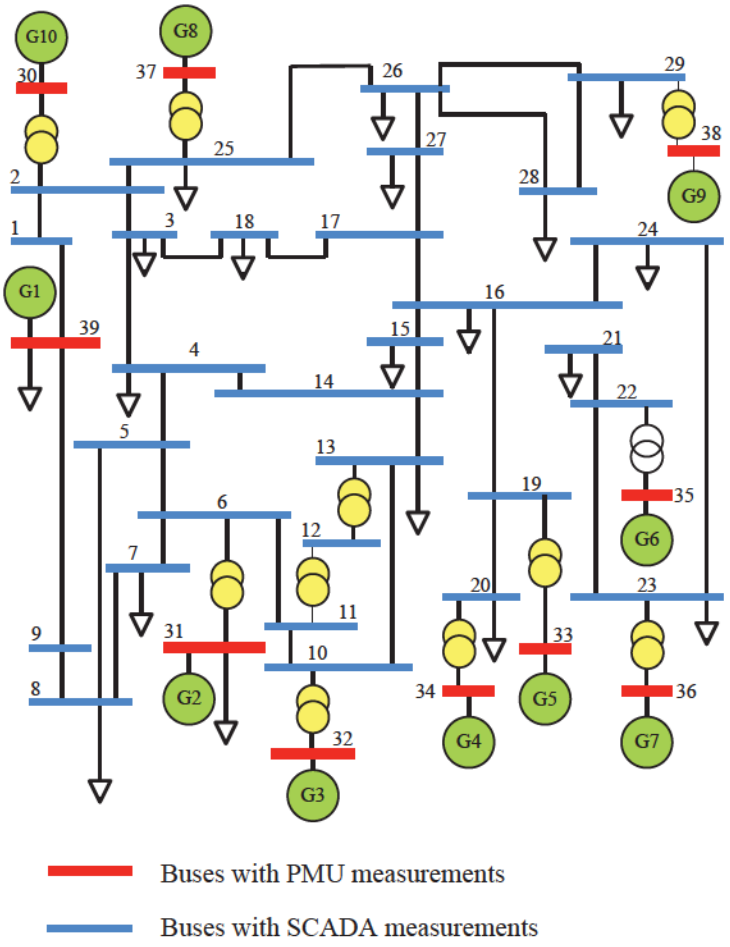


Figure 3.16: IEEE 39-bus power system used to build the large test cases.

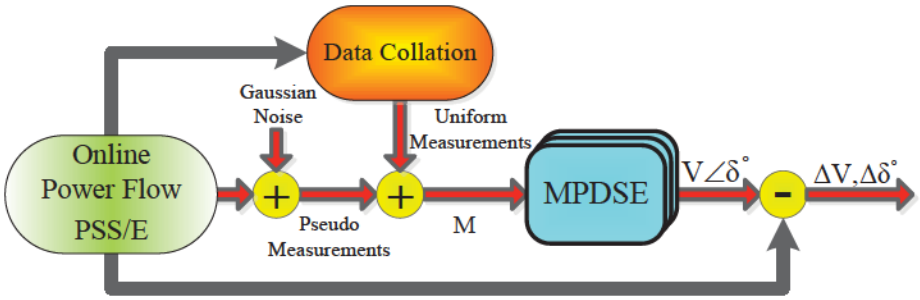


Figure 3.17: MPDSE test procedure.

error of estimation was large. However, in a long term simulation neglecting the effect of incomplete historical data the average estimation error for voltage magnitudes and phase angles was 0.002 p.u. and 0.05 deg., respectively. A snapshot of the estimation error at buses numbers 10, 11, 13 and 32 is provided in Fig. 3.19. In addition, the performance of

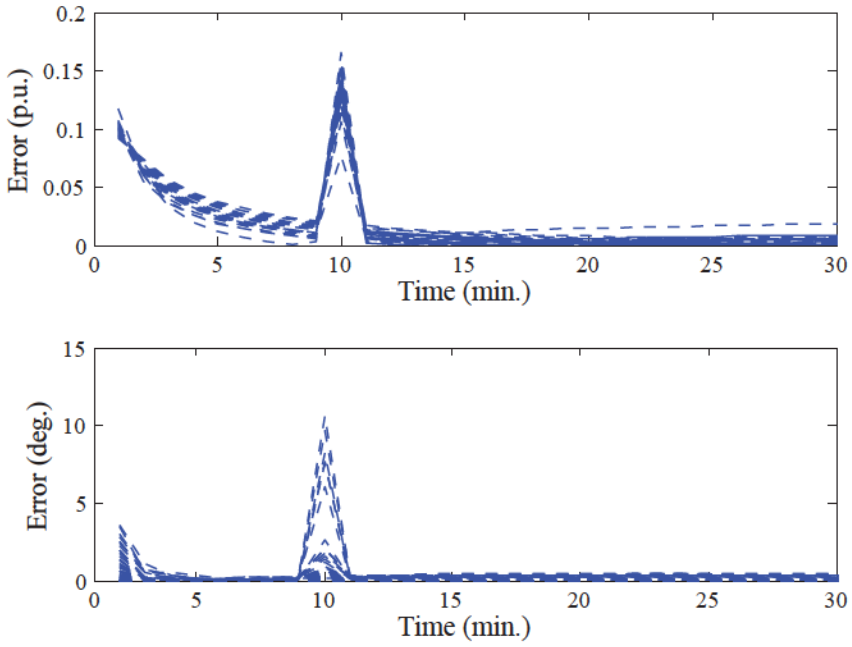


Figure 3.18: Estimation errors in MPDSE for Case 1 compared to PSS/E® under fault conditions.

the proposed state estimation method is tested for various number of PMU installations. The normalized Euclidian norm of the state estimation is defined as a factor to evaluate the accuracy using:

$$\mathbf{x}_{\text{EN}} = \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\sqrt{\dim(\mathbf{x})}}, \tag{3.47}$$

Table 3.3: DSE Error Norm (Eq. 3.47) for Different Percentage of PMU Installation

Case Case	**Normalized Error Norm in Voltage Magnitude**			**Normalized Error Norm in Phase Angle**		
	$N_P = 10\%N_T$	$N_P = 20\%N_T$	$N_P = 40\%N_T$	$N_P = 10\%N_T$	$N_P = 20\%N_T$	$N_P = 40\%N_T$
1	0.004	0.0027	0.0015	0.59	0.37	0.23
2	0.0038	0.0028	0.0013	0.57	0.39	0.25
3	0.0036	0.0027	0.0015	0.55	0.32	0.29
4	0.0041	0.0029	0.0016	0.56	0.34	0.24
5	0.0033	0.0028	0.0017	0.58	0.36	0.27
6	0.0039	0.003	0.0017	0.59	0.37	0.28
7	0.0041	0.0026	0.0016	0.57	0.34	0.21
8	0.0036	0.0029	0.0015	0.56	0.38	0.28

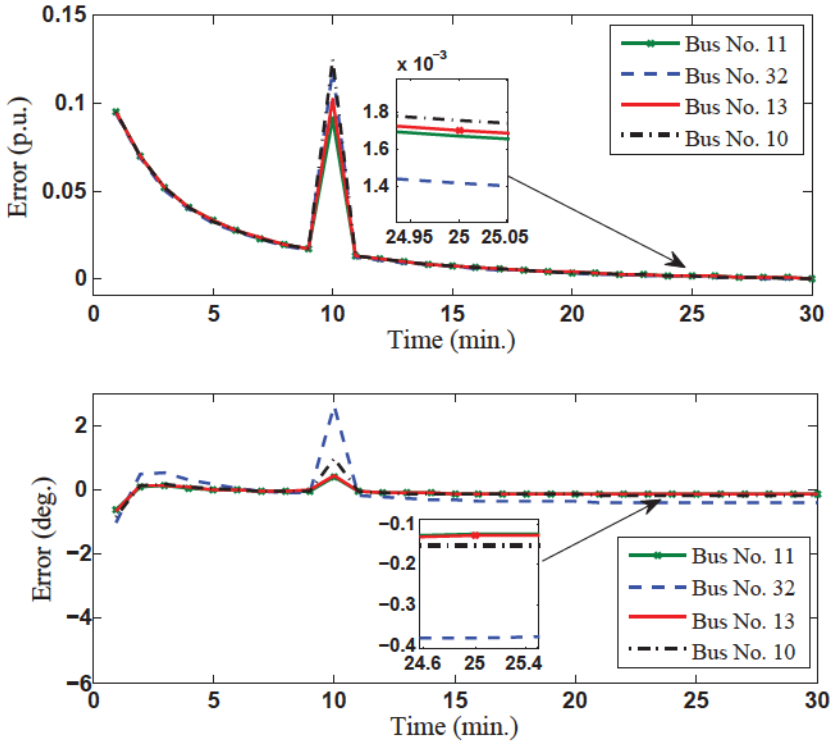


Figure 3.19: Snapshot of estimation error for Case 1 at buses numbers 10, 11, 13 and 32.

where x_{EN} is the normalized Euclidian norm of the estimation error, and dim(x) is the dimension of vector x. x and $\hat{x}$ are vector of true states and estimated states, respectively. Table 3.3 shows the accuracy index for both voltage magnitude and phase angle using PMU installation (N_P) at 10%, 20% and 40% of the total system buses (N_T). N_P is rounded to the next larger number. As it is shown the results are accurate for all of the case studies and the accuracy is increased as the percentage of PMU measurements increased.

Speed-up and Complexity Analysis

The total execution time under contingency scenario including sequential data collation, contingency check, admittance matrix update, and state estimation is reported in Table 3.4. The results obtained using double precision (64 bit) format for both CPU (T_{Ex}^{CPU}) and GPU (T_{Ex}^{GPU}) codes as the system size increased. The execution time for GPU-based program includes both execution and communication time. An incomplete Cholesky PCG iterative algorithm was used to condition the gain matrix which reduced the number of iterations in each solution. The condition numbers before and after preconditioning are shown in Table 3.4. The results also show that the execution time reduced by approximately 20%-50% using PCG compared to the LU decomposition method. The reason is that direct methods calculate an exact solution on a single iteration, so that they deal with large size matrices which take more time to be solved; however, the iterative method starts from

an initial guess and improves the solution in each iteration. Fig. 3.20 show the results of comparison between the CPU and GPU simulations along with the speed-up of the parallel code for both iterative and direct solvers. As can be seen, PCG converged faster than the LU decomposition algorithm but the speed-up ($S_p = T_{Ex}^{CPU}/T_{Ex}^{GPU}$) using GPU is almost the same for both methods. The reason is that the execution times for both CPU-based PCG and GPU-based PCG experience a similar drop in each case study. Therefore, the overall speed-up is almost the same. The accuracy was mostly the same for both linear solvers, so the details are omitted from Table 3.4.

Table 3.4: Summary of Overall Estimation Time for Multi-thread and Massive Thread DSE Under Contingency Condition

Case	No. of buses	Cond. no. CG	Cond. no. PCG	T_{Ex}^{CPU} LU	T_{Ex}^{GPU} LU	T_{Ex}^{CPU} PCG	T_{Ex}^{GPU} PCG	S_p LU	S_p PCG
1	39	2.4E+02	1.6E+01	0.49s	0.29s	0.38s	0.19s	1.69	2
2	78	9.7E+03	6.3E+02	1.16s	0.59s	0.83s	0.39s	1.96	2.12
3	156	3.9E+04	1.1E+03	4.5s	2.04s	3.1s	1.1s	2.2	2.81
4	312	8.9E+04	1.8E+03	19.2s	6.8s	13.4s	4.3s	2.8	3.02
5	624	2.8E+05	4.9E+04	45.3s	12.5s	38.8s	9.8s	3.6	3.9
6	1248	3.6E+06	2.6E+05	146s	26s	109.1s	20s	5.6	5.4
7	2496	2.4E+07	1.3E+06	369s	40s	290.4s	33s	9.2	8.8
8	4992	4.5E+08	9.4E+06	932s	59s	722.5s	48s	15.9	15.05

Table 3.5: State Estimation Execution Time

Case	T_{SE}^{CPU} LU	T_{SE}^{GPU} LU	T_{SE}^{CPU} PCG	T_{SE}^{GPU} PCG	S_p^{SE} LU	S_p^{SE} PCG
1	0.19s	0.13s	0.15s	0.06s	1.46	2.5
2	0.45s	0.24s	0.31s	0.12s	1.87	2.6
3	1.46s	0.59s	1.1s	0.33s	2.4	3.3
4	6.4s	2.05s	4.5s	1.2s	3.1	3.7
5	16.4s	3.4s	12.4s	2.6s	4.8	4.8
6	52.3s	7.2s	36.1s	5.4s	7.2	6.6
7	128s	9.8s	99.5s	8.5s	13	11.7
8	323s	15.91s	249.1s	12.9s	20.3	19.3

The portion of execution time related to actual dynamic state estimation process (parameter identification, state prediction and state filtering) along with the actual state estimation process speed-up is reported in Table 3.5. As the results show state estimation

takes approximately 25%-30% of the total execution time of massive-thread DSE and between 35%-40% of the total execution time of the multi-thread DSE.

To analyze the algorithms complexity, their efficiencies are expressed as functions of the problem size. Considering S_c as the system scale, by curve fitting the closest growth rate function ($GR = T_{Ex}(S_c)$) for the CPU-based and the GPU-based algorithms are calculated as $S_c * log_2(S_c)$ and $\sqrt{S_c} * log_2(\sqrt{S_c})$, respectively. The growth rate functions for CPU (GR^{CPU}) and GPU (GR^{GPU}) are plotted in Fig. 3.20.

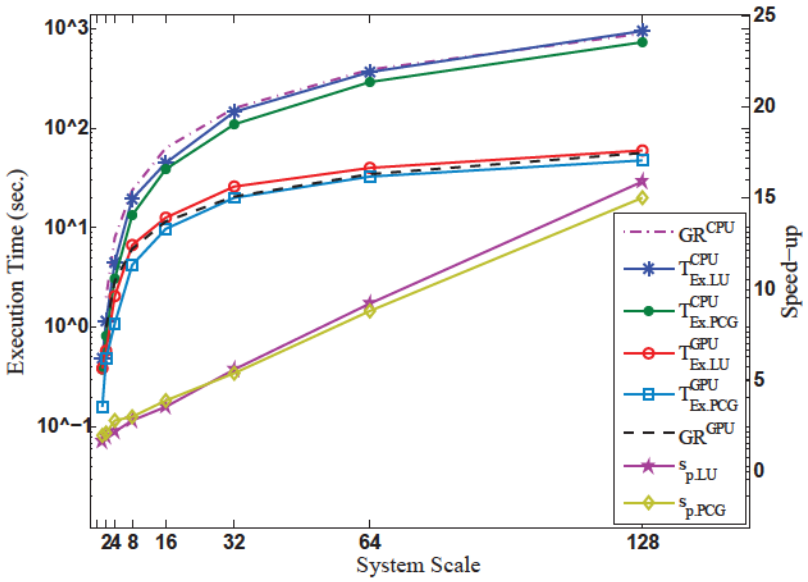


Figure 3.20: Execution time ($T_{Ex.}$) and speed-up (S_p) comparisons of multi-thread and massive-thread DSE along with growth rate functions.

As the size of the problem increases, the required time for execution increased proportional to these functions which proves that the speed-up will increase with growing system sizes. It can be seen that the CPU-based algorithm follows a higher order complexity compared to the GPU-based algorithm. The complexity reduction is the main advantage afforded by the GPU for MPDSE.

3.8 Discussion

Although GPUs are well suited for large-scale data-parallel processing, writing efficient code for them is not without its difficulties. One of the most important bottleneck in parallel GPU programming is the overhead in data transfer between host and device. The size and frequency of data transfers can create a bottleneck which significantly impacts the execution time of an algorithm. This fact is considered in both proposed methods by moving all the time consuming steps to GPU to reduce the data transfer which resulted in significant speedup for both SSE and DSE applications.

As shown in Table 3.1 and Table 3.4, the advantage of utilizing GPU for parallelization is significant when the size of the system is increased. One explanation is that for small size of data the communication overhead between the host and device supersedes the execution time on the CPU. With growing system sizes, the CPU is barely able to handle the computation tasks in a reasonable time. With growing system sizes less time is spent on state estimation. The reason is that, as the size of the system grows the parallel portion of the program expands faster than the serial portion, while underscore GPU's advantages. These results are not unique due to the fact that the programming structure is one of the most important factors which affects the processing time. Therefore, a different programming style may lead to faster results; nevertheless, the speed-up would still be valid for increasing system sizes although with a slightly lower numerical value.

In terms of accuracy, the small differences compared to PSS/E® results are justifiable considering the fact that the order of block execution in each GPU grid is undefined in kernel definition. Therefore, it leads to slightly different results if different CUDA blocks perform calculations on overlapping portions of data.

3.9 Summary

The application of parallel processing for static/dynamic state estimation is motivated by the desire for faster computation for online monitoring of the system behavior. The approach proposed in this chapter investigates the process of accelerating the static/dynamic for large-scale networks. In the first part, WLS-based state estimation was explored using massively parallel graphic processing unit.

In the second part, a data collation method was proposed where PMU devices are installed at a subset of buses, and the remaining measurements were extrapolated employing the exponential moving average method. A massively parallel dynamic state estimation process was formulated on the GPU along with a multi-threaded CPU implementation.

In both methods, for a fair comparison exactly the same algorithm was used on both CPU and GPU. Numerical experiments in this work proved that successful parallelization utilizing iterative and direct linear solver results in a notable reduction in execution time. The results verify the accuracy of the proposed GPU-based estimators under both normal and contingency conditions.

Algorithm: GPU Implementation of EKF-based DSE

Given an initial $\tilde{x}_0$

Calculate

$F_k = \alpha(1+\beta)I_{ind}$

$a_k = (1+\beta)(1-\alpha)\tilde{x}_k + \ldots$ — 3*(2n-1) summations → GPU

$\gamma_k = \alpha\hat{x}_k + (1-\alpha)\tilde{x}_k$ — 2*(2n-1) summations → GPU

$\xi_k = \beta + (\gamma_k - \gamma_{k-1}) + \ldots$

$Q_k = Q_k\sqrt{\ldots}$ — 6 M-M P[I], 2(2n-1)² VA → GPU

Until termination do

while $\Delta x \geq \varepsilon$ do

for i=1,… until convergence

Calculate

$g(x) = H^T R^{-1} H(m - h(x))$ — 1 M-M P, 2 M- V P[II] → GPU

$G(x) = H^T R^{-1} H + \tilde{\rho}^{-1}$ — 2 M-M P[III] → GPU

Solve $G(x)\Delta(x) = g(x)$ — LU/PCG → GPU

Update $x_i = x_{i-1} + \Delta(x)$ — (2n-1) VA → GPU

end for

end while

$\tilde{x}_{k+1} = F_k\hat{x}_k + a_k$ — (2n-1) VA, (2n-1) M-V P → GPU

$\tilde{\rho}_{k+1} = F_k\tilde{\rho}_k F_k^T + Q_k$ — 2 (2n-1)² M-M P, (2n-1)² VA → GPU

Update

$\hat{x}_{k+1} = \tilde{x}_{k+1} + K_{k+1}\cdots$ — 1 M-M P, 2 VA[IV] → GPU

$K_{k+1} = \tilde{\rho}_{k+1}H_{k+1}^T \ldots.$ — 2 (2n-1) (2n + 2m+1) M-M P → GPU

$\rho_{k+1} = (I - K_{k+1}H_{k+1})\tilde{\rho}_{k+1}$ — 2 (2n-1)² M-M P, (2n-1)² VA → GPU

Update $k = k+1$

end

Matrix-matrix product (M-M P), Vector add (VA), Matrix-vector product (M-V P).

[I] $2(2n-1)^2+4(2m+2n+1)(2n-1)$ operations

[II] $(2m+2n+1)(2n-1)+2(2n-1)$ operations

[III] $(2n-1)(2n-1)+(2m+2n+1)(2n-1)$ operations

[IV] $(2n-1)(2n+2m+1), (2n-1)+(2n+2m+1)$ operations

Figure 3.21: Pseudo code for GPU implementation of MPDSE.

Algorithm: OpenMP-based algorithm of EKF-based DSE

#pragma omp_set_num_threads(4)

Given an initial $\tilde{x}_0$

If $k \geq k_{final}$ do

#pragma omp parallel sections

#pragma omp section

Calculate $F_k = \alpha(1+\beta)I_{ind}$

#pragma omp section

#pragma omp parallel shared (…), private (…)

Calculate $a_k = (1+\beta)(1-\alpha)\tilde{x}_k + \ldots$

#pragma omp section

Calculate
$$\gamma_k = \alpha\hat{x}_k + (1-\alpha)\tilde{x}_k$$
$$\xi_k = \beta + (\gamma_k - \gamma_{k-1}) + \ldots$$

#pragma omp parallel shared

Calculate $Q_k = Q_k\sqrt{\ldots}$

while $\Delta x \geq \varepsilon$ do

#pragma omp parallel sections

#pragma omp section

Calculate
$$g(x) = H^T R^{-1} H(m - h(x))$$
$$G(x) = H^T R^{-1} H + \tilde{\rho}^{-1}$$

#pragma omp parallel for

Solve $G(x)\Delta(x) = g(x)$

#pragma omp parallel for reduction (+: …)

Update $x_i = x_{i-1} + \Delta(x)$

end while

#pragma omp parallel sections

#pragma omp section

Calculate
$$\tilde{x}_{k+1} = F_k\hat{x}_k + a_k$$
$$\tilde{\rho}_{k+1} = F_k\tilde{\rho}_k{F_k}^T + Q_k$$

#pragma omp parallel for

Update
$$\hat{x}_{k+1} = \tilde{x}_{k+1} + K_{k+1}\ldots$$
$$K_{k+1} = \tilde{\rho}_{k+1}{H_{k+1}}^T \ldots.$$
$$\rho_{k+1} = (I - K_{k+1}H_{k+1})\tilde{\rho}_{k+1}$$

Update $k = k+1$

end

Figure 3.22: Pseudo code for quad-core CPU implementation of DSE.

4

Relaxation-based Dynamic State Estimation: Multi-GPU Implementation

4.1 Introduction

Massive amounts of data generated in large-scale grids along with the wide usage of P-MUs poses a formidable challenge for online monitoring of power systems. DSE which is a prerequisite for normal operation of power systems involves the time-constrained solution of a large set of equations which requires significant computational resources. So for accurate monitoring of the system dynamic behaviour it is necessary to generate faster techniques to alleviate computational burden.

In contrast with most available approaches for DSE which only accelerated SSE process, the objective of this chapter is to explore DSE using EKF in large-scale power systems utilizing detailed synchronous generator modeling. Utilizing highly detailed models complicates the state estimation problem resulting in a high computational burden. A heterogeneous parallel multi-GPU and multi-core CPU implementation of large-scale DSE based on an accurate and robust relaxation method is presented to estimate both generator (dynamic) and network (static) states of the power system. Coarse-grained parallelism using relaxation method along with its implementation on a multi-GPU computational server where each of the individual GPUs has a fine-grained parallel architecture enables significant acceleration of the DSE process.

4.2 Formulation and State-space Model

Joint dynamic and static state estimation for a multi-machine power system can be mathematically described as follows:

$$\begin{cases} \mathrm{f}(\dot{\mathbf{x}}_\mathrm{g}, \mathbf{x}_\mathrm{g}, \mathrm{t}, \mathbf{u}) = 0 & \mathbf{x}_\mathrm{g}(\mathrm{t}_0) = \mathbf{x}_{\mathrm{g}0} \\ \mathrm{g}(\mathbf{x}, \mathrm{t}, \mathbf{u}) = 0 & \\ \mathrm{h}(\mathbf{x}_\mathrm{n}, \mathbf{m}, \varepsilon) = 0 & \mathbf{x}_\mathrm{n}(\mathrm{t}_0) = \mathbf{x}_{\mathrm{n}0} \end{cases} \tag{4.1}$$

where $\mathbf{x}$ is the vector of state variables including $\mathbf{x}_\mathrm{g}$, dynamic state of generator and, $\mathbf{x}_\mathrm{n}$, static state of the network. $\mathbf{x}_0$ is the initial values of state variables. $\mathbf{f}(.)$ describe the nonlinear dynamic behaviour of the generators, $\mathbf{g}(.)$ model the output function, and $\mathbf{h}(.)$ is the nonlinear function of network measurement.$\mathbf{u}$ and $\mathbf{m}$ represent the output vector and network measurements vector, respectively. t represents the simulation time. For a system with l generators, and n buses, there are $9\times l+2n$ elements in vector $\mathbf{x}$: 9 states per generator, and n voltage magnitudes, n phase angles.

One of the widely used method for detailed modeling of the synchronous generator is the Parks equations with an individual dq reference frame fixed on the generator's field winding [151]. The ninth-order model used in this work include automatic voltage regulator (AVR) and power system stabilizer (PSS) which includes two windings on the d-axis (one excitation field and one damper) and two damper windings on the q-axis. Although the excitation system increases the number of differential equations and complexity of the model, it increase the accuracy of the results. The details of the synchronous generator model are provided in Appendix A.

The vector of state variables $\mathbf{x}_\mathrm{g}$ for a single synchronous generator is given as:

$$\dot{\mathbf{x}}_\mathrm{g} = [\delta, \Delta\omega, \psi_\mathrm{fd}, \psi_\mathrm{1d}, \psi_\mathrm{1q}, \psi_\mathrm{2q}, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]^\mathrm{Tr}, \tag{4.2}$$

where δ and $\Delta\omega$ represent vector of rotor speed and angle, respectively. $\psi_\mathrm{fd}, \psi_\mathrm{1d}, \psi_\mathrm{1q}, \psi_\mathrm{2q}$ shows vector of rotor flux linkages and $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ are vector of exciter voltages. Tr represent the transpose of matrix.

In order to solve the system of equation, following three steps are required:

Step 1. The continuous-time differential equations are discretized using the Trapezoidal integration method which results in a new set of non-linear algebraic equations:

$$\mathbf{x}_\mathrm{g}^{\mathrm{t}+\tau} - \mathbf{x}_\mathrm{g}^{\mathrm{t}} = \frac{\tau}{2}[\mathbf{f}(\mathbf{x}_\mathrm{g}, \mathrm{t}+\tau, \mathbf{u}) - \mathbf{f}(\mathbf{x}_\mathrm{g}, \mathrm{t}, \mathbf{u})], \tag{4.3}$$

where τ is integration time-step.

Step 2. In second step the nonlinear algebraic equations are linearized which results in following:

$$\mathbf{x}_g^{t+\tau} = \mathbf{F}_{\mathbf{x}_g^t}\mathbf{x}_g^t + \boldsymbol{\eta}^t,\ \boldsymbol{\eta}^t \sim \mathrm{N}(0, \mathbf{Q}^t), \tag{4.4}$$

where $\mathbf{F}_{\mathbf{x}_g^t} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}_g^t}\,|_{\mathbf{x}_g^t}$ represents the 9×9 state transition between two time steps. η and Q are 9×1 linearization error, and 9×9 error covariance matrix, respectively.

Step 3. The resulting linear algebraic equations are solved using iterative method to obtain the system states.

The network model and nonlinear measurement function which was earlier explained in detail in Section 3.2.1. can be rewritten as:

$$\mathbf{x}_n^{t+\tau} = \mathbf{G}(\mathbf{x}_n)^{-1}\mathbf{g}(\mathbf{x}_n) + \mathbf{x}_n^t, \tag{4.5}$$

4.3 Relaxation Method

Relaxation-based approaches partition the system into a number of subsystems based on either the system equations or component connectivity which reduce the complexity [98, 153]. It facilitates the parallel solution of the small subsystems, and the exchange of computational data between them. This method can be used at different levels of equations by braking the system into subsystems in a way that the variable inside of each subsystem (local variables) are strongly interdependent while the dependency between variable in two different subsystems (local and global variables) is weak enough to ignore their interconnection. The relaxation method was the first attempt to exploit both space and time parallelism [154].

4.3.1 Space Parallelism

This parallel approach partitions the original system into subsystems and distributes them among the parallel processors. It usually addresses the task-level parallelism in which serial algorithms are converted into various smaller and independent tasks that may be solved in parallel. The obvious part that parallelism can be exploited in is the solution of linear algebraic equations. As an example consider (4.4) and (4.8) which can be decoupled and solved with Gauss-Jacobi method.

$$\mathbf{x}_g^{k+1} = \mathbf{F}(\mathbf{x}_g^k)\mathbf{x}_g^k + \boldsymbol{\eta}^k, \tag{4.6}$$

$$\mathbf{x}_n^{k+1} = \mathbf{G}(\mathbf{x}_n^k)^{-1}\mathbf{g}(\mathbf{x}_n^k) + \mathbf{x}_n^k, \tag{4.7}$$

where k is the iteration index. Equation (4.6) and (4.7) can be solved to update x_g and x_n at each time-step. Therefore, the work associated with each time-step can be distributed among the parallel processors and run simultaneously.

4.3.2 Time Parallelism

This technique concurrently solves many time-steps by dividing simulation time into a series of blocks that each of them contains a number of steps which lead to the solution of the system. In the parallel-in-time method the vector x is determined for T_s time-steps simultaneously, where T is the number of time-steps for which the output results are required. As an example consider (4.3) which can be rewritten as follows for a set of T equations:

$$\begin{aligned} \mathbf{x}_g^1 - \mathbf{x}_g^0 &= \frac{\tau}{2}[\mathbf{f}_1 + \mathbf{f}_2], \\ \mathbf{x}_g^2 - \mathbf{x}_g^1 &= \frac{\tau}{2}[\mathbf{f}_2 + \mathbf{f}_1], \\ &\vdots \\ \mathbf{x}_g^{Ts} - \mathbf{x}_g^{T-1} &= \frac{\tau}{2}[\mathbf{f}_T + \mathbf{f}_{T-1}], \end{aligned} \tag{4.8}$$

where the superscript denotes the individual time-step.

Overall, relaxation has following main advantages over existing parallel-processing method:

- It is an inherently parallel method.
- Each subsystem can be solved independently.
- Subsystems become smaller than the original large system, it takes less time to solve.
- Different levels of accuracy for modeling in each subsystem can be used.

The same as other parallel computation algorithms, the preliminary step in utilizing relaxation method is to decompose the problem into smaller tasks that can be distributed among several processors. After decomposition, each subsystem will be solved independently. In every iteration each subsystem is solved for its local variables while other subsystems are considered constant or relaxed during the time-step. Each subsystem uses the previous value of other subsystems as a guess for its new iteration. At the end of each iteration the global variables of all subsystems are exchanged for the next time-step and this process is repeated until convergence is gained. The parallelism inherent in the relaxation method, offers a coarse grained parallelization as a top level algorithm which should be implemented before using a numerical method for solving the system of equations. Figure 4.1 gives an example of Gauss-Seidel relaxation in different level of equations.

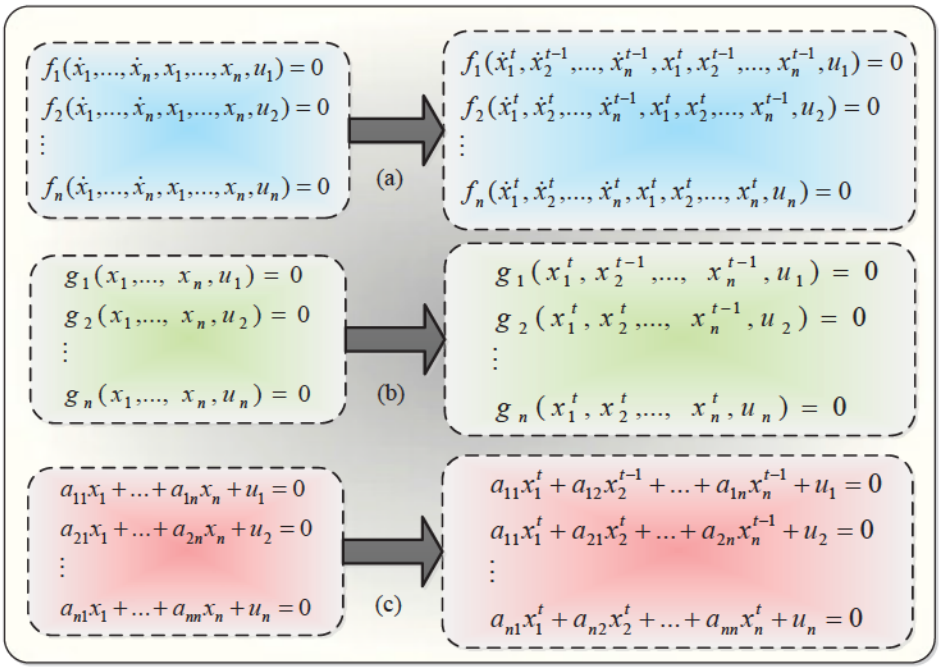


Figure 4.1: Gauss-Seidle relaxation method applied in different level of equations: (a) differential equation, (b) non-linear equation, (c) linear equation.

4.4 Domain Decomposition

Any technique that divides a system of equations into several subsets which can be solved individually can be classified as domain decomposition method. The subsystem is a subset of system variables. Since subsystems can be solved simultaneously decomposition techniques are suitable for parallel hardware architectures.

There are two main approaches for domain decomposition: overlapping and non-overlapping sub-domains. In cases that after domain decomposition subsystems have common states and need to exchange data it is called overlapping decomposition, otherwise it is non-overlapping decomposition.

Consider a power network that is decomposed into two parts as shown in Fig. 4.2. As an example, the objective function for static WLS state estimation can be formulated as:

$$\mathbf{J}(\mathbf{x}) = \underset{x_1,x_2}{\operatorname{argmin}} \begin{cases} [\mathbf{m}_1 - \mathbf{h}_1(\mathbf{x}_1)]^{\mathrm{T}}\mathbf{R}_1^{-1}[\mathbf{m}_1 - \mathbf{h}_1(\mathbf{x}_1)] + \\ [\mathbf{m}_2 - \mathbf{h}_2(\mathbf{x}_2)]^{\mathrm{T}}\mathbf{R}_2^{-1}[\mathbf{m}_2 - \mathbf{h}_2(\mathbf{x}_2)] + \\ [\mathbf{m}_{\mathrm{B}} - \mathbf{h}_{\mathrm{B}}(\mathbf{x}_{\mathrm{B}})]^{\mathrm{T}}\mathbf{R}_{\mathrm{B}}^{-1}[\mathbf{m}_{\mathrm{B}} - \mathbf{h}_{\mathrm{B}}(\mathbf{x}_{\mathrm{B}})] \end{cases} \quad subject\ to (\mathbf{x}_{\mathrm{B1}} - \mathbf{x}_{\mathrm{B2}}) = 0 \tag{4.9}$$

where $\mathbf{h}_{\mathrm{i}}(\mathbf{x}_{\mathrm{i}}), i = 1, 2$ is the nonlinear measurement function for each subsystem i while $\frac{\partial \mathbf{h}_1(\mathbf{x}_1)}{\partial \mathbf{x}_{\mathrm{B1}}} = \frac{\partial \mathbf{h}_2(\mathbf{x}_2)}{\partial \mathbf{x}_{\mathrm{B2}}} = 0$, and the complete vectors are as follow:

- $\mathbf{x}_1 = [\mathbf{x}_{1,\mathrm{int}}, \mathbf{x}_{\mathrm{B1}}]^{\mathrm{T}}$.

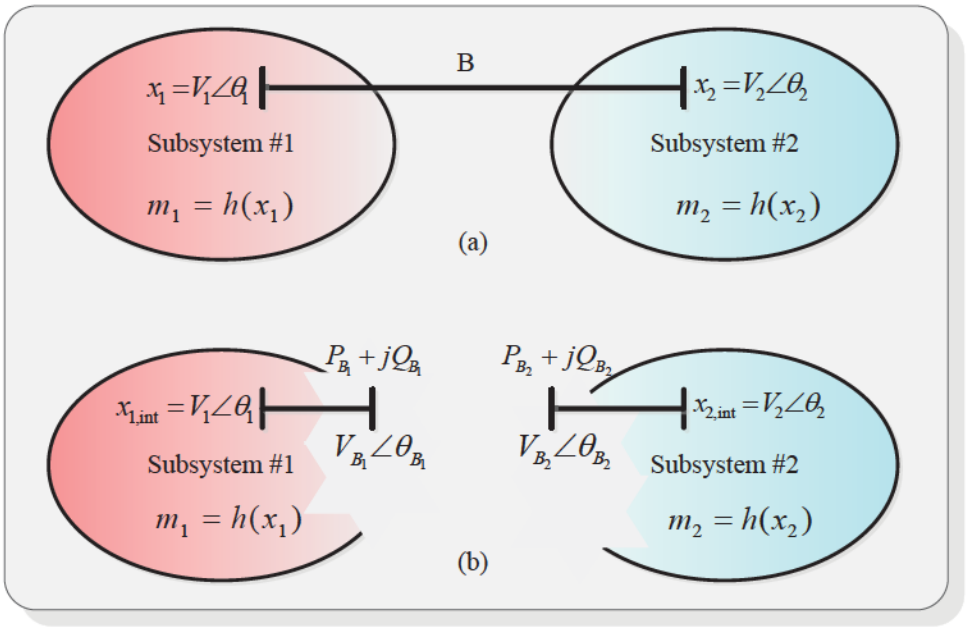


Figure 4.2: Domain decomposition: (a) interconnection of two subsystems, (b) split of two subsystem.

- $\mathbf{x}_2 = [\mathbf{x}_{2,\text{int}}, \mathbf{x}_{B2}]^{\text{T}}$.

4.4.1 Power System Domain Decomposition

The preliminary step for parallel implementation of relaxation-based DSE on GPU is to partition the system into interdependent subsystems while making the dependency between any two subsystems weak enough to ignore their interconnection. It is already shown that the rate of convergence in relaxation method is highly dependent on the method of partitioning [155]. There are different approaches to decompose a power system for parallel processing:

- Decomposition based on geographical distance; however, this approach stymied by the computational load balancing problem and may result in inaccuracy due to neglecting the effect of subsystems.
- Distributing equal numbers of generators and buses among subsystems; however, this is not an efficient method since the generator models vary in both size and complexity and network buses have different connectivity.
- Splitting the computation burden among processors based on the total number of equations; however, this method will increase the programming complexity.

The best way to decompose a system for parallel processing is to distribute equal amount of work among processors. The best result guaranteed when subsystem are independent of each other or in another word tightly coupled variables are gathered in same subsystem.

4.4.2 Coherency Analysis

From power system point of view determination of tightly coupled variables can find a physical meaning. Following a disturbance in the system, some generators lose their synchronism with the network which causes sudden changes in the buses connected to those generators and naturally partition the system into several areas. Generators in each of these areas are said to be coherent. In this situation, state estimation will take more iterations for some area since it should be repeated after clearing the disturbance. Coherent generators can be grouped in the same subsystem which can be solved independently from other subsystems. Partitioning the system into several areas in which generators are in step together or are coherent, will increase the accuracy of the state estimation and save time by localizing the effect of disturbances. The partitioning achieved using the coherency property is independent of the size of disturbance and the level of detail used in the generators which makes it suitable for our case studies [146].

In a network a pair of generators is called coherent if the difference between their rotor angles remains constant over time (4.10):

$$\delta_i(t) - \delta_j(t) = \Delta\delta_{ij} \pm \epsilon \tag{4.10}$$

where $\Delta\delta_{ij}$ is a constant value, and ϵ is a small positive number.

For efficient parallelization, in our work computational load balancing was also considered in domain decomposition. Equal load distribution among subsystems reduces the complexity resulting in faster computations which also makes it efficient for implementation in a multi-GPU architecture. Fig. 4.3 shows a power system decomposed into J subsystems. Decomposition based on the coherency approach and equal load work criteria, divides the full set of equations into several independent functions running in separate GPUs. The proposed method decomposes the system in two steps. First considering the coherency characteristic groups of coherent generators are identified. Then equal computational load among subsystems is considered which reduced the size of subsystems with significantly more load compared to other subsystems. It should be mentioned that upcoming GPUs are designed to handle load balancing automatically. However, the proposed method is designed to work with all types of GPU.

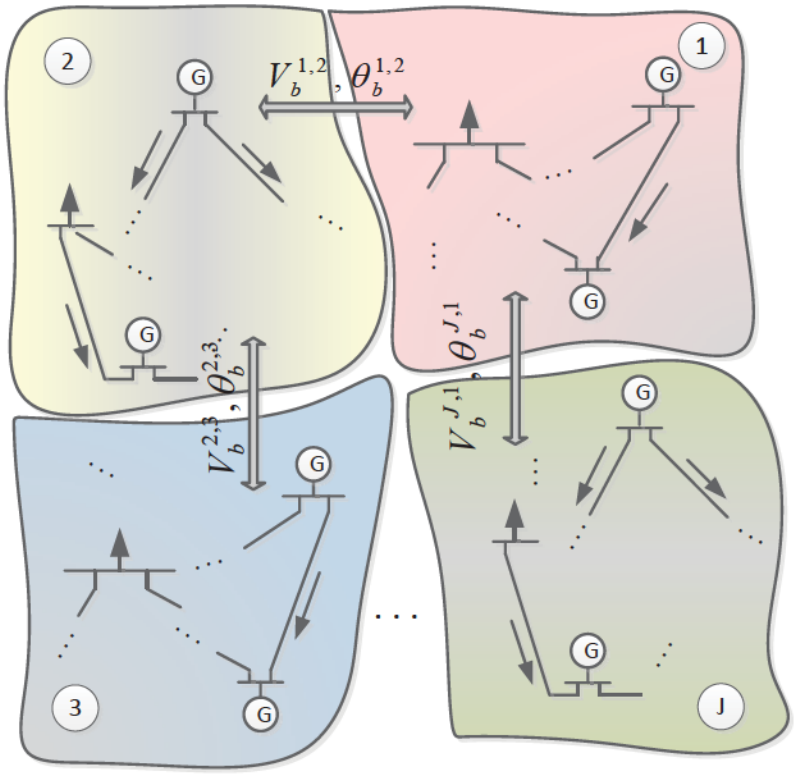


Figure 4.3: Original power system decomposed into J subsystems for RJDSE implementation.

4.5 Iterative Gauss-Jacobi Method

Two of the most famous iterative methods used for relaxation-based solutions are the Gauss- Seidel (G-S) and Gauss-Jacobi (G-J) methods. Consider a set of equations with n unknowns as follows:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n, \end{aligned} \tag{4.11}$$

In order to calculate unknowns values (4.11) can be rewritten as:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n), \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - \cdots - a_{2n}x_n), \\ &\vdots \\ x_n &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - \cdots - a_{nn-1}x_{n-1}), \end{aligned} \tag{4.12}$$

In general, for any unmown value in (4.11) following equation can be written:

$$x_i = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, i \neq j}^{n} a_{ij} x_j \right], \quad i = 1, 2, ..., n \tag{4.13}$$

The algorithm starts with an initial guess for the solution. At each iteration it updates x. The iterations stops when the absolute relative approximate error is less than a pre-specified tolerance for all unknowns. If in each iteration the algorithm uses the results of previous iteration it is called G-J (4.14) and if it uses the most updated value it is called G-S method. Since G-S uses the latest iteration it has a serial nature which makes it less suitable for our case study. So we only focus on G-J formulation. G-J iteration sequence for two subsystems are described in Fig. 4.4, respectively. Fig. 4.5 describe G-J algorithms.

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^k - \sum_{j=i+1}^{n} a_{ij} x_j^k \right] \tag{4.14}$$

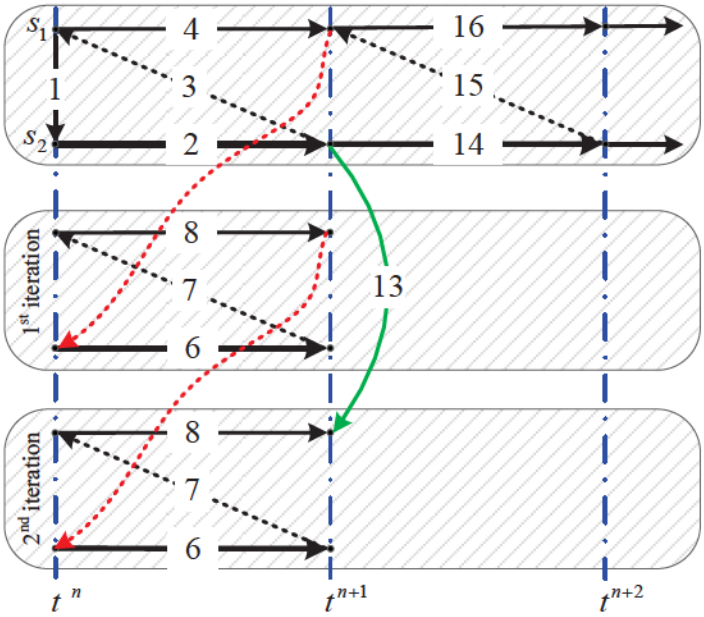


Figure 4.4: Gauss-Jacobi iterative method for two sub-systems.

4.5.1 Relaxation-based Gauss-Jacobi Method

In the relaxation based G-J (RG-J) method, the i^{th} subsystem uses the current iterate value from subsystems (1,..., *i*-1) and the previous iterate value from subsystems (*i*+1,...,*n*) as inputs. In another word, the solution of Subsystem *i* for the current time-step is fully independent from the solution of other subsystems. The subsystems are discretized and

Algorithm: Gauss Jacobi method

Choose an initial solution x_0
Set $k = 0$
While not converge ***do***
  for i=1 to n
    $\overline{x_i} = 0$
    for j=1 to i-1 and j=i+1 to n
      $\overline{x_i} = \overline{x_i} + a_{i,j} x_{(k-1)j}$
    end
    $\overline{x_i} = (b_i - \overline{x_i}) / a_{i,i}$
  end
    $x_i^k = \overline{x}_i$
  Check convergence
  Update $k = k+1$
end

Figure 4.5: Gauss-Jacobi Algorithm .

solved independently using space parallelism. This method exploits time parallelism over the simulation period since subsystems are solved concurrently.

Consider following differential equation which can be used for RG-J:

$$\dot{x}^L = f(x^L, u^L, x^G, u^G, t,), \tag{4.15}$$

where x^L and u^L are local variables which define the dynamic behavior of Subsystem *i*, and x^G and u^G are global variables that defining all subsystems excluding Subsystem *i*.

To solve each subsystem (4.15) all three steps explained earlier in Section 4.2 should be followed step by step. After the convergence of iterative solutions in all subsystems, the state and algebraic variables calculated from the last time-step are updated.

4.6 Parallel Relaxation-based WLS Static State Estimation

As first case study, proposed method was implemented on CPU architecture to evaluate the efficiency for further implementation in multi-GPU architecture. Consider a decomposition of the domain Ω into M non-overlapping sub-domains:

$$\Omega = \bigcup_{i=1}^{M} \Omega_i, \quad \Omega_i \cap \Omega_j = \{\emptyset\}_{i \neq j} \tag{4.16}$$

Let $\Delta(\mathbf{x})_i^{(0)}$ denote the initial condition in sub-domain Ω_i. A general RG-J algorithm for state estimation can be written as:

$$\begin{cases} \mathbf{G}(\mathbf{x})_i^{(k+1)}\Delta(\mathbf{x})_i^{(k+1)} = \mathbf{g}(\mathbf{x})_i^{(k+1)} & \text{in } \Omega_i \\ \mathbf{x}_i^{(k+1)} = \mathbf{x}_j^{(k)} & \text{on } \partial\Omega_i \cap \Omega_j \\ \mathbf{Z}_i^{(k+1)} = \mathbf{Z}_j^{(k+1)} & \text{on } \partial\Omega_i \cap \Omega_j \end{cases} \tag{4.17}$$

The flowchart of the relaxation-based Jacobi implementation of WLS static state estimation on CPU for a time interval of [0 T] is depicted in Fig. 4.6.

After decomposition, each sub-domain is stored and computed by a processor core. All processor cores solve the sub-domains in Parallel. The process starts from an initial condition for all the subsystems. To achieve convergence several iterations may be required, where each of the subsystems exchange boundary information and are then solved with updated data collected from other subsystems. If the stopping criteria is not satisfied, new iteration is performed. This process is repeated using a Jacobi method to iterate among subsystems until all variable converge with the necessary accuracy. In case of bad measurements, state estimation will repeat only for the sub-domain affected by bad measurement.

4.6.1 Experimental Results

To evaluate accuracy and efficiency of the proposed algorithms, the results were compared with traditional centralized state estimation. It is assumed that PMUs are installed at the boundary buses. To assess the accuracy of the state estimator, the results were also compared with the original power-flow results from PSS/E®. Using the partitioning pattern mentioned in [147], Case 1, IEEE 39-bus system has been divided into 3 subsystems: $\{1, 8, 9\}$, $\{2, 3, 4, 5, 6, 7\}$, and $\{10\}$. For computational load balancing another criteria was considered to have almost equal number of buses in each sub-domain. Satisfying both conditions simultaneously resulted in following 4 domains which are shown in Fig. 4.7.

Accuracy Analysis and Bad Data Detection

The role of bad data detection in state estimation is necessary since bad measurements easily affect the accuracy of the results. One of the popular method for BDD which is used in this paper is based on normalized residual test [150]:

$$r_N(l) = \frac{|\mathbf{r}(l)|}{\sigma_r(l)} \leq \lambda, \tag{4.18}$$

where $r_N(l)$ is the largest residual among all and $\sigma_r(l)$ is the standard deviation of the l^{th} component of the residual vector. More details about BDD methods is provided in Chapter 5. In this work the measurements having the largest normalized residual and larger than

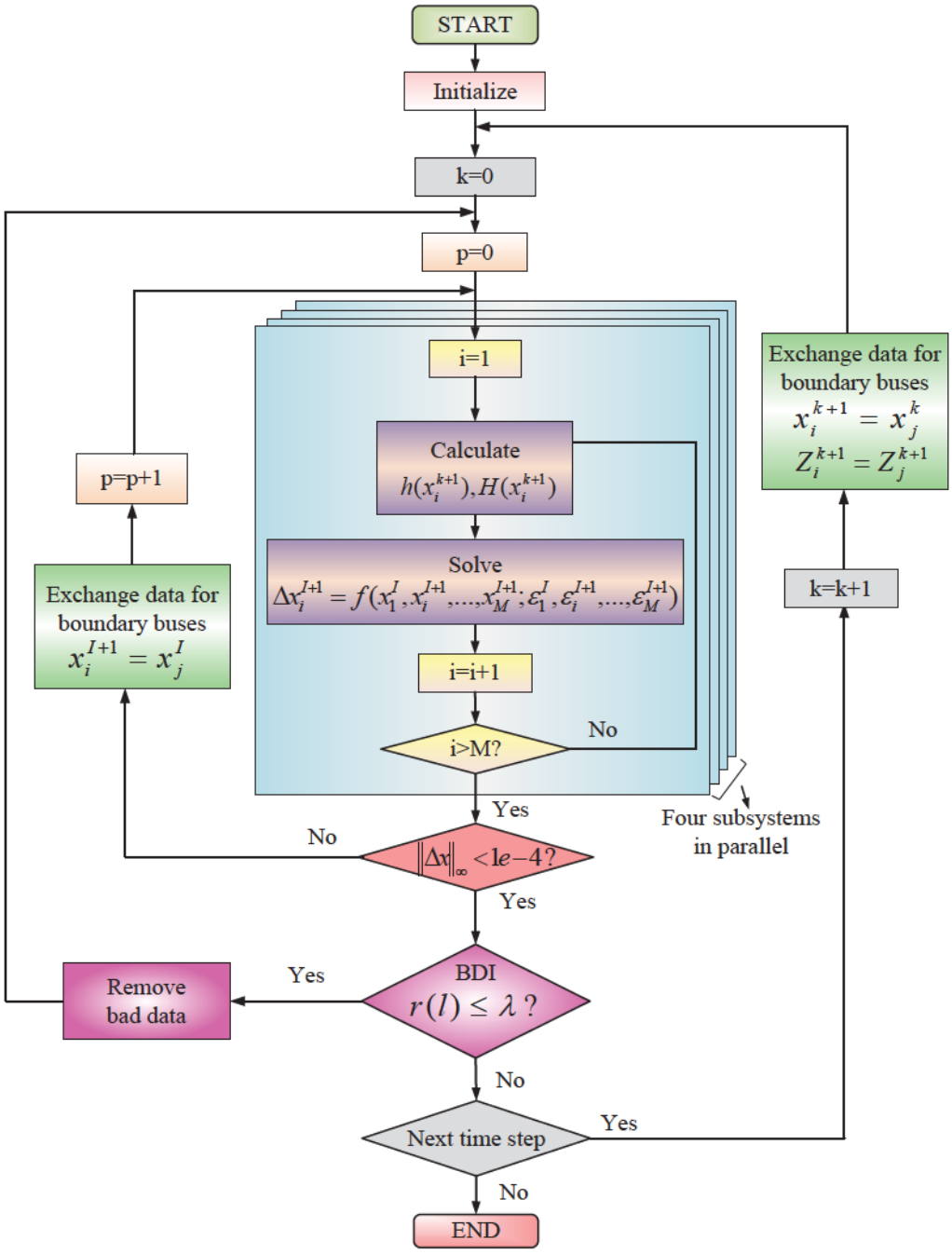


Figure 4.6: The relaxation-based Jacobi WLS algorithm with BDD; k: time step, i: the number of subsystems, p: iteration counter, l: index of component in residual vector.

3 were considered as bad data, with a 99.7% confidence level. After removing bad data, state estimation was repeated starting from the most recent estimate.

The results of parallel distributed state estimation are compared with traditional centralized state estimation method. The simulations were done using the data sets listed in

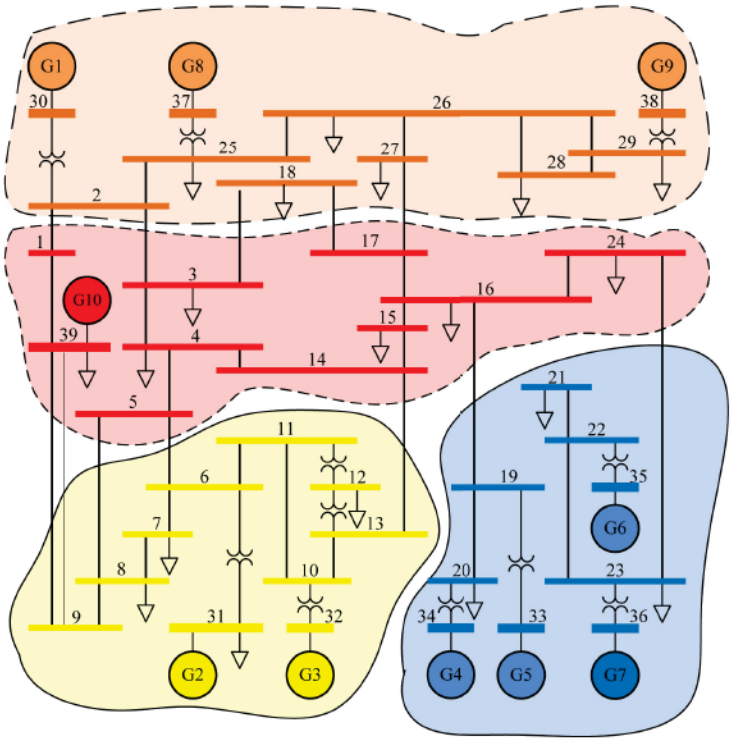


Figure 4.7: Decomposing a Case 1 into 4 subsystems to apply the additive Schwarz algorithm.

Table 4.1, with a tolerance of 0.0001 for convergence of the estimated parameters. Performance of the proposed method was evaluated for different case studies. The estimated states for Case 1 are shown in Fig. 4.8 and Fig. 4.9. It is clear from the results that the proposed approach can accurately estimate the voltage magnitude and phase angle.

Table 4.1: Summary of Results for Comparison of RG-J WLS with Centralized WLS

Case	No. of buses	No. of meas.	$E_V^{Cen.}$	$E_\phi^{Cen.}$	$E_V^{Dec.}$	$E_\phi^{Dec.}$	$T_{Ex}^{Cen.}$	$T_{Ex}^{Dec.}$	S_p
1	39	171	0.009	0.85	0.006	0.46	0.08s	0.11s	0.72
2	78	347	0.004	0.6	0.003	0.43	0.26s	0.33s	0.78
3	156	699	0.0032	0.55	0.001	0.47	0.51s	0.61s	0.83
4	312	1421	0.0041	0.6	0.0014	0.48	1.49s	0.91s	1.6
5	624	2865	0.0033	0.7	0.0012	0.49	5.2s	1.84s	2.8
6	1248	5825	0.004	0.65	0.0011	0.5	24.5s	8.1s	3.1
7	2496	11553	0.0044	0.6	0.0013	0.49	68.3s	15.5s	4.4
8	4992	23151	0.005	0.65	0.0017	0.49	364.5s	56.3s	6.5

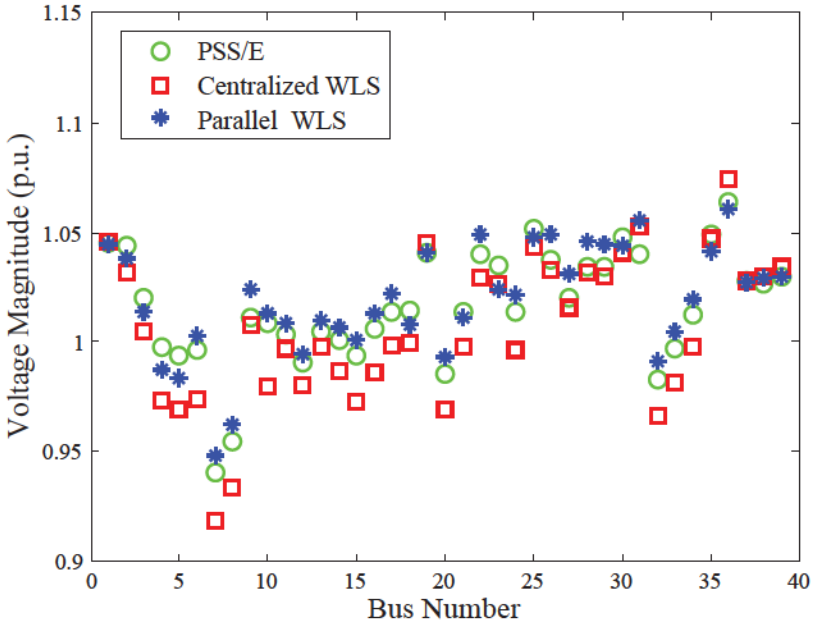


Figure 4.8: Voltage magnitudes for Case 1 with respect to system size.

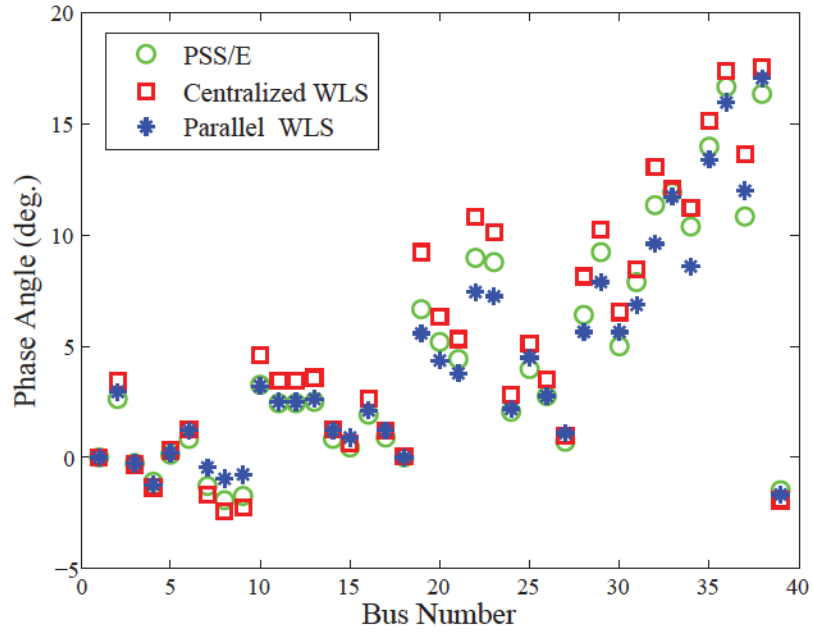


Figure 4.9: Phase angles for Case 1 with respect to system size.

The normalized Euclidian norm of the state estimation is also defined as a factor to evaluate the accuracy using:

$$\mathrm{E}_{\mathbf{x}} = \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\sqrt{\dim(\mathbf{x})}}, \tag{4.19}$$

where $E_{\mathbf{x}}$ is the normalized Euclidian norm of the estimation error, and $dim(\mathbf{x})$ is the dimension of vector $\mathbf{x}$. $\mathbf{x}$ and $\hat{\mathbf{x}}$ are vector of true states and estimated states, respectively. Table 4.1 shows the accuracy index for both voltage magnitude ($E_{\mathbf{V}}$) and phase angle (E_{ϕ}) for all case studies which clarifies the performance of the proposed method for large-scale systems.

Speed-up Analysis

To demonstrate the efficiency of the proposed approach execution time using decomposed SE ($T_{Ex}^{Dec.}$) is compared with traditional centralized ($T_{Ex}^{Cen.}$) SE method. As can be seen from Fig. 4.10 the percentage of required execution time for centralized WLS method increase very fast which shows the higher complexity of this method. In contrast, in proposed method growth rate of execution time is close to a linear behaviour.

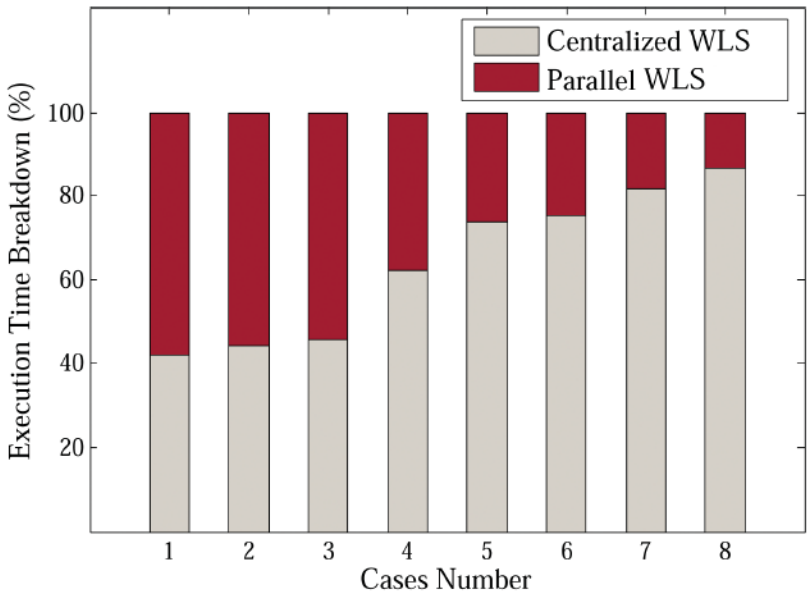


Figure 4.10: Percentage of execution time breakdown with respect to system size.

4.7 Relaxation-based Joint Dynamic State Estimation

In this section a relaxation-based joint dynamic state estimation (RJDSE) is implemented on 4 GPUs using relaxation-based Gauss-Jacobi method. All equations are expressed based on the unique SIMD-based architecture of GPU. Instead of using single element values vectors or matrices of them are used. In addition, all matrix-matrix and matrix-vector which includes many independent *for* loops are implemented in a fully parallel manner.

CUDA which is the general-purpose programming model for the GPU hardware was used in this work. The entire simulation code was written in C++ integrated with CUDA using CUBLAS and CUSPARSE libraries [156,157] using double precision floating point.

4.7.1 Hierarchy of Parallelism

After partitioning the system, the following set of equations can describe the dynamics of each subsystem:

$$\begin{cases} \mathbf{f}(\mathbf{x}_1^t, ..., \mathbf{x}_i^{t+\tau}, ..., \mathbf{x}_J^t, \dot{\mathbf{x}}_1^t, ..., \dot{\mathbf{x}}_i^{t+\tau}, ..., \dot{\mathbf{x}}_J^t, t, \\ \mathbf{u}_1^t, ..., \mathbf{u}_i^{t+\tau}, ..., \mathbf{u}_J^t) = \mathbf{0} \\ \mathbf{g}(\mathbf{x}_1^t, ..., \mathbf{x}_i^{t+\tau}, ..., \mathbf{x}_J^t, t, \mathbf{u}_1^t, ..., \mathbf{u}_i^{t+\tau}, ..., \mathbf{u}_J^t)) = \mathbf{0} \\ \mathbf{h}(\mathbf{x}_1^t, ..., \mathbf{x}_i^{t+\tau}, ..., \mathbf{x}_J^t, \mathbf{Z}_1^t, ..., \mathbf{Z}_i^{t+\tau}, ..., \mathbf{Z}_J^t, \varepsilon) = \mathbf{0} \end{cases} \tag{4.20}$$

where indices L and G stands for local and global variables, respectively. Eq. (4.20) is solved in parallel and iteratively for all subsystems. After each iteration, the global state variables are exchanged and updated between all interconnected subsystems. In order to calibrate the results, an overall loop based on the Gauss-Jacobi algorithm is applied outside the solutions of sub-systems. Since the Gauss-Jacobi algorithm only uses the previously computed values for the solution of each sub-system, all computations of sub-systems can be processed in parallel.

The algorithm starts at top level with Gauss-Jacobi iteration. By functional parallelism, almost equal tasks are assigned to each GPU. The iteration starts at the same time and in parallel inside all GPUs. Inside each iteration fine-grained parallelism is used to accelerate the process. After each iteration, only estimated states of boundary buses are exchanged. If the Gauss-Jacobi algorithm convergence is not satisfied, new iterations will be performed. Since the subsystems are fully independent of each other, after each time-step the results of network estimation will be transferred to the generator state estimation model. The same procedure is performed for generator dynamic state estimation. While network estimation is working on estimation for the time-step $t + \tau$, generator state estimation is working for the time-step t, simultaneously. Thus, the network estimation is always one step ahead of the generator estimation. The results will be checked for bad data identification after each time-step of network estimation. In case of bad data, only the subsystem affected by bad data will repeat state estimation instead of the whole system. Fig. 4.11 shows the complete flowchart of the RJDSE.

In summary, fine-grained parallelism is performed inside the functional parallelism, and functional parallelism is a subset of coarse-grained parallelism (Fig. 4.12). In the best case scenario, coarse-grained parallelism by dividing the system into J subsystems reduce the execution time to τ/J; using functional parallelism and J_1 independent tasks results in execution time of $\tau/J.J_1$; and finally utilizing J_0 independent matrix-matrix, matrix-vector and other type of fine-grained parallelism execution time can be further reduced to

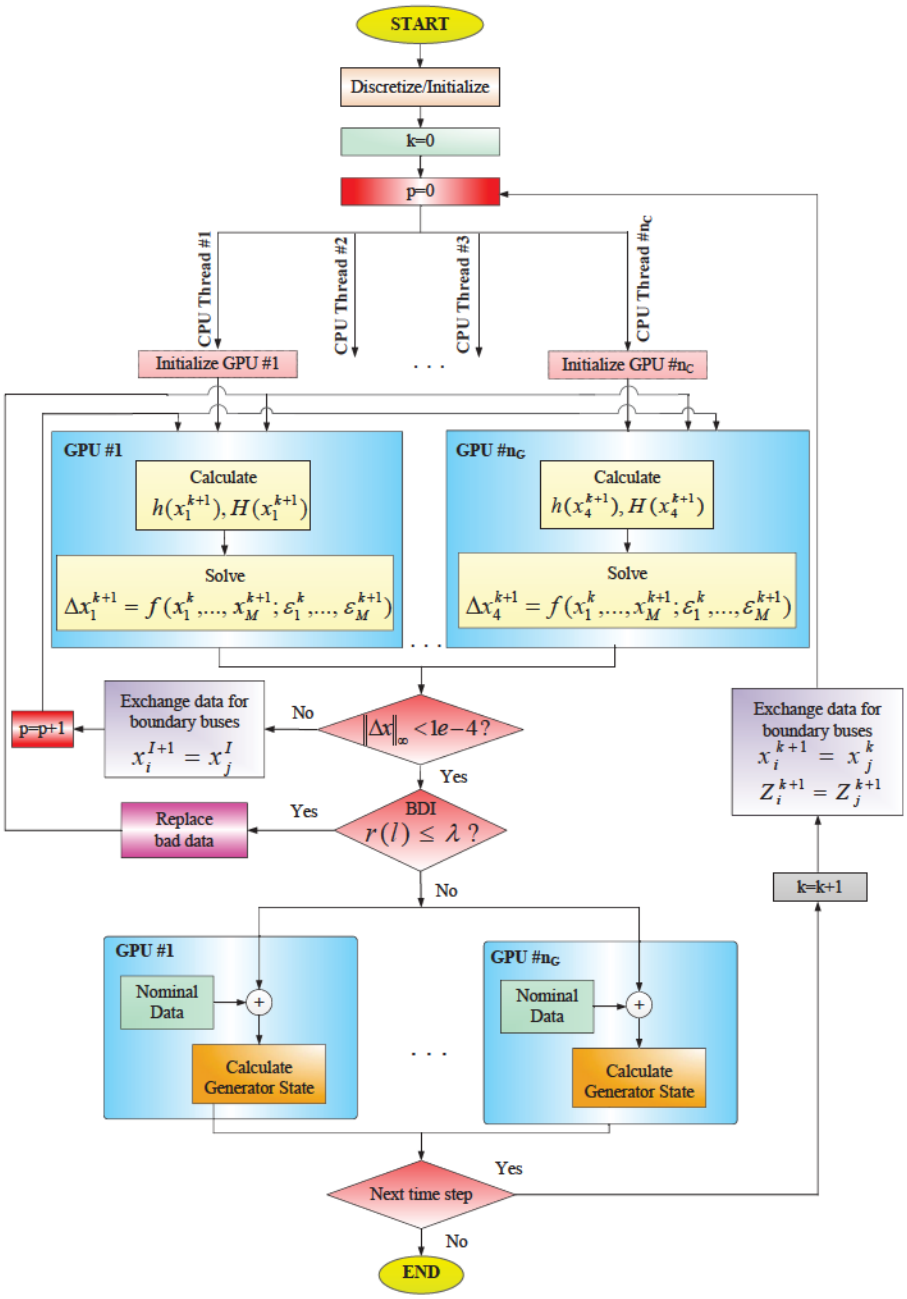


Figure 4.11: Flowchart of RJDSE implementation on multi-GPU architecture.

$\tau/J.J_1.J_0$. However, in reality the speedup is less than this considering the different costs of parallelization and data transfer to GPU.

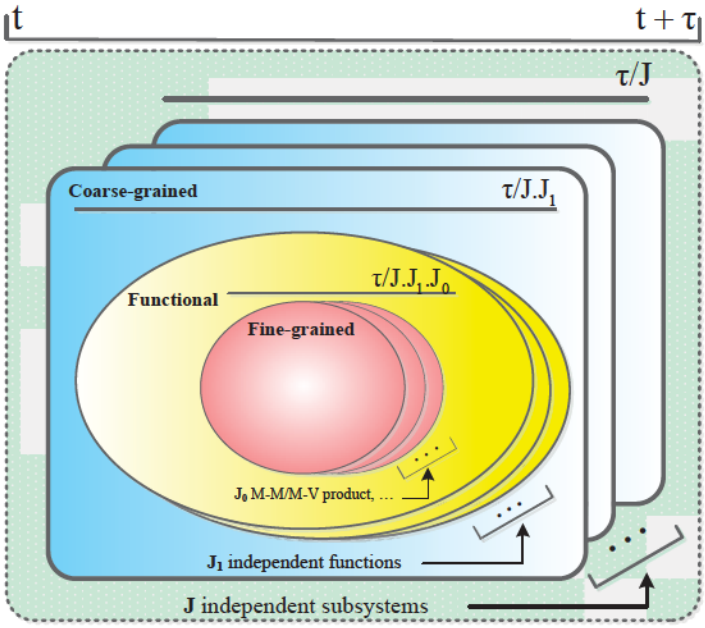


Figure 4.12: Hierarchy of parallelism in RJDSE, τ: integration time-step, t: simulation time.

4.7.2 Implementation of RJDSE on GPU

Case 1 which is the IEEE 39-bus system has been partitioned into four sub-domains satisfying both computational load balancing and coherency characteristic of the generators. Similarly, all the large test cases are partitioned. The simulation starts by initialization on the CPU. After that, the measurement set corresponding to each subsystem was transferred to GPU assigned for that specific subsystem. All the subsystems start the simulation at the same time. After each iteration boundary data was exchanged among subsystems. Once network state was converged, the results were used for generator state estimation. Network and generator state estimation was running simultaneously as shown in Fig. 4.13.

4.7.3 Experimental Results

The results of RJDSE implementation on multiple GPUs are demonstrated in this section. The accuracy of the simulation has been verified using the PSS/E® software. The initial condition for generator states was chosen based on the steady-state condition. Fig. 4.14 shows the overall block diagram of the proposed method.

Accuracy Analysis and BDD

Accuracy of the proposed method was evaluated under both normal and emergency conditions. A temporary three-phase fault is considered at t=3s which is cleared after 100 ms. The normalized Euclidian norm (4.19) of the state estimation is defined as a factor to eval-

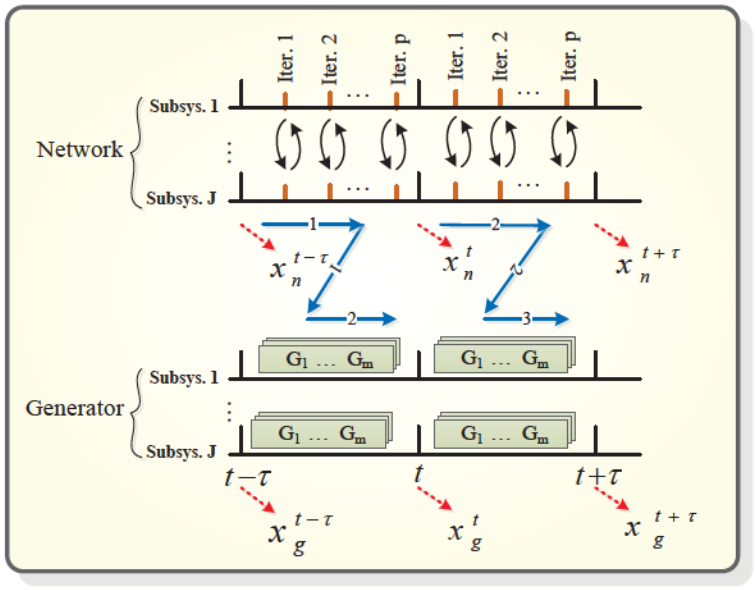


Figure 4.13: Time progression of RJDSE on GPU.

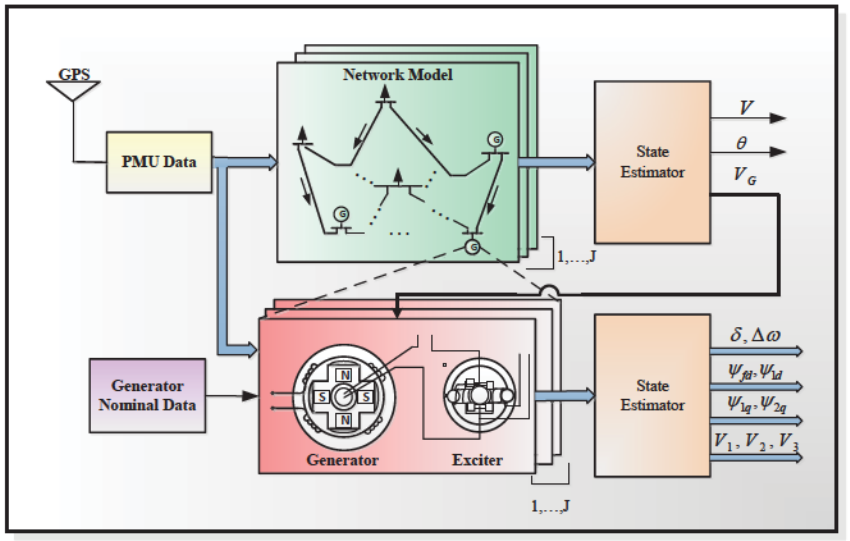


Figure 4.14: Overall block diagram of the proposed RJDSE method.

uate the accuracy. Results of simulation on all case studies are shown in Fig. 4.15.

The simulation results was followed by BDD. The largest normalized residuals with threshold of $\varkappa = 3$ were considered as bad data. Here the bad data refers to measurements with gross errors. Once bad data is identified corresponding measurement was updated by deducting gross error $(\frac{R_{ii}}{\sigma_{ii}} r_i)$ from bad data. Using the updated measurements state estimation was repeated only for the subsystems which were affected by bad data. For

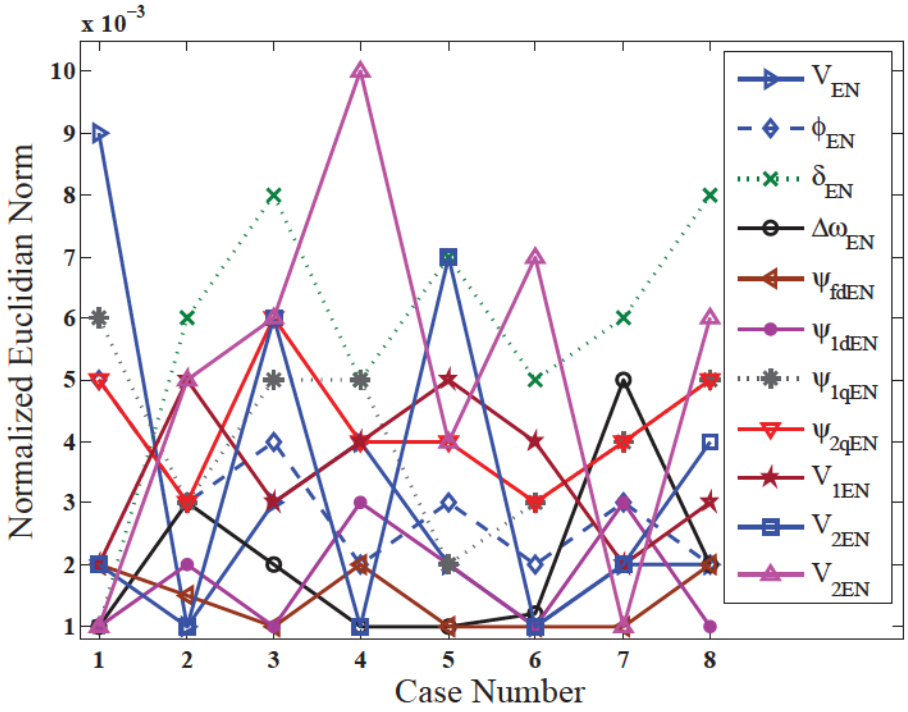


Figure 4.15: Normalized euclidian norm of the estimation error using RJDSE.

large scale systems, this localization of bad data can save lots of time which can in turn accelerate the state estimation process.

Also, the estimated states for Case 1 are shown in Fig. 4.16, Fig. 4.17, and Fig. 4.18. As shown the maximum of the average errors for all case studies are less than 0.001 p.u. All the simulation results proves that proposed method is able to accurately capture the dynamic behaviour of the system.

Complexity Analysis and Speedup

Generally, when a system with N buses is partitioned into M subsystems, each sub-domain has approximately N/M buses. Assume that solving a linear system with iterative method has complexity of $O(N^\alpha)$ where $\alpha \geq 1$. Using domain decomposition technique, the complexity of solving each subsystem is $O((N/M)^\alpha)$ which results in the complexity of $O((N)^\alpha/(M)^{\alpha-1})$ for the entire system. It may not be realistic to expect the same speedup in practice; however, the results clearly indicate the advantages of domain decomposition in accelerating DSE.

As can be seen from Table 4.2 and Fig. 4.19 the percentage of required execution times increases faster in single-GPU ($T^{Ex.}_{S.GPU}$) simulation compared with multi-GPU ($T^{Ex.}_{M.GPU}$)

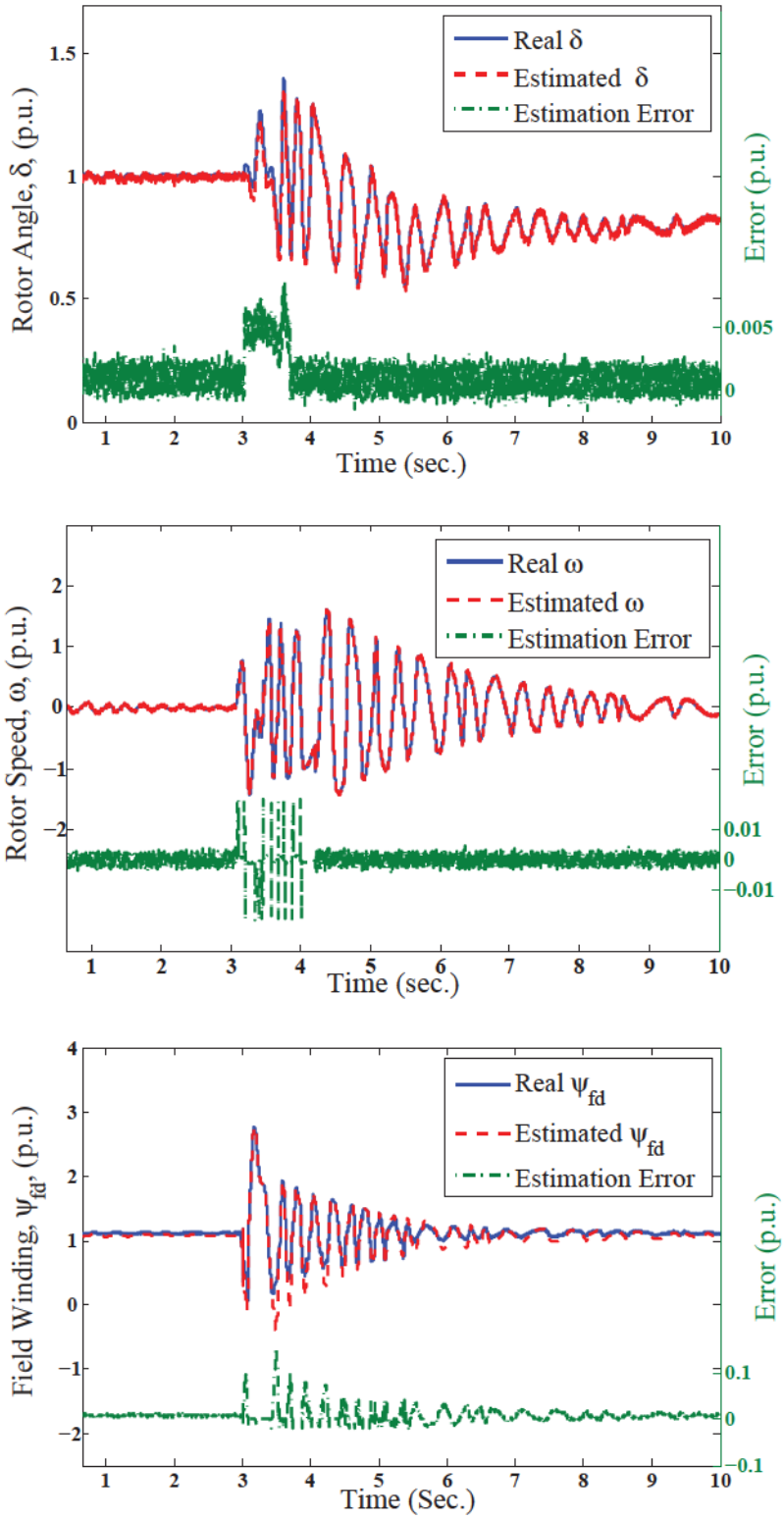


Figure 4.16: Generator state estimation $(\delta, \omega, \psi_{fd})$ and error of estimation in RJDSE.

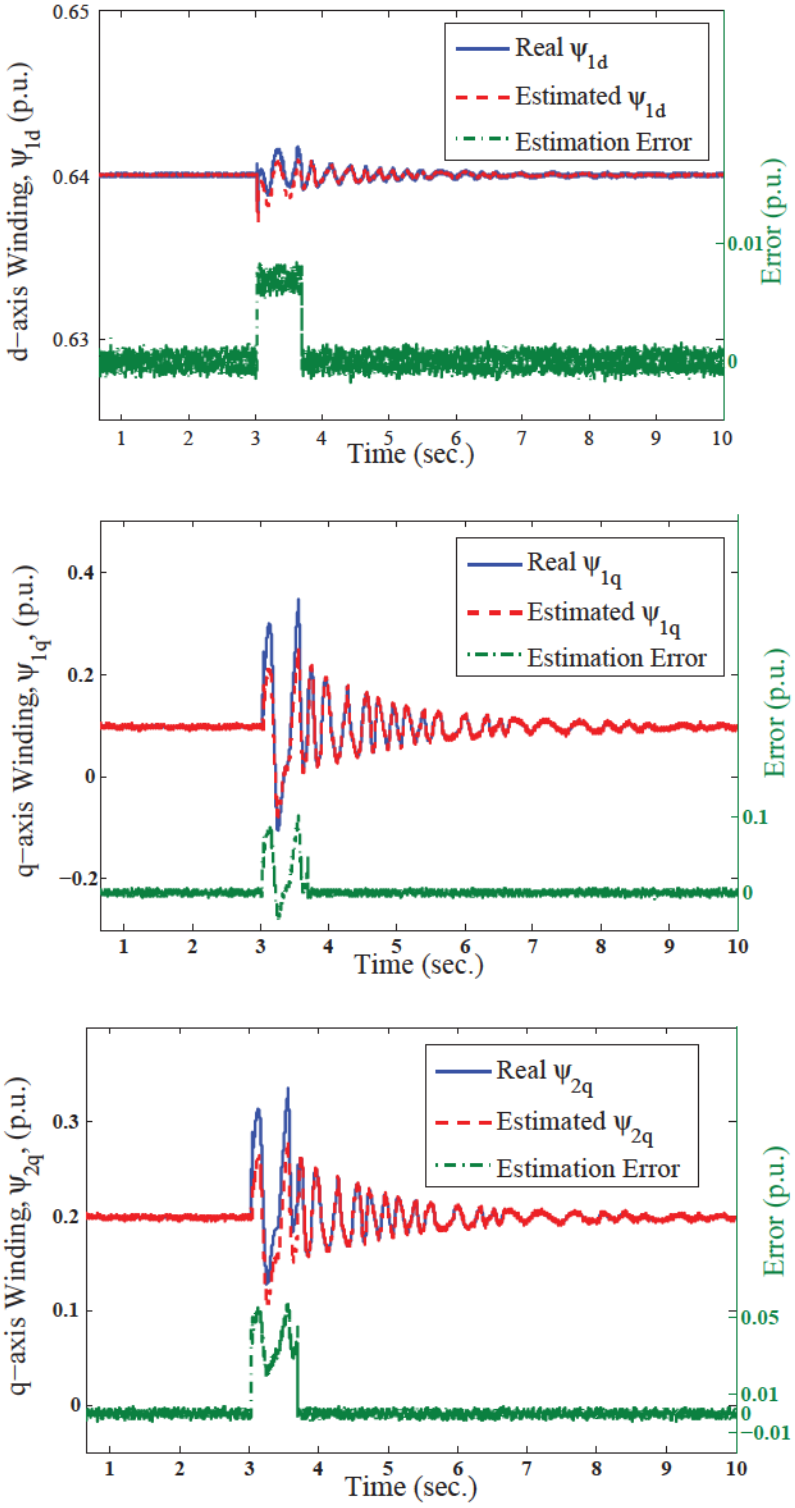


Figure 4.17: Generator state estimation ($\psi_{1d}, \psi_{1q}, \psi_{2q}$) and error of estimation in RJDSE.

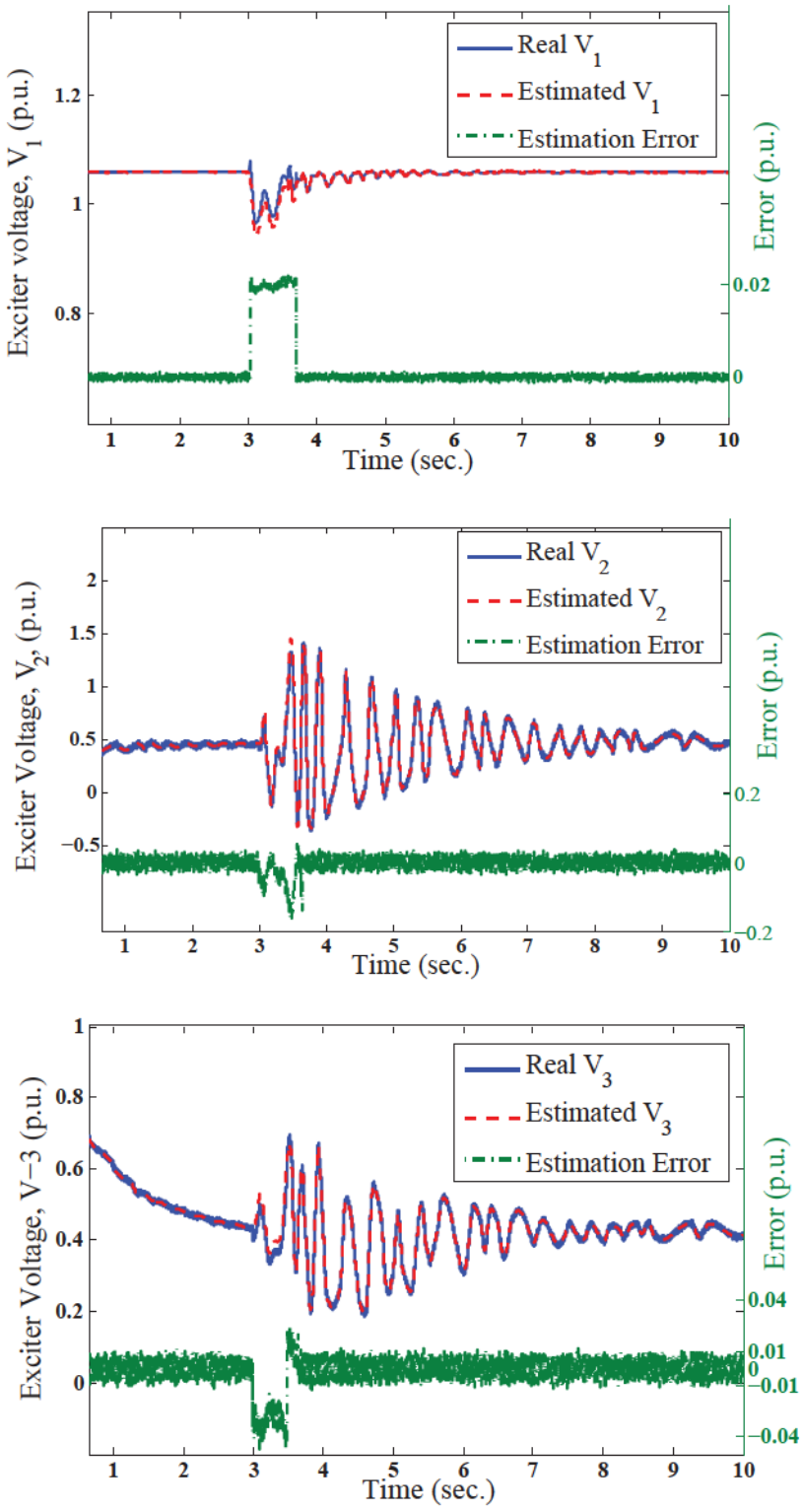


Figure 4.18: Generator state estimation (V_1, V_2, V_3) and error of estimation in RJDSE.

implementation which shows the higher complexity of this method. The execution time also verify that exploiting parallelism using GPUs can results in significant speed-up in the state estimation process.

Table 4.2: Execution Time in Single-GPU and Multi-GPU Implementation

Case	No. of buses	No. of gen.	$T^{Ex.}_{S.GPU}$	$T^{Ex.}_{M.GPU}$
1	39	10	0.4s	0.4s
2	78	20	1.1s	0.9s
3	156	40	2.16s	1.45s
4	312	80	6.9s	4.3s
5	624	160	12.6s	7.3s
6	1248	320	28.9s	15.3s
7	2496	640	43.1s	22.2s
8	4992	1280	62.8s	30.6s

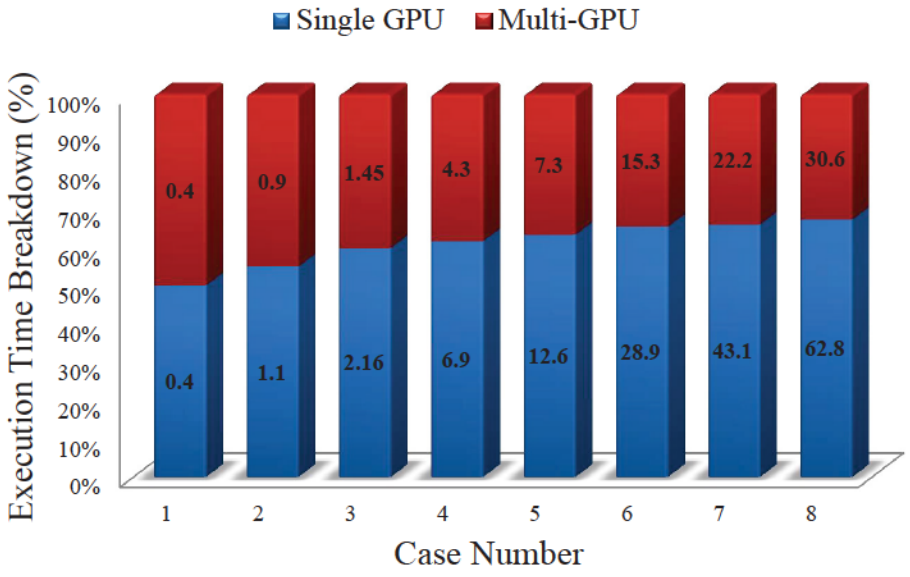


Figure 4.19: Percentage of execution time for varying test cases on single and multi-GPU simulators.

4.8 Discussion

Overall, simulation results prove the efficiency of the proposed method compare to other existing parallel SE approaches. There are small differences compared to PSS/E® (actual) results which are due to the fact that the order of block execution in each GPU grid is undefined in the kernel definition. Therefore, it leads to slightly different results if different CUDA blocks perform calculations on overlapping portions of data.

In terms of speed-up, as the results show for small cases, the parallel method is slower than the centralized method which is due to communication overhead in parallelism. Also comparison between single-GPU and multi-GPU shows the possibility of even more speed-up using more GPU. In addition, since the data set is divided into several subsets, in case of bad data SE was only repeated for the affected subset not for the whole system which saved lots of time and accelerated the DSE process.

In addition to the high parallelization offered by the relaxation method, its implementation on multiple GPUs covers the maximum size limitation imposed by CUDA/CUBLAS. Most of the CUBLAS function are limited to matrices with maximum size of 2040 row or column. In our case study the size of Jacobian matrix is $9l \times 9l$. Thus, the largest system that can be modeled is limited to 450 machines. In the relaxation method, however, the system is decomposed into subsystems which are solved individually, so each subsystem can use the maximum compute capacity of the available hardware.

4.9 Summary

Using traditional SE method, the size and cost of the simulator is usually prohibitive, especially for simulating large-scale systems. The main objective of this chapter was to investigate the effect of domain decomposition and relaxation-based techniques on accuracy and speed of SE by proposing a parallel relaxation-based joint DSE.

The proposed method is general and extensible to any number of GPUs connected in a cluster. Results show that more GPUs can reduce expected computation time. Result comparisons verified the accuracy and efficiency of the proposed method. In addition, the performance of the slow coherency method as the partitioning tool was analyzed, and it was concluded that for different fault locations in the system, results derived from this method had lower amounts of error.

In summary, the main contributions of the proposed approach are as follows:

- Parallel multi-GPU implementation of joint generator and network DSE.

- Application of relaxation method which distributes equal workload among all the processors and eliminates the need for all subsystem to be observable.
- Domain decomposition based on coherency method which reduce the effect of non-overlapping decomposition on the accuracy.
- Eliminating central coordinator which reduces the communication time between the subsystems.
- Distributed localized bad data analysis in parallel with state estimation.

5

Robust Dynamic State Estimation Against Cyber-attack

5.1 Introduction

Although the advancement of cyber technologies in sensing, communication and smart measurement devices significantly enhanced power system security and reliability, its dependency on data communications makes it vulnerable to cyber-attacks. Coordinated false data injection (FDI) attacks manipulate power system measurements in a way that emulate the real behaviour of the system and remain unobservable, which misleads the state estimation process, and may result in power outages and even system blackouts [80]. The increasing demand for reliable and economical electricity services raises critical challenges in online monitoring and control of future power grids which rely on DSE; therefore, security of DSE and its vulnerability to cyber-attack is a major concern.

To overcome the effect of cyber-attacks, in this chapter considering the stochastic nature of the system disturbances a cyber-physical model of the power system utilizing the Markov chain is proposed. A set of possible states along with the probability of each state is generated. A Markov chain based on these states is then defined. After each estimation process all states are checked on the Markov chain. If the estimated states are close to a value with low probability or out of the Markov chain, the possibility of the cyber-attack is deemed high. Furthermore, to increase the security of the system, critical measurements are identified and protected. The proposed attack detection method was built upon a parallel Kalman filtering algorithm for DSE. In order to speed up the whole process, the proposed robust DSE was implemented on GPU.

5.2 Bad Data Detection

Detecting and identifying bad data in state estimation is usually done by comparing the telemetered measurements from SCADA with the estimated values of the states. Traditionally, bad data were assumed to be caused by random errors resulting from a fault in a meter and/or its attendant communication system [74, 161]. Some bad data such as negative voltage magnitudes or measurements with several orders of magnitude larger or smaller than expected values can be easily detected and eliminated prior to state estimation. However, it is not the case for all types of bad data. Therefore, state estimators have to be followed by BDD to ensure the accuracy of the estimation. The traditional BDD analysis is usually based on the properties of measurements residuals.

There are different types of measurements in power system which may show different properties and affect the outcome of the state estimation accordingly [5]. These measurements can be classified in two main categories as follows:

- *Critical measurement*-those measurements whose elimination from the measurement set will result in an unobservable system are called critical measurements. The measurement residual of a critical measurement is always zero.
- *Redundant measurement*- a measurement which is not critical is a redundant measurement. Only redundant measurements may have nonzero measurement residuals.

Two of the most important methods used for detecting bad data are the Chi-squares and largest normalized residual (LNR) tests.

5.2.1 Chi-squares Test

Consider a set of J independent variable (S_i) where:

$$S_i \sim N(0,1), \quad i = 1, ..., J, \tag{5.1}$$

A new variable S' defined as follows:

$$S' = \sum_{i=1}^{J} S_i^2, \tag{5.2}$$

will have a χ^2 distribution with J degree of freedom. The degree of freedom shows the number of independent variables. In a power system, degree of freedom is equal to the difference between the total number of measurements and the system states. Choosing a probability of error, e.g. 0.03, the threshold $\hat{x}_t$ can be set such that:

$$P_r\{\hat{x} \geq \hat{x}_t\} = 0.03, \tag{5.3}$$

where P_r is the probability function. The $\hat{x}_t$ represents the largest acceptable value for $\hat{x}$ that will not trigger BDD. If $\hat{x}$ exceeds this threshold, then it will not have a χ^2 distribution which implies it is a bad data with 97% probability.

Consider the objective function in WLS state estimation written in terms of measurements error:

$$\mathbf{J}(\hat{x}) = \sum_{i=1}^{2l+2n+1} (m_i - h_i(\hat{x}))^2 R_{ii}^{-1}, \tag{5.4}$$

Define the threshold for d degree of freedom with p probability ($\chi^2_{d,p}$):

$$p = P_r\{\mathbf{J}(\hat{x}) \leq \chi^2_{d,p}\}, \tag{5.5}$$

If $\mathbf{J}(\hat{x}) \geq \chi^2_{d,p}$ then bad data exist, otherwise measurements are free of bad data. The χ^2 distribution values for different degrees of freedom can be determined using Matlab® or in various statistical publications [5].

5.2.2 Largest Normalized Residual Test

The normalized residual vector r^N can be defined as follow:

$$r_i^N = \frac{|\mathbf{r}_i|}{\sqrt{\sigma_{ii}}}, \tag{5.6}$$

where r_i^N is the i^{th} normalized residual, σ_{ii} is the diagonal component of the residual covariance matrix which can be calculated as:

$$\mathrm{Cov}(\mathbf{r}) = \boldsymbol{\sigma} = \mathbf{R} - \mathbf{H}\mathbf{G}^{-1}\mathbf{H}^T, \tag{5.7}$$

The largest value of normalized residual (LNR) vector r_L^N is compared against a desired threshold to decide on the existence of bad data.

Considering the residual vector in WLS state estimation, the LNR test can be written as:

$$r_i^N = \frac{|\mathbf{m}_i - \mathbf{h}_i(\hat{x})|}{\sqrt{\sigma_{ii}}} > r_t, \tag{5.8}$$

where r_t s the desired threshold. If $r_L^N > r_t$ then L^{th} measurement is bad data. The r_L^N is the largest residual among all [5].

5.2.3 Bad Data Removal

If a measurement is suspected as a bad data it should be removed from measurement set before the next cycle of state estimation. The measurement that is corrupted with bad data can be written as:

$$\mathbf{m}_i^{bad} = \mathbf{m}_i + \mathbf{e}_i, \tag{5.9}$$

where m_i^{bad} is the bad measurement and e_i is the gross error. An approximate value for e_i is computable, however is out of scope of our work [5]. Subtracting this error from bad measurement results in:

$$\mathbf{m}_i \simeq \mathbf{m}_i^{bad} - \frac{R_{ii}}{\sigma_{ii}}\mathbf{r}_i^{bad}, \tag{5.10}$$

where $\mathbf{r}_i^{bad}$ is the residual of bad measurements. State estimation can be repeated after correcting the bad measurement.

It should be considered that measurement redundancy is a key issue in the performance of BDD. However, existing measurement configurations may not always yield such desired level of redundancy which makes the BDD impractical for such cases. For example bad data associated with critical measurements can not be detected or identified. Many researchers have considered the problem of BDD in power systems [75, 77, 78, 162], however conventional BDD approaches usually fail when the network malfunction is deliberately caused by an attacker who manipulates the communication between RTUs and the SCADA system [4, 79].

5.3 False Data Injection Attack

In false data injection (FDI) attacks, the adversary who has the knowledge of the network configuration changes some of meter readings from SCADA and manipulates the state variables arbitrarily. This type of malicious attacks can effectively bypass the existing BDD technique. The main reason for this failure is that all traditional BDD techniques assume that bad measurements result in significant measurements residual, However there is no trace of measurements residual in cyber-attack.

The general rule for a hidden attack is that the attacker must alter the data so that the measurements can plausibly correspond to the physical properties of the system. The main idea of FDI attack is to add a nonzero attack vector a to the original measurements vector m which results in a false estimation as follow:

$$\hat{\mathbf{x}_a} = \hat{\mathbf{x}} + \mathbf{c} \tag{5.11}$$

where $\mathbf{c}$ and $\hat{\mathbf{x}}_a$ are the error added to the original estimation, and corrupted state, respectively. The i^{th} nonzero element in a means that the attacker compromises the i^{th} measurement by replacing its original value with $\mathbf{m}_i + \mathbf{a}_i$.

Considering the measurement residual a necessary condition to hide an attack can be derived as follows:

$$\begin{aligned} \mathbf{r}_a &= \|\mathbf{m}_a - \mathbf{H}\hat{\mathbf{x}}_a\| = \|\mathbf{m} + \mathbf{a} - \mathbf{H}(\hat{\mathbf{x}} + \mathbf{c})\| \\ &= \|\mathbf{m} - \mathbf{H}\hat{\mathbf{x}} + (\mathbf{a} - \mathbf{H}\mathbf{c})\| = \|\mathbf{m} - \mathbf{H}\hat{\mathbf{x}}\|. \end{aligned} \tag{5.12}$$

The above equality constraint results in $\mathbf{a} = \mathbf{H}\mathbf{c}$. A structured sparse attack like $\mathbf{a} = \mathbf{H}\mathbf{c}$ will result in the same residual and will not be detected by BDD. In this case, the system operator would mistake $\hat{\mathbf{x}} + \mathbf{c}$ for a valid estimate.

Definition: The sparse attack vector $\mathbf{a} = [a_1, ..., a_m]^T$ is called false data injection attack if and only if it satisfies the relation $\mathbf{a} = \mathbf{H}\mathbf{c}$, where $\mathbf{c} = [c_1, ..., c_n]^T$ is a arbitrary nonzero vector [80].

In general, the minimum number of measurements the attacker needs to manipulate for a hidden attack on estimated value at bus *i* depends on the following:

- The number of adjacent buses to bus *i*,
- The number of measurements at bus *i*,
- The lines connecting bus *i* to its adjacent buses.

Fig. 5.1 shows a possible cyber-attack on an energy control center. For example, assume that the attacker wants to alter the active power flow on the line connecting bus *i* and *j*. Based on the following equation the attacker has to at least change one of the four variables, voltage magnitude: v_i, v_j, and phase angle: θ_i, θ_j.

$$P_{ij} = v_i^2.g_{ij} - v_i v_j(g_{ij}\cos(\theta_i - \theta_j) - b_{ij}\sin(\theta_i - \theta_j)). \tag{5.13}$$

Imagine that the attacker adjusted the estimated value for v_j to v_j^a, the following equation must be solved in order to find the voltage magnitude which will yield the desired power flow:

$$P_{ij} = v_i^2.g_{ij} - v_i v_j^a(g_{ij}\cos(\theta_i - \theta_j) - b_{ij}\sin(\theta_i - \theta_j)). \tag{5.14}$$

where g_{ij} and b_{ij} represents line admittance parameters. Since power flow and power injection are functions of voltage magnitudes and phase angles, the value of other measurements can be calculated considering the relationship between power flow and power injection. Also, the attacker must change all the measurements which are functions of v_j. In another words, the following should be satisfied in order to keep the attack hidden:

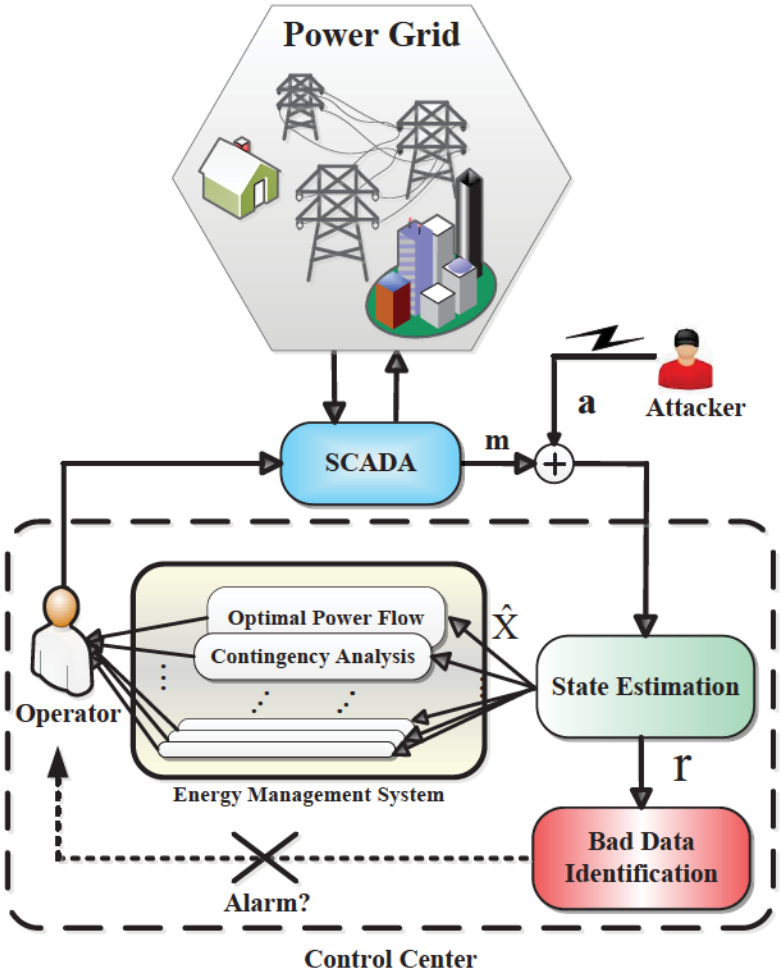


Figure 5.1: Dynamic state estimation under cyber-attack, a: attack vector, m: measurement, r: measurement residual, x̂: estimated state .

$$\begin{aligned} \sum_{i} \Delta P_i + \Delta L_P = 0, \\ \sum_{i} \Delta Q_i + \Delta L_Q = 0 \end{aligned} \tag{5.15}$$

where ΔP and ΔQ represent the alterations in active power flow/power injection and reactive power flow/power injection, respectively. ΔL represents the power losses.

5.3.1 Minimum Cost FDI Attacks

In order to find a minimal attack on measurement *i* the attacker has to optimise vector a to have the least number of non-zero elements by solving following problem [163]:

$$\begin{aligned} \alpha_i &= \min_{\mathbf{a}} \|\mathbf{a}\|_0 = \min_{\mathbf{c}} \|\mathbf{Hc}\|_0, \\ &\text{such that } \; 1 = \sum_j H_{ij} c_j, \end{aligned} \tag{5.16}$$

where $\|\mathbf{a}\|_0$ denotes the number of non-zero elements in a, and H is the Jacobian matrix in WLS state estimation problem. The above problem is hard to solve, so the easiest way is to find upper and lower bound on α_i.

Since at least one measurements need to be corrupted, the lower bound is $\alpha_i^u = 1$. By considering the i^{th} row of Jacobian matrix and the columns for which this row has nonzero elements, an upper bound can be found to construct a false-data attack vector. The upper bound implies the minimum number of measurements required for a successful FDI attack.

Assume that H_{ij} is non zero. The following attack vector archives the attackers goal:

$$\begin{aligned} a_i^j &= \frac{a_i}{H_{ij}} H_{.,j}, \\ \alpha_i^u &= \min_{j:H_{ij}\neq 0} \|H_{.,j}\|_0, \end{aligned} \tag{5.17}$$

where $H_{.,j}$ is the *j*-the column of H and α_i^u is the upper bound which is associated with the sparest vector among all a_i^j.

For further clarification consider a simple small 3-bus power network shown in Fig. 5.2.

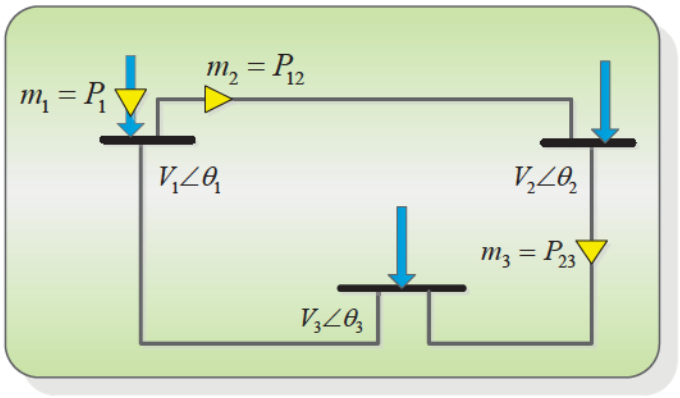


Figure 5.2: 3-bus power network with three measurements P_1, P_{12}, and P_{23}.

For simplicity assume that the resistance in the transmission lines is small compared to its reactance and all voltages are set to 1 p.u. $\theta_1 = 0$ as the reference angle and the state vector to be estimated is $[\theta_2, \theta_3]^T$. Considering $[P_1, P_{12}, P_{23}]$ as measurements, jacobian matrix for network in Fig. 5.2 can be calculated as:

$$H = \begin{bmatrix} -1 & -1 \\ -1 & 0 \\ 1 & -1 \end{bmatrix} \tag{5.18}$$

Using (5.16) in order to calculate α_1 following should be satisfied:

$$-c_1 - c_2 = 1 \Longrightarrow \begin{cases} \text{If} & c_1 = -1, c_2 = 0 \quad Hc = [1, 1, -1]^T \\ \text{If} & c_1 = 0, c_2 = -1 \quad Hc = [1, 0, 1]^T \end{cases} \tag{5.19}$$

It is obvious that the second choice results in sparsest a_1 and the upper bound is $\alpha_1^u = 2$. The same way a_2 and a_3 can be calculated.

5.4 Markov-Chain Formulation

Consider a physical system that has k possible states and at any one time, the system is in one of its *k* states. The system called Markov Chain if given an observation set at time *t*, the probability of the system being in a specific state depends only on its status at time *t*-1 [164]. Considering a set of states x_i which taking values s_i, a first order Markov chain fulfill following properties:

$$Pr(x_t = s_{i_t} | x_{t-1} = s_{i_{t-1}}, ..., x_0 = s_{i_0}) = Pr(x_t = s_{i_t} | x_{t-1} = s_{i_{t-1}}) \tag{5.20}$$

The power system can be modeled as a stochastic hybrid dynamical system where the stochastic availability of generation and state is explicitly included. Because many cyber-attacks require a series of related events to accomplish, an l^{th} order Markov-chain is suitable for our case study to improve attack detection performance by incorporating the continuous events. A stochastic process which fulfils the following properties is called an l^{th} order Markov-chain.

$$Pr(x_t = s_{i_t} | x_{t-1} = s_{i_{t-1}}, ..., x_0 = s_{i_0}) = Pr(x_{t+1} = s_{i_0} | x_t = s_{i_1}, ..., x_{t-l} = s_{i_l}) \tag{5.21}$$

where Pr refers to probability function and t represent the time. In this process the probability of getting into the next state depends upon the *l* previous states.

To define a Markov model, the following probabilities have to be specified:

- The transition probability matrix $\mathbf{TP} = [tp_{ij}]_{k \times k}$,
- Initial probabilities $\pi_i = Pr(x_0 = s_i)$

where

$$\mathrm{tp_{ij}} = \Pr(\mathrm{x_{t+1}} = \mathrm{s_j} | \mathrm{x_t} = \mathrm{s_i}), \quad i, j \in 1, 2, ..., k \tag{5.22}$$

In this chapter $\mathrm{tp_{ij}}$ is modeled by a function of Euclidean distance between historical measurement and current observation. Since 10 set of measurements are selected as historical data, we use a 10^{th} order Markove chain. The proposed parallel DSE using EKF calculates the state of the system. The Euclidian distance of the historical data and estimation of the trusted buses are calculated. Based on the results a probability is assigned to each data set. When the Markov model is ready, the projected estimates and the Markov model are compared by the detector. If the difference between the two is above a pre-computed threshold, an alarm is triggered to notify a possible attack or failure.

The Euclidean method compares the difference between the two sets of data $(\mathrm{x_1}, \mathrm{x_2})$ based on the distance metric as shown in (5.23).

$$\mathrm{ED}(\mathbf{x_1}, \mathbf{x_2}) = \sqrt{(\mathbf{x}_{1,1} - \mathbf{x}_{2,1})^2 + ... + (\mathbf{x}_{1,\mathrm{n}} - \mathbf{x}_{2,\mathrm{n}})^2} \tag{5.23}$$

If the difference is larger than a pre-computed threshold, the detector triggers an alarm. However, to avoid false alarms due to measurement or system errors, the threshold was set to filter 99% of noise. Fig. 5.3 shows the overall block diagram of the proposed method.

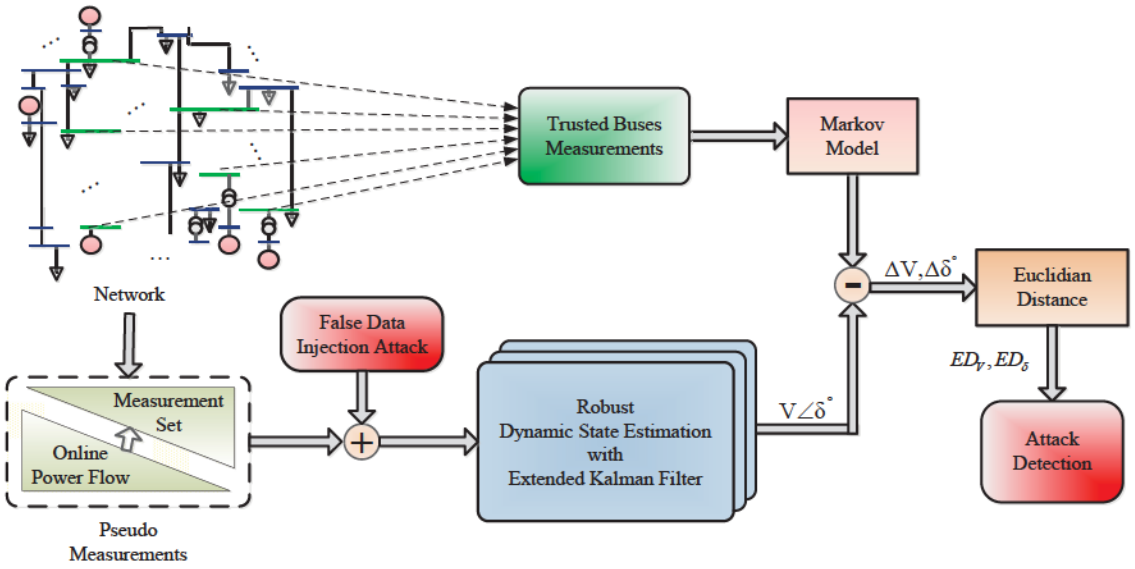


Figure 5.3: Overall block diagram of the proposed robust DSE method.

5.5 Critical Measurement Protection

Changing the critical measurements using updated information decreased the chance of successful cyber-attack. Large-scale power grids contain thousands of meters which makes the protection of measurements highly expensive. In order to reduce the cost, in our work the critical meters are identified and protected based on optimal PMU placement. Critical measurements are the ones whose elimination make the entire system unobservable. One of the important properties of critical measurements is that its measurement residual will always be zero. The measurement residual r can be written as follows:

$$\mathbf{r} = \Delta\mathbf{m} - \Delta\hat{\mathbf{m}} = \Delta\mathbf{m} - (\mathbf{h}(\hat{\mathbf{x}}) + \varepsilon) \backsimeq \Delta\mathbf{m} - (\mathbf{H}\Delta\hat{\mathbf{x}} + \varepsilon). \tag{5.24}$$

Substituting gain matrix from (3.15) in (5.24) we get,

$$\mathbf{r} = \Delta\mathbf{m} - \mathbf{H}\mathbf{G}^{-1}\mathbf{H}^{T}\mathbf{R}^{-1}\Delta\mathbf{m} = (\mathbf{I} - \mathbf{S})\Delta\mathbf{m} \tag{5.25}$$

For residual equal to zero, the diagonal element S_{ii} of matrix **S** should be one, which implies that the i^{th} measurement is critical.

5.5.1 Optimal PMU Placement

By optimal PMU placement at strategic buses in the system, we try to increase the accuracy of the system, protect most of the critical measurements, and provide a subset of trusted buses to use them in the attack detection algorithm. The objective of PMU placement problem is to accomplish this task using minimum number of PMUs. This problem can be formulated and solved as shown below [165]:

$$\begin{aligned} \text{min.} \sum_{i=1}^{n} \mathbf{\Lambda}_i \times \mathbf{\Gamma}_i, \\ \text{subject to :} \sum_{j=1}^{n} \mu_{i,j} \times \Gamma_i \geq 1 \text{ at bus i} \end{aligned} \tag{5.26}$$

$$\Gamma_i = \begin{cases} 1 & \text{If PMU is installed at bus i} \\ 0 & \text{Otherwise} \end{cases} \tag{5.27}$$

where $\mu_{i,j}$ is the element of connectivity matrix which is 1 if bus i and bus j are connected , and 0 otherwise. Λ_i is the cost of PMU installation at bus i. The constraint on (5.26) ensures that all subsystems created upon removal of each critical measurements will have at least one PMU measurement. It actually transform all critical measurements into redundant measurements.

Consider a 5-bus system shown in Fig.5.4. The connectivity matrix can be written as:

$$\mu = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{5.28}$$

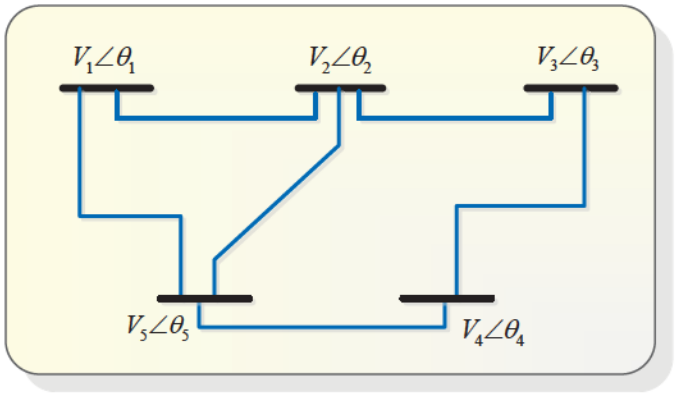


Figure 5.4: 5-bus power network.

The constraint for this case can be formed as:

$$\begin{cases} \Gamma_1 + \Gamma_2 + \Gamma_5 \geq 1 \\ \Gamma_1 + \Gamma_2 + \Gamma_3 + \Gamma_5 \geq 1 \\ \Gamma_2 + \Gamma_3 + \Gamma_4 \geq 1 \\ \Gamma_3 + \Gamma_4 + \Gamma_5 \geq 1 \\ \Gamma_1 + \Gamma_2 + \Gamma_4 + \Gamma_5 \geq 1 \end{cases} \tag{5.29}$$

For example, first constrain implies that at least one PMU must be installed at either one of buses 1, 2 or 5 to protect bus 1. To optimize the number of PMU, the best option is to install one PMU in bus 2 since it is common in four constrain. Te other PMU can be installed in either of bus 3, 4 or 5. it should be considered that in our case study the objective for using optimal PMU placement is to increase the accuracy of the detector not just minimising the number of installed PMU.

5.6 Parallel Implementation of the Robust DSE Against FDI

In this section, a trust-aware scheme for DSE is proposed which is robust under FDI attack. In the first step, critical measurements are identified. Utilizing optimal PMU placement for critical measurements, a group of trusted buses are introduced into the network. Secondly, using historic data of the trusted buses normal activities a Markov-chain model representing the normal behavior of the network is created. Thus, given an observed sequence,

the system has to decide if there is a cyber-attack or system is under normal operation condition. The Euclidian distance of the results from Markov-chain model is then calculated. The higher the distance observed activities receive from the Markov-chain model of the normal profile, the more likely the observed activities are anomalies resulting from cyber-attacks, and vice versa. If the results match to data set with a probability less than 0.1 they are assumed to be corrupted with cyber-attack. In other worlds the probability of the attack increases when results match to data set with lower probability. Also, in case of the load change, the change in voltage magnitude or phase angle caused due to the load change can be predicted, so the model parameters can be adjusted to reflect the change in the voltage due to the load change.

5.6.1 Implementation of Robust DSE Against FDI on GPU

The proposed robust DSE combines several aspects of parallelism to utilize the full capability of GPUs as efficiently as possible. Initializations are done on the CPU. After that all of the data are transferred to the GPU for executing the robust DSE algorithm. In the first step, the traditional serial algorithm is converted into smaller independent tasks which results in task parallelism to be solved in parallel. All of the independent tasks in the three main steps of EKF are calculated in parallel to accelerate the algorithm.

In order to take advantages of SIMD-based architectures of GPUs for the basic computations data parallelism is used for matrix-vector and matrix-matrix products which are time consuming for large data-sets. By assigning each independent *for* loops to individual threads, the task can be executed in parallel by converting into a kernel. In the robust DSE algorithm, several tasks are composed of matrix-matrix and matrix-vector product or summations which can be assigned to an individual kernel to run in parallel. Each kernel is responsible for the calculation of that specific task. As the number of independent threads is a lot more than the CPU cores, this type of parallelization is not possible on the CPU. Sparse matrix-vector multiplication and sparse triangular solve is used for GPU implementation using cuSPARSE library [156]. Fig. 5.5 shows the flowchart of the proposed robust DSE method.

5.6.2 Experimental Results

To explore the efficiency of the GPU based robust DSE against FDI large-scale systems were constructed for simulations. IEEE 39-bus, IEEE 118-bus, and IEEE 2496-bus systems were implemented on the GPU for simulation studies. Case 3 is build by duplicating IEEE 39-bus system. The same hardware setup as previous chapters is used for case studies.

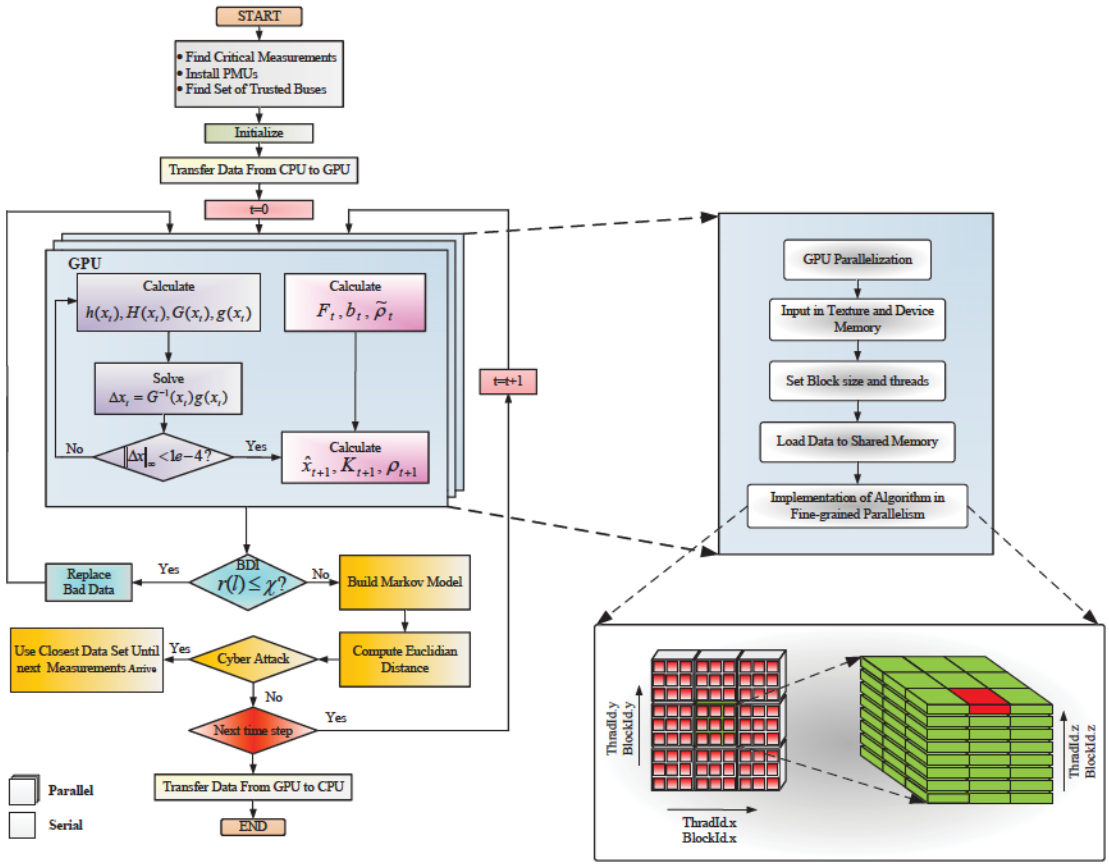


Figure 5.5: Flowchart of the proposed robust DSE method implemented on GPU.

Accuracy Analysis and BDD

In order to evaluate the accuracy of the proposed method, the results of state estimation in normal operation condition are plotted in Fig. 5.6 and Fig. 5.7. As there is no attack in the system, the result of state estimation is close enough to PSS/E® (real states). It is also shown in Fig. 5.7 that both LNR and proposed FDI test were resulting in lower values than the threshold indicating that there was no attack in the systems. The same experiment is performed for all case studies, however only results of IEEE-118 bus system, are plotted to save the space.

Attack Detection Analysis

In the second scenario, the proposed approach was evaluated for false data injection attack. The goal of the attack was to change the power injection at bus 22 by influencing the estimated values for the state variables at this bus in the IEEE 118-bus system shown in Fig. 5.8. For this attack to remain hidden other measurements have to be changed as well.

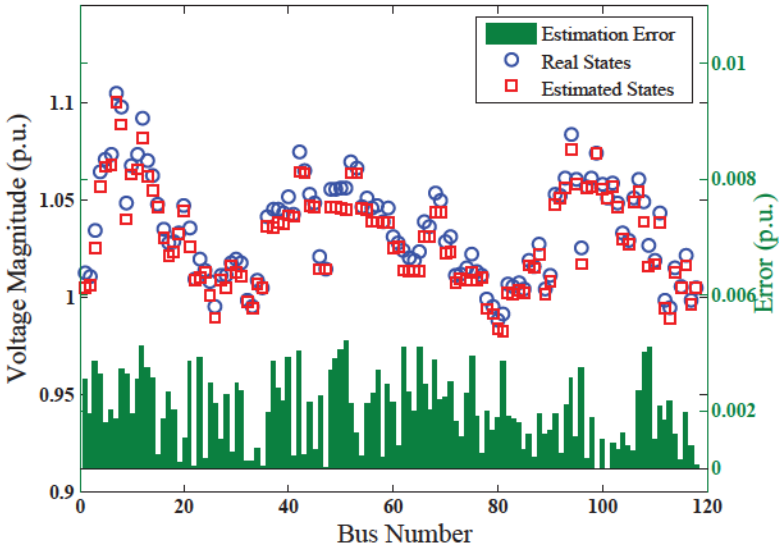


Figure 5.6: Voltage magnitude under normal operation condition.

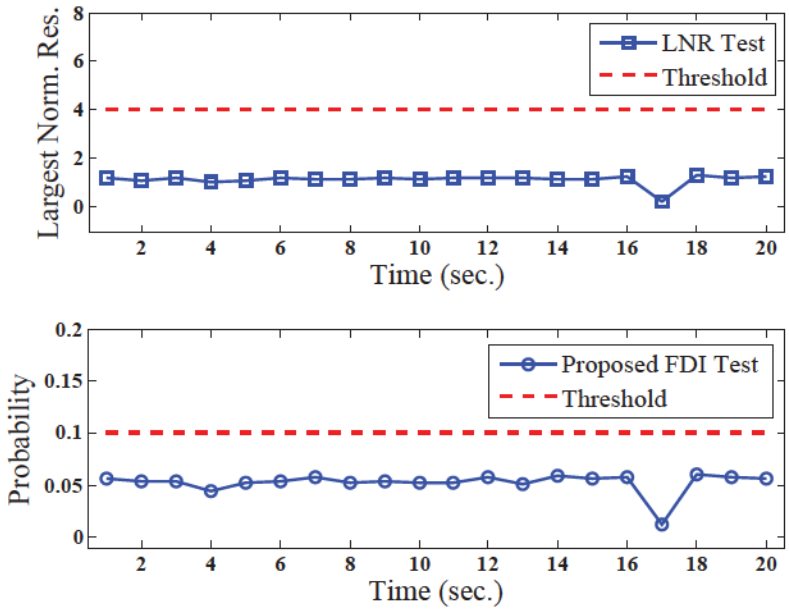


Figure 5.7: Detectors output along with threshold under normal operation condition.

In order to satisfy (5.15), power injections at buses 20 and 23 need to be changed. Also, the power flows on the 21-22 and 22-23 connecting lines need to be adjusted as well which will change the power flow on line 20-21. As a results the estimated value for bus 21 should also change to keep the attack hidden.

Fig. 5.9 shows the behavior of the LNR and proposed FDI test under the cyber-attack. It is clear from results that the estimation does not match with the measured values. During

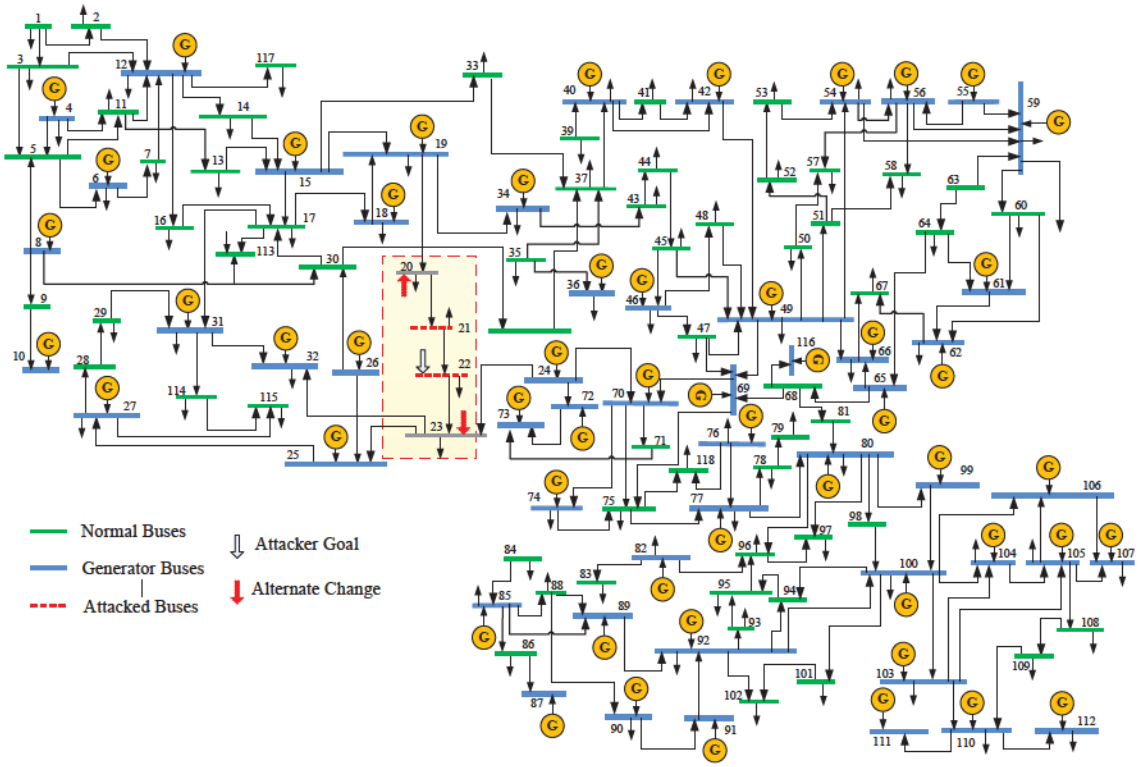


Figure 5.8: IEEE 118-bus power system.

this attack, the LNR detector resulted in values below threshold and thus it was not able to detect the attack in the system as shown in Fig. 5.10. However, in the same setup, the proposed detector exceeds the threshold; hence, the FDI attack can be detected. The results of LNR test in Fig. 5.10 show that this method can detect bad data using smaller threshold. However, it should be considered that from the first case study we know that there is not a bad data in the system. So the threshold for LNR was selected to filter 99.9% of the noise to only focus on cyber-attack. If we change the threshold to 3, in the first run LNR will detect a bad data so state estimation will run one more time after removing the bad data. As explained earlier for cyber-attack several measurements should change. So the trace of cyber-attack still exists in the system and LNR will not detect it this time. In other worlds, by selecting smaller threshold, instead of focusing on cyber-attack we just waste the time by redoing state estimation. The same experiment is performed for all case studies resulting in similar results, which proved the effectiveness of the proposed approach.

Computational Efficiency and Resource Distribution

In order to certify the efficiency of the proposed GPU-based robust DSE the speed-up ratio is defined as $S_p = T_{CPU}/T_{GPU}$, where T_{GPU} and T_{CPU} are the execution time of the serial algorithm running only on CPU and parallel algorithms on the GPU, respectively. As the

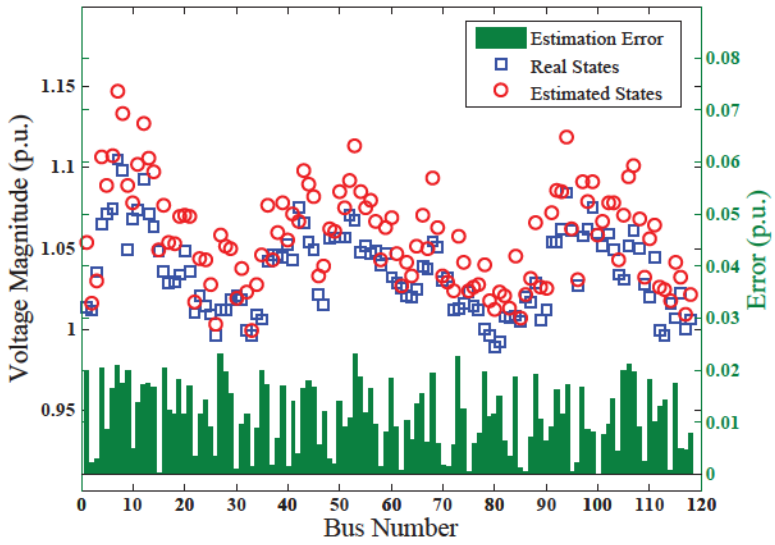


Figure 5.9: Voltage magnitude under FDI attack.

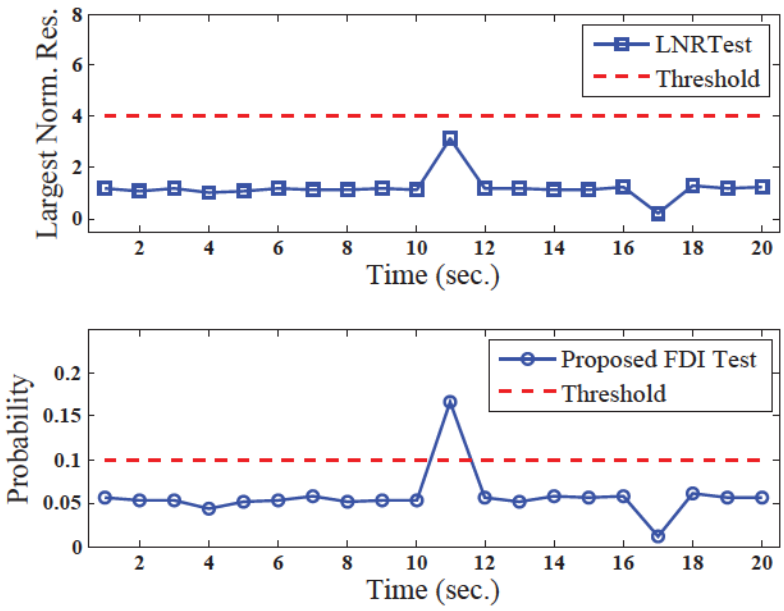


Figure 5.10: Detectors output along with threshold under FDI attack.

results reported in Table 5.1 show, the advantage of utilizing GPU for parallelization is significant when the size of the system is increased. It is obvious that execution time on CPU follows a high order complexity as the system size grows. However, the execution of the robust estimator on GPU almost increase linearly with respect to the system size as a result of fine grained parallelism on GPU. So it is expected to see higher speed up using larger case studies. Details of the case studies along with average estimation error for voltage magnitude ($E_V^{Ave.}$) and phase angle ($E_\delta^{Ave.}$) are shown in Table 5.1.

Table 5.1: Summary of DSE Results Under FDI Attack

Case	No. of buses	No. of meas.	Jacobian (H)	$E_V^{Ave.}$	$E_\delta^{Ave.}$	T_{CPU}	T_{GPU}	S_p
1	39	171	171×77	0.0027	0.059	0.31s	0.21s	1.47
2	118	609	609×235	0.0022	0.049	2.6s	0.55	4.7
3	2496	11553	11553×4991	0.0021	0.051	243s	31s	7.8

Based on Gustafson's law [130], the maximum achievable speed-up by parallelization is proportional to the number of CPU cores in the system. Unlike CPU, there is no fixed law to predict the maximum achievable speed-up using GPU. As it is obvious from results, more cores increases the processing power and throughput of the GPU and results in significantly faster algorithm. So it is expected to see even better performance using GPU for larger case studies which make it suitable for real implementation.

The resource distribution from CUDA on the Tesla S2050 GPU server is shown in Table 5.2. As can be seen from the results, the number of cores increase dynamically as the size of the system increases. Distribution of threads, blocks and memory varies in different kernels. Typically, the number of thread per block is a constant number which was 128 in our case studies. The number of blocks per grid is different based on the problem size in each case study. The maximum number of block per grid in each dimension was 16. The maximum number of grids for each Case study is reported in Table 5.2.

Table 5.2: GPU Resource Occupancy for MPDSE

Case	Occupancy	No. of cores	Max. no. of grids
1	51.4%	230	7
2	63.2%	283	53
3	84.8%	379	9384

5.7 Discussion

Overall, accuracy analysis verify that the proposed method can accuracy estimate the state of the system along with detecting the possible FDI attack. There are small differences compared to PSS/E® results which are justifiable considering the fact that the order of block execution in each GPU grid is undefined in kernel definition the same as explained

in previous chapters. It should be considered that this will not affect the resulted of detector since the threshold is set in a way to detect attacks with high probability. In rare cases the detector may report false attack however, it is always better to be prepared rather than paying expenses caused by a neglected attack.

In terms of speed-up, as it is shown the more GPU core results in more speed-up. It is possible to achieve higher speed-up using more cores however it should be considered that a method should keep computational efficiency and cost at the same time. Methods which works using costly hardware setup are not applicable on real system infrastructure. In our case study as the resource distribution shows it is possible to achieve higher speed-up even with the same number of cores.

The threshold for proposed detector was selected based on trial and error to decrease the chance of false alarm. It is possible to find a better threshold with detailed analysis, however in our case study the main goal was to show that Markov chain modeling is a good candidate for cyber-attack analysis. Selecting a different threshold will not change the fact that proposed method can detect FDI attack which bypass the traditional BDD techniques.

5.8 Summary

In this chapter, a robust parallel dynamic state estimation approach utilizing GPU and EKF was presented. As a solution to mitigate FDI attack, a new analytical technique is proposed based on the Markov chain theory and using Euclidian distance approach when the SCADA system is subject to a hidden FDI attack. The proposed approach can detect FDI attack using trusted set of measurements which was secured using optimized PMU installation. Considering the stochastic nature of the power system, using Markov chain theory and history of the system's dynamic behaviour a Markov model was prepared to check the accuracy of the estimation results using euclidian distance method. The estimated states are analyzed by calculating the Euclidian distance from the Markov model. States which match the lower probability are considered as attacked states.

Simulation results verify the accuracy of the proposed method both in normal operation condition and under FDI attack. It is shown that proposed method is able to detect the presents of malicious attacks which were undetectable by BDD methods.Moreover, large case studies along with parallel implementation on GPUs shows the speed and applicability of the proposed approach for real time large-scale power system.

6
Conclusions and Future Works

Continued growth in demand followed by system development and complex interconnections within the new smart grid paradigm has led to significant operational and control problems. These problems necessitate the need for major changes in computational resources for real-time action by system operators in energy control centers which are hard to achieve using traditional measurements provided by SCADA.

State estimation is a major requirement for safe operation and control of power systems. It is the core of EMS which is computationally very demanding for large-scale power system operation and control. The state estimation problem is rich in parallelism which makes it very suitable for utilizing massively parallel processing techniques. In addition, for secure operation of power system, robustness of state estimation against different type of fault and cyber-attack should be considered.

The investigation in this thesis branched into three directions. In part one the focus is the online static and dynamic state estimation. For this purpose, WLS state estimation was implemented on single GPU as a static and dynamic estimator. In the second part the relaxation method in conjunction with domain decomposition was proposed and successfully implemented on multiple GPU for the estimation of dynamic parameters of synchronous generator. In the last part, for cyber security analysis, results are verified through bad data identification and also modeled as a Markov chain to check the robustness against false data injection attack.

6.1 Contributions of Thesis

The main contributions of this thesis can be summarized as follows:

- The proposed methods accelerate state estimation for large-scale power systems using the general purpose computing capacity of the GPU and utilizing algorithm, task and data parallelism in different steps. The overall results revealed the advantage of GPU-based estimation over the CPU-based one for large-scale systems satisfying both accuracy and speed.

- The unique data collation technique by considering time synchronisation, data extrapolation and transformation resulting in a uniform set of measurements that were fed to the state estimator. Considering different weights for SCADA and PMU measurements along with the time synchronisation and extrapolation of SCADA measurements proposed method offers dynamic monitoring of the system behaviours.

- For the first time, a unified framework is investigated for GPU-based programming of dynamic state estimation. Although state estimation has a rich literature resources, in this thesis the effort was to aggregate all of the required equations for synchronous machine and network modeling, and step-by-step numerical methods to solve these equations.

- The relaxation method used in this work showed an efficient performance in multiple GPU implementation. Novel integration of the relaxation method with the coherency based partitioning and its implementation on GPU resulted in an inexpensive and efficient state estimator. In the GPU applications, the relaxation method was also very helpful to overcome the technology restriction existing in the present GPUs. By the use of this method it was possible to conduct the state estimation of large-scale power systems which are modeled in detail.

- The unique decomposition approach used in this thesis minimizes the effect of boundary buses in accuracy by exchanging data. It uses only local measurements for DSE in each subsystem which reduce the size of the problem and minimize data communication, localizes the effects of bad data to subsystems, and does not require either local observability or a central coordinator. Coherency analysis along with computational load balancing resulted in a highly efficient decomposition method for parallel programming.

- The proposed robust dynamic state estimation against false data injection attack localizes the effect of false data injection attacks. It identifies critical measurements and protect them by optimal PMU placement which again reduce the effect of cyber-attack. For the first time, using Markov chain modeling, the proposed method can

easily detect the malicious attacks which are not detectable by traditional BDD approaches.

6.2 Directions for Future Work

The following topics are proposed for future work:

- Since the programming structure is one of the most important factors which affects the execution time, it is possible to achieve faster results by different multi-thread programming paradigms.
- There exist many other possibilities for solving state estimation problem including least absolute value (LAV), maximum likelihood estimation (MLE), other existing iterative and non-iterative solvers which can be applied to state estimation problem.
- Proposed methods can be expanded to include more details in network or generator modeling. It is also possible to change the number of processor nodes or GPUs which are running in parallel.
- It is possible to implement other parallel processing based techniques to investigate higher speed-ups. It is predicted that if a method accelerates the CPU-based simulation, it would also accelerate the GPU-based model, if that approach was efficiently implemented on the GPU.
- It can play an important role in online dynamic cyber security assessment to detect the possible cyber attack as fast as possible which save time for further action. The cyber security analysis is an open research area which can take advantages of the GPUs' power.
- The application of the proposed method is not limited to the power system analysis, so future research can also be done to develop GPU-based algorithm for any dynamic system regardless of complication and the type of states or parameters that are needed to be estimated.

Bibliography

[1] A.Benigni, J. Liu, F. Ponci, A. Monti, G. Pisano, S. Sulis, "Decoupling power system state estimation by means of stochastic collocation", *IEEE Trans. on Instrumentation and Measurement,* vol. 60, no. 5, pp. 1623-1632, May 2011.

[2] U.S.-Canada Power System Outage Task Force, *Final report on the August 14, 2003 blackout in the United State and Canada: causes and recommendations,* April 2004.

[3] A. Teixeira, S. Amin, H. Sandberg, K. H. Johansson, and S. S. Sastry, "Cyber-security analysis of state estimators in electric power systems", *Proc. IEEE Conf. on Decision and Control,* March 2010.

[4] H. Wu, "PMU impact on state estimation reliability for improved grid security", *Proc. of IEEE PES General Meeting,* vol. 25, no. 1, pp. 1349-1351, 2006.

[5] A. Abur, A. Gómez-Expósito, "Power system state estimation theory and implementation", *Marcel Dekker, Inc.,* 2004.

[6] A. Jain, N. R. Shivakumar, "Power system tracking and dynamic state estimation", *Proc. of IEEE PES,* vol. 1, no. 8, pp. 15-18 Mar. 2009.

[7] A. S. Debs and R. E. Larson, "A dynamic estimator for tracking the state of the power system", *IEEE Trans. on Power App. and Syst.,* vol. 89, pp. 1670-1678, Sept. 1970.

[8] K. Stouffer, J. Falco, K. Kent, "Guide to supervisory control and data acquisition (SCADA) and industrial control systems security", *NIST Special Publication 800-82 (Initial Public Draft),* Sept. 2006.

[9] J. De La Ree, V. Centeno, J. S. Thorp, A. G. Phadke, "Synchronized phasor measurement applications in power systems", *IEEE Transactions on in Smart Grid,* vol. 1, no. 1, pp. 20-27, June 2010.

[10] T. D. Han, T. S. Abdelrahman, "hiCUDA: high-level GPGPU programming", *IEEE Trans. on Parallel and Distributed Systems,* vol. 22, no. 1, pp. 78-90, Jan. 2011.

[11] F . C. Schweppe and J. Wildes, "Power system static-state estimation, part I: exact model", *Proc. of Power Ind. Comp. Conf.,* 1969.

[12] F . C. Schweppe and J. Wildes, "Power system static-state estimation, part II: approximate model", *IEEE Trans. on Power App. and Syst.,* 1970.

[13] F . C. Schweppe and J. Wildes, "Power system static-state estimation, part III: implementation", *IEEE Trans. on Power App. and Syst.,* 1970.

[14] V. Cutsem, J. L. Howard and M. R. Pavella, "A two level static state estimator for electric power systems", *IEEE Trans. on Power App. and Syst.,* vol. 100, 1981.

[15] K. Clements, O. Denison, and R. Ringlee, "A multi-area approach to state estimation in power system networks", *Proc. of IEEE PES,* 1972.

[16] A. Garcia, A. Monticelli, and P. Abreu, "Fast decoupled state estimation and bad data processing", *IEEE Trans. on Power App. and Syst.,* vol. 98, no. 5, pp. 1645-1652, 1979.

[17] A. Monticelli and A. Garcia, "Fast decoupled state estimators", *IEEE Trans. on Power Syst.,* vol. 5, no. 2, pp. 556-564, May 1990.

[18] S. Y. Lin and C. H. Lin, "An implementable distributed state estimator and distributed bad data processing schemes for electric power systems", *IEEE Trans. on Power Syst.,* vol. 9, 1994.

[19] S. Y. Lin, "A distributed state estimator for electric power systems", *IEEE Trans. on Power Syst.,* vol. 7, 1992.

[20] A. A. El-Keib, J. Nieplocha, H. Singh, and D. J. Maratukulam, "A decomposed state estimation technique suitable for parallel processor implementation", *IEEE Trans. on Power Syst.,* vol. 7, no. 3, pp. 1088-1097, Aug. 1992.

[21] M. Hiroyuki, "An artificial neural net based method for power system state estimation", *Proc. of Int. Joint Conf. on Neural Networks,* 1993.

[22] H. Sasaki, K. Aoki and R. Yokoyama, "A parallel computation algorithm for static state estimation by means of matrix inversion lemma", *IEEE Trans. on Power Syst.,* vol. 2, no. 3, pp. 624-632, Aug. 1987.

[23] S. Iwamoto, M. Kusano and V.H. Quintana, "Hierarchical state estimation using a fast rectangular coordinate method", *IEEE Trans. on Power Syst.,* vol. 4, no. 3, pp. 870-879, Aug. 1989.

[24] D. M. Falciio, F. F. Wu and L. Murphy, "Parallel and distributed state estimation", *IEEE Trans. on Power Syst.,* vol. 10, 1995.

[25] R. Ebrahimian, and R. Baldick, "State estimation distributed processing", *IEEE Trans. on Power Syst.,* vol. 15, no. 4, pp. 1240-1246, Nov. 2000.

[26] C. G. Quiles, A. Villa and A. G. Exposito, "A factorized approach to WLS state estimation", *IEEE Trans. on Power Syst.,* vol. 26, no. 3, pp. 1724-1732, Aug. 2011.

[27] P. Rousseaux, V. Cutsem and M.R. Pavela, "Multi level dynamic state estimation for electric power systems", *Proc. of 8th power syst. compt. conf.,* pp. 19-24, 1984.

[28] H. Sun, Zh. Wang and D. Nikovski, "Two-level state estimation method for power systems with SCADA and PMU measurements", *Proc. of IEEE PES Conf. on Inn. Smart Grid Tech.,* vol. 1, no. 5, pp. 21-24, May 2012.

[29] T. V. Cutsem and M. Ribbens-Pavella, "Critical survey of hierarchical methods for state estimation of electrical power systems", *IEEE Trans. on Power App. and Syst.,* vol. 102, no. 10, pp. 3415-3424, Oct. 1983.

[30] G. N. Korres, and G. C. Contaxis, "Application of a reduced model to a distributed state estimator", *Proc. of IEEE PES,* vol. 2, pp. 999-1004, 2000.

[31] W. Shaobu,G. Wenzhong, A. P. S. Meliopoulos, "An alternative method for power system dynamic state estimation based on unscented transform", *IEEE Trans. on Power Syst.,* vol. 27, no. 2, pp. 942-950, May 2012.

[32] G. G. Rigatos, "A derivative-free Kalman filtering approach to state estimation-based control of nonlinear systems", *IEEE Trans. on Ind. Elec.,* vol. 59, no. 10, pp. 3987-3997, Oct. 2012.

[33] E. Ghahremani, I. Kamwa, "Dynamic state estimation in power system by applying the extended Kalman filter with unknown inputs to phasor measurements", *IEEE Trans. on Power Syst.,* vol. 26, no. 4, pp. 2556-2566, Nov. 2011.

[34] C. Huanyuan, L. Xindong, S. Caiqi, Y. Cheng, "Power system dynamic state estimation based on a new particle filter", *Proc. of IEEE PES,* pp. 1-7, July 2011.

[35] P. Rousseaux, D. Mallieu, V. Cutsem and M.R. Pavela, "Dynamic state prediction and hierarchical filtering for power systems state estimation", *Proc. of Automatica IFAC,* vol. 24, pp. 595-618, 1988.

[36] G. Durgaprasad and S. S. Thakur, "Robust dynamic state estimation of power systems based on M-estimation and realistic modeling of system dynamics", *IEEE Trans. on Power Syst.,* vol. 13, no. 4, pp. 1331-1336, Nov. 1998.

[37] G. Welch and G. Bishop, "SCAAT: incremental tracking with incomplete information", *Proc. of Comp. Graph. Conf.,* pp. 333-344, 1997.

[38] F. Shabani, N. R. Prasad, and H. A. Smolleck, "State estimation with aid of fuzzy logic", *Proc. of 5th IEEE Int. Conf. Fuzzy Syst.,* vol. 2, pp. 947-953, Sept. 1996.

[39] J. K. Mandal, A. K. Sinha, L. Roy, "Incorporating nonlinearity of measurement function in power system dynamic state estimation", *Proc. of IEE, Gen., Trans. & Dist.,* vol. 142, no. 3, pp. 289296, May 1995.

[40] K. R. Shih, S. J. Huang, "Application of a robust algorithm for dynamic state estimation of a power system", *IEEE Trans. on Power Syst.,* vol. 17, no. 1, pp. 141147, Feb. 2002.

[41] F. Aminifar, M. Shahidehpour, M. Fotuhi-Firuzabad, S. Kamalinia, "Power system dynamic state estimation with synchronized phasor measurements", *IEEE Trans. on Inst. and Meas.,* vol. 63, no. 2, pp. 352-363, Feb. 2014.

[42] L. Fan, Z. Miao, Y. Wehbe, "Application of dynamic state and parameter estimation techniques on real-world data", *IEEE Trans. on Smart Grid,* vol. 4, no. 2, pp. 1133-1141, June 2013.

[43] J. Zhang, G. Welch, G. Bishop, Z. Huang, "A two-stage kalman filter approach for robust and real-Time power system state estimation", *IEEE Trans. on Sust. Energy,* vol. 5, no. 2, pp.629-636, April 2014.

[44] N. Zhou, D. Meng, Z. Huang, G. Welch, "Dynamic state estimation of a synchronous machine using PMU data: a comparative study", *IEEE Trans. on Smart Grid,* vol.6, no.1, pp.450,460, Jan. 2015.

[45] L. Lin, Linawati, L. Jasa, E. Ambikairajah, "A hybrid state estimation scheme for power system", *Proc. IEEE Circuits and Systems Conf.,* vol. 1, pp. 555-558, 2002.

[46] M. Huanga, W. Li, and W. Yana, "Estimating parameters of synchronous generators using square-root unscented Kalman filter", *Int. J. Elect. Power Syst. Res.,* vol. 80, pp. 1137-1144, Sep. 2010.

[47] E. Farantatos, G. K. Stefopoulos, G. J. Cokkinides, and A. P. Meliopoulos, "PMU-based dynamic state estimation electric power systems", *Proc. of IEEE PES,* pp. 1-8, Sep. 2009.

[48] A. G. Phadke, J. S. Thorp and K. J. Karimi, "State estimation with phasor measurements", *IEEE Trans. on Power Syst.,* vol. 1, no. 1, pp. 233-238, Feb. 1986.

[49] S. Chakrabarti, D. Eliades, E. Kyriakides, and M. Albu, "Measurement uncertainty considerations in optimal sensor deployment for state estimation", *Proc. of IEEE Int. Symp. Intelligent Signal Proc.,* pp. 1-6, Oct. 2007.

[50] C. Yunzhi, H. Xiao, and G. Bei, "A new state estimation using synchronized phasor measurements", *Proc. of IEEE Int. Symp. Circ. and Syst.* , pp. 2817-2820, May 2008.

[51] C. Bruno, C. Candia, L. Franchi, G. Giannuzzi, M. Pozzi, R. Zaottini, and M. Zaramella, "Possibility of enhancing classical weighted least squares state estimation with linear PMU measurements", *Proc. of IEEE Power Tech.,* pp. 1-6, 2009.

[52] S. Chakrabarti and E. Kyriakides, "PMU measurement uncertainty considerations in WLS state estimation", *IEEE Trans. on Power Syst,* vol. 24, no. 2, pp. 1062-1071, May 2009.

[53] G. Valverde, S. Chakrabarti, E. Kyriakides, and V. Terzija, "A constrained formulation for hybrid state estimation", *IEEE Trans. on Power Syst.,* vol. 26, no. 3, pp. 1102-1109, Aug. 2011.

[54] L. Wu, L. Xia, "Research on data compatibility of PMU/SCADA mixed measurement state estimation", *Proc. of Comm. in Comp. and Inf. Scie.,* vol. 288, pp. 703-712 ,2012.

[55] Z. Ming, V. A. Centeno, J. S. Thorp, and A. G. Phadke, "An alternative for including phasor measurements in state estimators", *IEEE Trans. on Power Syst.,* vol. 21, no. 4, pp. 1930-1937, Nov. 2006.

[56] L. Zhao and A. Abur, "Multiarea state estimation using synchronized phasor measurements", *IEEE Trans. on Power Syst.,* pp. 611-617, May 2005.

[57] W. Jiang, V. Vittal, and G. T. Heydt, "A distributed state estimator utilizing synchronized phasor measurements", *IEEE Trans. on Power Syst.,* vol. 22, no. 2, pp. 563-571, May 2007.

[58] H. Xue, Q. Jia, N. Wang, Z. Bo, H. Wang, and H. Ma, "A dynamic state estimation method with PMU and SCADA measurement for power systems", *Proc. of Int. Power Eng. Conf.,* pp. 848-853, Dec. 2007.

[59] W. Jiang, V. Vittal, G. T. Heydt, "Diakoptic state estimation using phasor measurement units", *IEEE Trans. on Power Syst.,* pp. 1580-1589, Nov. 2008.

[60] K. Das, J. Hazra, D. P. Seetharam, R. K. Reddi, A. K. Sinha, "Real-time hybrid state estimation incorporating SCADA and PMU measurements", *Proc. of IEEE PES Conf. on Inn. Smart Grid Tech.,* vol. 1, no. 8, pp. 14-17 Oct. 2012.

[61] E. Farantatos, G. K. Stefopoulos, G. J. Cokkinides, and A. P. Meliopoulos, "PMU based dynamic state estimation for electric power systems", *Proc. of IEEE PES,* pp. 1-8, Jan. 2009.

[62] T. Yang, H. B. Sun, and A. Bose, "Transition to a two-level linear state estimator part II: algorithm", *IEEE Trans. on Power Syst.,* vol. 26 ,no. 1, Feb. 2011.

[63] S. M. Amin and B. F. Wollenberg, "Toward a smart grid: power delivery for the 21st century", *Proc. of IEEE PES,* vol. 3, no. 5, pp. 34-41, Sep. 2005.

[64] K. Moslehi and R. Kumar, "Smart grid - a reliability perspective", *Proc. of Inn. Smart Grid Tech. Conf.,* pp. 1-8, Jan. 2010.

[65] I. Dzafic, S. Henselmeyer, and H. T. Neisius, "High performance state estimation for smart grid distribution network operation", *Proc. of IEEE PES Conf. on Inn. Smart Grid Tech.,* pp. 1-6, Jan. 2011.

[66] P. Du, Z. Huang, Y. Sun, R. Diao, K. Kalsi, K. Anderson, Y. Li, and B. Lee, "Distributed dynamic state estimation with extended Kalman filter", *Proc. of North Amer. Power Symp.,* pp. 1-6, Aug. 2011.

[67] C. G. Quiles, A. G. Exposito and A. Villa, "State estimation for smart distribution substations", *IEEE Trans. on Smart Grid,* vol. 3, no. 2, pp. 986-995, Jun. 2012.

[68] A. G. Expsito, A. Villa, C. G. Quiles, P. Rousseaux, and V. Cutsem, "A taxonomy of multi-area state estimation methods", *Proc. of Elect. Power Syst. Res.,* vol. 81, pp. 1060-1069, Apr. 2011.

[69] N. Kashyap, S. Werner, T. Riihonen, Y. F. Huang, "Reduced-order synchrophasor assisted state estimation for smart grids", *Proc. of IEEE 3rd Int. Conf. on Smart Grid Comm.,* vol. 605, no. 610, pp. 5-8, Nov. 2012.

[70] L. Xie, D. H. Choi S. Kar, H. V. Poor, "Fully distributed state estimation for wide-area monitoring systems", *IEEE Trans. on Smart Grid,* vol. 3, no. 3, pp. 1154-1169, Sept. 2012.

[71] K. Ch. Sou, H. Sandberg, K. H. Johansson, "On the exact solution to a smart grid cyber-security analysis problem", *IEEE Trans. on Smart Grid,* vol. 4, no. 2, pp. 856-865, Jun. 2013.

[72] O. Kosut, L. Jia, R. Thomas, and L. Tong, "Malicious data attacks on the smart grid", *IEEE Trans. on Smart Grid,* vol. 2, pp. 645-658, 2011.

[73] S. Zonouz, K. M. Rogers, R. Berthier and T.J. Overbye, "SCPSE: security-oriented cyber-physical state estimation for power grid critical infrastructures", *IEEE Trans. on Smart Grid,* vol. 3, no. 4, pp. 1790-1799, Dec. 2012.

[74] A. Abur and A. G. Expoosito, "Bad data identification when using ampere measurements", *IEEE Trans. on Power Syst.,* vol. 12, no. 2, May 1997.

[75] N. Xiang, S. Wang, and E. Yu, "A new approach for detection and identification of multiple bad data in power system state estimation", *IEEE Trans. on Power App. and Syst.,* vol. 101, no. 2, pp. 454-462, Feb. 1982.

[76] A. Monticelli and A. Garcia, "Reliable bad data processing for real-time state estimation", *IEEE Trans. on Power App. and Syst.,* vol. 102, no. 5, pp. 1126-1139, May 1983.

[77] L. Milli, T. V. Cutsem, and M. R. Pavella, "Bad data identification methods in power system state estimation", *IEEE Trans. on Power App. and Syst.,* vol. 103, no. 11, pp. 3037-3049, Nov. 1985.

[78] A. Monticelli, F. F. Wu, and M. Y. Multiple, "Bad data identification for state estimation by combinatorial optimization", *IEEE Trans. on Power Del.,* vol. 1, no. 3, pp. 361-369, July 1986.

[79] E. N. Asada, A. V. Garcia, and R. Romero, "Identifying multiple interacting bad data in power system state estimation", *Proc. of IEEE PES,* pp. 571-577, Jun. 2005.

[80] Y. Liu, M. K. Reiter, and P. Ning, "False data injection attacks against state estimation in electric power grids", *Proc. of ACM Conf. on Comp. and Comm. Sec.,* pp. 21-32, Nov. 2009.

[81] A. R. Metke and R. L. Ekl, "Security techn ology for smart grid networks", *IEEE Trans. on Smart Grid,* vol. 1, no. 1, pp. 99-107, Jun. 2010.

[82] G. N. Ericsson, "Cyber security and power system communication essential parts of a smart grid infrastructure", *IEEE Trans. on Power Del.,* vol. 25, no. 3, pp. 1501-1507, Jul. 2010.

[83] L. Xie, Y. Mo, and B. Sinopoli, "Integrity data attacks in power market operations", *IEEE Trans. on Smart Grid,* vol. 2, no. 4,pp. 659-666, 2011.

[84] G. Hug, J. A. Giampapa, "Vulnerability assessment of AC state estimation with respect to false data injection cyber-attacks", *IEEE Trans. on Smart Grid,* vol. 3, no. 3, pp. 1362-1370, Sept. 2012.

[85] T. T. Kim and H. V. Poor, "Strategic protection against data injection attacks on power grids", *IEEE Trans. on Smart Grid,* vol. 2, pp. 326-333, Jun. 2011.

[86] O. Kosut, L. Jia, R. Thomas, and L. Tong, "Malicious data attacks on the smart grid", *IEEE Trans. on Smart Grid,* vol. 2, pp. 645-658, 2011.

[87] S. Ntalampiras, "Detection of integrity attacks in cyber-physical critical infrastructures using ensemble modeling", *IEEE Trans. on Industrial Informatics,* vol. 11, no. 1, pp. 104-111, Feb. 2015.

[88] L. Xie, Y. Mo, B. Sinopoli, "Integrity data attacks in power market operations", *IEEE Trans. on Smart Grid,* vol. 2, no. 4, pp. 659-666, 2011.

[89] S. Sridhar, G. and Manimaran, "Data integrity attacks and their impacts on SCADA control system", *Proc. of IEEE PES,* pp. 1-6, July 2010.

[90] F. Cleveland, "Cyber security issues for advanced metering infrastructure", *Proc. of IEEE PES,* pp. 1-5, July 2008.

[91] A. Srivastava, T. Morris, T. Ernster, C. Vellaithurai, P. Shengyi and U. Adhikari, "Modeling cyber-physical vulnerability of the smart grid with incomplete information", *IEEE Trans. on Smart Grid,* vol. 4, no. 1, pp. 235-244, Mar. 2013.

[92] G. Dan, H. Sandberg, "Stealth attacks and protection schemes for state estimators in power systems", *in Proc. 1st IEEE Int. Conf. Smart Grid Commun.,* pp. 214-219, Jun. 2010.

[93] S. Cui, Z. Han, S. Kar, T. T. Kim, H. V. Poor, A. Tajer, "Coordinated data-injection attack and detection in the smart grid: a detailed look at enriching detection solutions", *IEEE Signal Process. Mag.,* vol. 29, no. 5, pp. 106115, Sep. 2012.

[94] R. B. Bobba, K. M. Rogers, Q. Wang, H. Khurana, K. Nahrstedt, and T. J. Overbye, "Detecting false data injection attacks on dc state estimation", *Proc. of 1st Workshop on Sec. Cont. Syst.,* 2010.

[95] K. A. Loparo and F. F. Abdel-Malek, "A probabilistic approach to dynamic power system security", *IEEE Trans. on Circ. and Syst.,* vol. 37, no. 6, pp. 787-799, June 1990.

[96] F. F. Wu and Y.-K. Tsai, "Probabilistic dynamic security assessment of power systems: Part I- basic models", *IEEE Trans. on Circ. and Syst.,* vol. 30, no. 3, pp. 148-159, March 1983.

[97] R. M. Kolacinski and K. A. Loparo, "A mathematic framework for analysis of complex cyber-physical power systems", *Proc. of IEEE PES,* pp. 1-8, July 2012.

[98] J. M. Ortega and W. C. Rheinboldt, "Iterative solution of nonlinear equations in several variables", *Academic Press,* 1970.

[99] A. R. Newton and A. S. Vincentelli, "Relaxation-based electrical simulation", *IEEE Trans. Elec. Dev.,* vol. 30, no. 9, pp. 1184-1207, Sept. 1983.

[100] M. I. Spong, M. L. Crow and M. A. Pai, "Transient stability simulation by waveform relaxation methods", *IEEE Trans. Power Syst.,* vol. 2, no. 4, pp. 943-952, Nov. 1987.

[101] M. L. Crow, "Waveform relaxation methods for the simulation of systems of differential/ algebraic equations with application to electric power systems", *Ph.D. Dissertation,* University of Illinois, 1990.

[102] L. Hou and A. Bose, "Implementation of the waveform relaxation algorithm on a shared memory computer for the transient stability problem", *IEEE Trans. Power Syst.,* vol. 12, no. 3, pp. 1053-1060, Aug. 1997.

[103] Y. Nakazono and H. Asai, "Application of Relaxation-Based technique to ADI-FDTD method and its estimation", *Proc. of IEEE Int.Symp. on Circ. and Syst.,* pp. 1489-1492, May 2007.

[104] F. Dorfler, F. Pasqualetti, F. Bullo, "Distributed detection of cyber-physical attacks in power networks: a waveform relaxation approach", *Proc. of 49th Ann. Allerton Conf. on Comm. cont. and Comp.,* pp. 1486-1491, 2011.

[105] V. J. Marandi, V. Dinavahi, "Instantaneous relaxation-based real-time transient stability simulation", *IEEE Trans. on Power Syst.,* vol. 24, no. 3, pp. 1327-1336, Aug. 2009.

[106] T. F. Chan, and D. Goovaerts, "On the relationship between overlapping and nonoverlapping domain decomposition methods", *IAM Journal on Matrix Analy. and App.,* vol. 13, no. 2, April 1992.

[107] B. Smith, "Domain decomposition: parallel multilevel methods for elliptic partial differential equations", *Cambridge University Press,* 2004.

[108] H. Karimipour, and V. Dinavahi, "Accelerated parallel WLS state estimation for large-scale power systems on GPU", *In Proc. of North American Power Symposium (NAPS),* pp. 1-6, Sept. 2013.

[109] R. Ebrahimian, and R. Baldick, "State estimation distributed processing", *IEEE Trans. on Power Syst.,* vol. 15, no. 4, pp. 1240-46, Nov. 2000.

[110] W. Jiang, V. Vittal, and G. T. Heydt, "Diakoptic state estimation using phasor measurement units", *IEEE Trans. on Power Syst.,* vol. 23, no. 4, pp. 15801589, Nov. 2008.

[111] B. A. Carre, "Solution of load-flow problems by partitioning systems into trees", *IEEE Trans. on Power App. Syst.,* vol. PAS-87, no. 11, pp. 1931-1968, Nov. 1968.

[112] L. Xie, D.H. Choi, S. Kar, "Cooperative distributed state estimation: Local observability relaxed", *Proc. of IEEE PES,* pp. 1-11, July 2011.

[113] Y. Weng, Q. Li, R. Negi, M. Ilic, "Distributed algorithm for SDP state estimation", *Proc. of Innovative Smart Grid Tech.,* pp. 1-6, Feb. 2013.

[114] A. Gopal, D. Niebur, S. V. subramanian, "DC power flow based contingency analysis using graphics processing units", *Proc. of IEEE Power Tech,* pp. 731-736, Jul. 2007.

[115] N. Garcia, "Parallel power flow solutions using a bi-conjugate gradient algorithm and a Newton method: a GPU-based approach", *Proc. of IEEE PES,* pp. 1-4, Jul. 2010.

[116] C. Vilacha, J. C. Moreira, E. Miguez and A. F. Otero, "Massive Jacobi power flow based on SIMD-processor", *Proc. of 10th Int. Conf. on Env. and Elec. Eng.,* pp. 1-4, May 2011.

[117] V. J. Marandi, V. Dinavahi, "SIMD-based large-scale transient stability simulation on the graphics processing unit", *IEEE Trans. on Power Syst.,* vol. 25, no. 3, pp. 1589-1599, Aug.2010.

[118] V. J. Marandi, Z. Zhou, V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs", *IEEE Trans. on Paral. and Dist. Syst.,* vol. 23, no. 7, pp. 1255-1266, Jul. 2012.

[119] M. J. Flynn, "Very high speed computing systems", *Proc. of the IEEE,* pp. 1901-1909, Dec. 1966.

[120] J. R. Gurd, "A taxonomy of parallel computer architectures", *Proc. of Int. Conf. on Des. and App. of Paral. Digit. Proc.,* pp. 57-61, Apr. 1988.

[121] NVIDIA, "NVIDIA Tesla: a unified graphics and computing architecture", *NVIDIA CUDA C Programming Guide 4.0.,* 2013.

[122] Z. Li, V. D. Donde, J. C. Tournier and F. Yang, "On limitations of traditional multi-core and potential of many-core processing architectures for sparse linear solvers used in large-scale power system applications", *Proc. of IEEE PES,* pp. 1-8, Jul. 2011.

[123] B. Chapman, G. Jost, R. van der Pas, "Using OpenMP: portable shared memory parallel programming", *IEEE Trans. on Parallel and Distributed Systems,* vol. 22, no. 1, pp. 78-90, Jan. 2011.

[124] NVIDIA, "CUDA C Programming guide", Feb. 2014.

[125] J. E. Stone, D. Gohara, G. Shi, "OpenCL: a parallel programming standard for heterogeneous computing systems", *Computing Sci. Eng.,* vol. 12, no. 3, pp. 66-72, May. 2010.

[126] J. Zhang, G. Welch, G. Bishop, "LoDiM: a novel power system state estimation method with dynamic measurement selection", *Proc. of IEEE PES,* pp. 1-7, July 2011.

[127] V. J. Marandi, V. Dinavahi, "Instantaneous relaxation-based real-time transient stability simulation", *IEEE Trans. on Power Syst.,* vol. 24, no. 3, pp. 1327-1336, Aug. 2009.

[128] S. Chakrabarti, E. Kyriakides, G. Ledwich, and A. Ghosh, "Inclusion of PMU current phasor measurements in a power system state estimator", *IET Generation, Transmission, and Distribution,* vol. 4, no. 10, pp. 1104-1115, Sept. 2010.

[129] D. Blythe, "Rise of the graphics processor", *Proc. of IEEE,* vol. 96, no. 5, pp. 761-778, May 2008.

[130] J. L. Gustafson, "Reevaluating Amdahl's law", *Communications of the ACM,* vol. 31, no. 5, pp. 532-533, Jan. 1988.

[131] N. Husted , S. Myers , A. shelat , P. Grubbs , "GPU and CPU parallelization of honest-but-curious secure two-party computation", *Proc. of* 29^{th} *Computer Security Applications,* pp. 169-178, 2013.

[132] M. Glavic, T. Van Cutsem, "Reconstructing and tracking network state from a limited number of synchrophasor measurements", *IEEE Trans. on Power Syst.,* vol. 28, no. 2, pp. 1921-1929, 2013.

[133] M. Gol, A. Abur, "Rapid tracking of bus voltages using synchro-phasor assisted state estimator", *Proc. of IEEE ISGT-Europe,* pp. 1-5, Oct. 2013.

[134] Y. Chen, Sh. Jin, M. Rice, Zh. Huang, "Parallel state estimation assessment with practical data", *Proc. of Power and Energy Society General Meeting (PES),* pp. 1-5, 2013.

[135] Y. Chakhchoukh, V. Vittal, G. T. Heydt, "PMU based state estimation by integrating correlation", *IEEE Trans. on Power Syst.,* vol. 29, no. 2, pp. 617-626, 2014.

[136] A. M. Leite da Silva, M. B. Do Coutto Filho, and J. F. de Queiroz, "State forecasting in electric power systems", *Proc. of Gen., Trans. and Distr.,* vol. 130, no. 5, pp. 237-244, Sept. 1983.

[137] W. Ding, J. Wang, C. Rizos, "Improving adaptive Kalman estimation in GPS/INS integration", *SIAM,* 2008.

[138] T. A. Davis, "Direct methods for sparse linear systems", *SIAM, Philadelphia, PA,* 2006.

[139] Y. Saad, "Iterative methods for sparse linear systems", *The Journal of Navigation,* vol. 60, pp. 517-529, 2007.

[140] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain", *Pittsburgh, PA, USA, Tech. Rep.,* 1994.

[141] M. Benzi, "Preconditioning techniques for large linear systems: a survey", *Journal of Comput. Phys.,* vol. 182, pp. 418-477, 2002.

[142] M. Asprou, E. Kyriakides, M. Albu, "The effect of variable weights in a WLS state estimator considering instrument transformer uncertainties", *IEEE Trans. on Instr. and Measur.,* vol. 63, no. 6, pp. 1484-1495, June 2014.

[143] R. G. Brown, "Exponential smoothing for predicting demand", *Cambridge, Massachusetts: Arthur D. Little Inc.,* 1956.

[144] J. J. Shynk, "Probability, Random Variables, and Random Processes: theory and signal processing applications", *Wiley-Interscience Press.,* 2012.

[145] S. B. Yusof, G.J. Rogers, and R.T.H. Alden, "Slow coherency based network partitioning ncluding load buses", *IEEE Trans. Power Syst.,* vol. 8, no. 3, pp. 1375-1382, Aug. 1993.

[146] H. You, V. Vittal, and X. Wang, "Slow coherency-based islanding", *IEEE Trans. Power Syst.,* vol. 19, no. 1, pp. 483-491, Feb. 2004.

[147] X. Wang, V. Vittal, G. T. Heydt, "Tracing generator coherency indices using the continuation method: a novel approach", *IEEE Trans. Power Syst.,* vol. 20, no. 3, pp. 1510-1518, Aug. 2005.

[148] A. Quarteroni and A. Valli, "Domain decomposition methods for partial differential equations", *Oxford University Press,* 1999.

[149] A. Toselli and O. Widlund, "Domain decomposition methods, algorithms and theory", *Springer-Verlag,,* 2005.

[150] R. R. Nucera, V. Brandwajn, and M. L. Gilles, "Observability analysis and bad data analysis using augmented blocked matrices", *IEEE Trans. Power Syst.,* vol. 8, no. 2, pp. 426433, May 1993.

[151] P. M. Anderson, A. A. Fouad, "Power system control and stability", *Iowa State University Press,* 1977.

[152] IEEE Std 421.5-2005, "IEEE recommended practice for excitation system models for power system stability studies", *IEEE Power Eng. Soc.,* pp. 1-85, 2006.

[153] M. La Scala, A. Bose, "Relaxation/newton methods for concurrent time step solution of differential-algebraic equations in power system dynamic simulation", *IEEE Trans. Circ. Syst.,* vol. 40, no. 5, pp. 317-330, May 1993.

[154] S. H. Roosta, "Parallel processing and parallel algorithms: theory and computation", *Springer-Verlag New York, Inc.,* 1999.

[155] C. T. Leondes, "Control and dynamic systems: analysis and control system", *Academic press Inc.,* 1991.

[156] NVIDIA, "CUSPARSE library", *NVIDIA Developer,* Feb. 2013.

[157] NVIDIA, "CUBLAS library", *NVIDIA Developer,* Aug. 2014.

[158] S. Azizi, A. S. Dobakhshari,S. A. Nezam Sarmadi, A. M. Ranjbar, "Optimal PMU placement by an equivalent linear formulation for exhaustive search", *IEEE Trans. on Smart Grid,* vol. 3, no. 1, pp. 174-182, March 2012.

[159] A. Gómez-Expósito, A. Abur, "Generalized observability analysis and measurement classification", *IEEE Trans. on Power Syst.,* vol. 13, no. 3, pp. 10901096, Aug. 1998.

[160] T. Vollmer, M. Manic, "Cyber-physical system security with deceptive virtual hosts for industrial control networks", *IEEE Trans. in Industrial Informatics*, vol. 10, no. 2, pp. 1337-1347, May 2014.

[161] A. Abur, "A bad data identification method for linear programming state estimation", *IEEE Trans. on Power Syst.*, vo1. 5, no. 3, pp. 894-900, Aug. 1990.

[162] A. Monticelli and A. Garcia, "Reliable bad data processing for real-time state estimation", *IEEE Trans. on Power App. and Syst.*, vol. 102, no. 5, pp. 1126-1139, May 1983.

[163] A. Teixeira, S. Amin, H. Sandberg, K. H. Johansson, S. S. Sastry, "Cyber security analysis of state estimators in electric power systems", *in Proc. of 49th IEEE Conf. in Dec. and Cont.*, pp. 5991-5998, Dec. 2010.

[164] S. P. Meyn, R. L. Tweedie, "Markov chain and stochastic stability", *Springer-verlag.*, 2005.

[165] J. Chen, A. Abur, "Improved bad data processing via strategic placement of PMUs", *in Proc. of PES General Meeting*, vol. 1, pp. 509-513, June 2005.

A Generator Model

The ninth-order order state-space model of the generator can be written as follows:

$$\begin{aligned}
\dot{x}_{g1}(t) &= \omega_R.x_{g2}(t),\\
\dot{x}_{g2}(t) &= \frac{1}{2S}[T_e(t) + T_m - D.x_{g2}(t)],\\
\dot{x}_{g3}(t) &= \omega_R.[e_{fd}(t) - R_{fd}.I_{fd}(t)],\\
\dot{x}_{g4}(t) &= -\omega_R.R_{1d}.I_{1d}(t),\\
\dot{x}_{g5}(t) &= -\omega_R.R_{1q}.I_{1q}(t),\\
\dot{x}_{g6}(t) &= -\omega_R.R_{2q}.I_{2q}(t),\\
\dot{x}_{g7}(t) &= \frac{1}{T_R}[v_t - x_{g7}(t)],\\
\dot{x}_{g8}(t) &= \frac{1}{T_2}[T_1K_{stab}.\dot{x}_{g2} - x_{g8} + (1 - \frac{T_1}{T_W})\alpha],\\
\dot{x}_{g9}(t) &= \frac{1}{T_A}[\beta - x_{g9}].
\end{aligned} \tag{A.1}$$

T_m and T_e represent mechanical input torque, and electrical output torque, respectively. e_{fd} and I_{fd} are field voltage and current. I_{1d}, I_{1q}, and I_{2q} describe d and q-axis currents. For the whole system, according to the aforementioned formulations the $9\times l$ vector of state variables x_g of the synchronous generator is given as:

$$\dot{x}_g = [\delta, \Delta\omega, \psi_{fd}, \psi_{1d}, \psi_{1q}, \psi_{2q}, v_1, v_2, v_3]^T, \tag{A.2}$$

where δ and $\Delta\omega$ represent vector of rotor speed and angle, respectively. $\psi_{fd}, \psi_{1d}, \psi_{1q}, \psi_{2q}$ shows vector of rotor flux linkages and v_1, v_2, v_3 are vector of exciter voltages. v_t represents the vector of terminal voltage which can be calculated as network state. For a system

with m generators, all of aforementioned vectors are $l \times 1$. Fig. A.1 shows AC5A type excitation system [152].

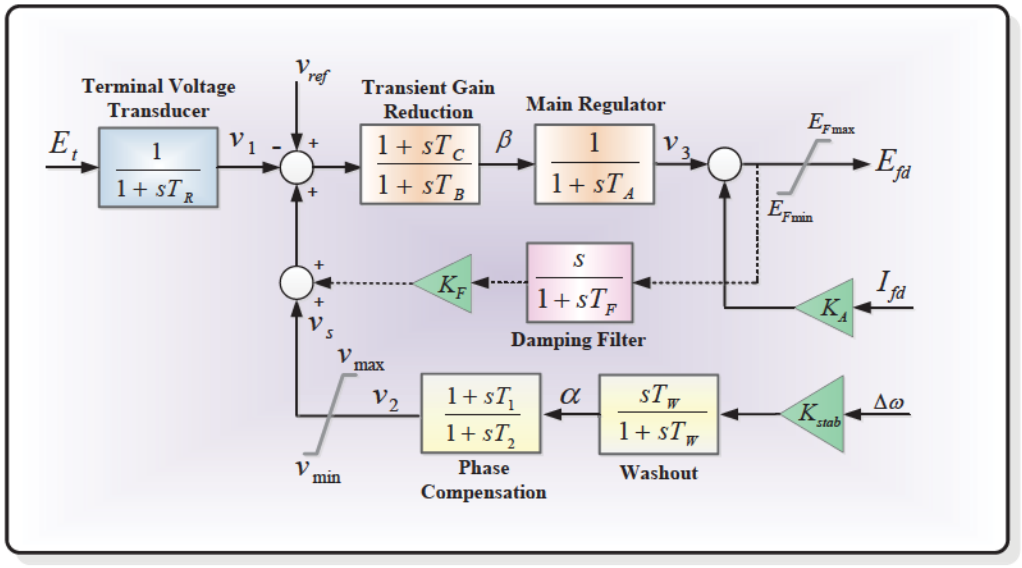


Figure A.1: Synchronous generator excitation system with AVR and PSS.

The output electrical torque T_e of the machine can be written as:

$$T_e = -[\psi_{ad} I_q - \psi_{aq} I_d], \tag{A.3}$$

where

$$\begin{aligned} \psi_{ad} &= L''_{ad}[-I_d + \frac{\psi_{fd}}{L_{fd}} + \frac{\psi_{d1}}{L_{d1}}], \\ \psi_{aq} &= L''_{aq}[-I_d + \frac{\psi_{q1}}{L_{q1}} + \frac{\psi_{q2}}{L_{q2}}]. \end{aligned} \tag{A.4}$$

where ω_R, S, D, R_{fd}, R_{1d}, R_{1q}, R_{2q}, L_{fd}, L_{d1}, L_{q1}, L_{q2}, L''_{ad}, L''_{aq}, T_R, T_w, T_1, T_2, and K_{stab} are constant system parameters whose definition can be found in [151].

B

TESLA Manufacturer Data Sheet

The specifications illustrated in this section are borrowed from [121].

B.1 SYSTEM CHASSIS

The Tesla S2050 use a 1U form factor chassis and conform to the EIA 310E specification for 19-inch 4-post racks with 900 mm to 1000 mm depth. The chassis dimensions are 1.73 inches high 17.5 inches wide 28.5 inches deep.

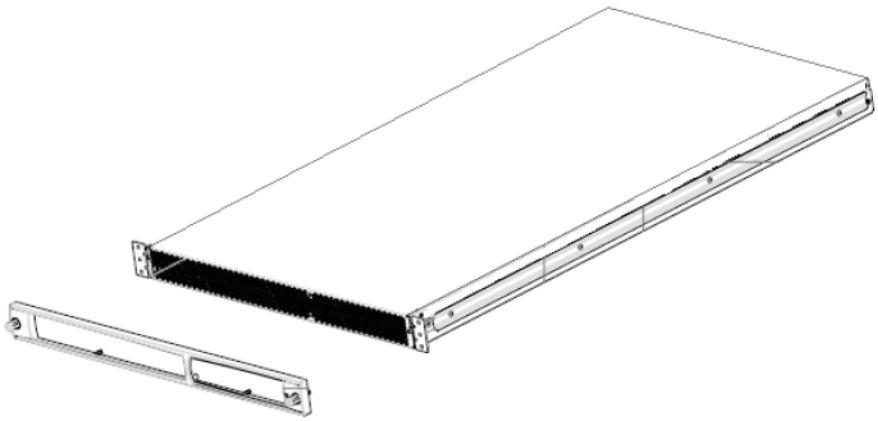


Figure B.1: System Chassis Drawing.

B.2 HOST INTERFACE CARD (HIC)

The HIC conforms to the PCI Express low profile form factor. This card is compatible with both PCI Express Gen1 and PCI Express Gen2 systems. A 8 version is also available for systems that do not have 16 PCI Express slots. The HICs ship with a full-height bracket installed and includes a low-profile bracket. Fig. B.2 shows the 16 version of the card with the full-height bracket.

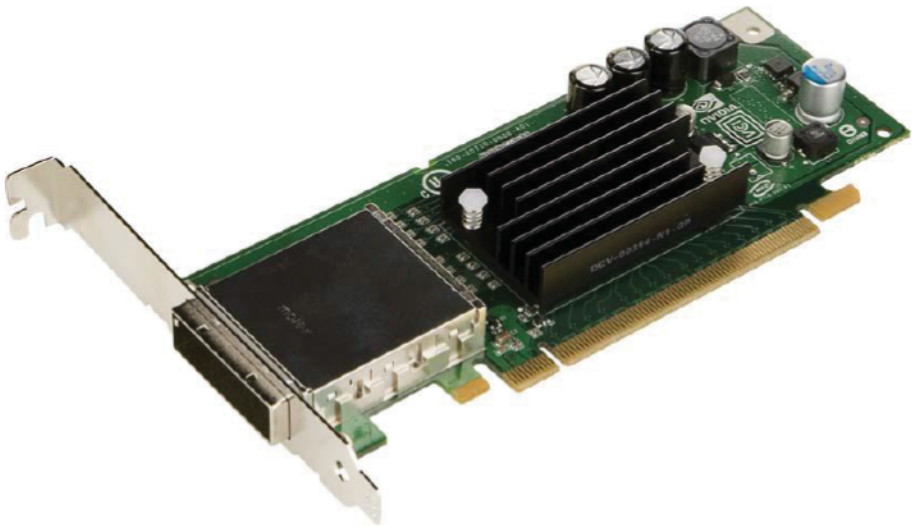


Figure B.2: Host Interface Card (x16 Version).

B.3 PCI EXPRESS CABLE

The Tesla S2050 use 0.5-meter PCI Express cables as the standard connection to the host system(s). Fig. B.3 shows the dimensions of this cable and its connectors. A 2.0-meter version of the cable is also available as a stand alone accessory and uses the same connectors as the 0.5-meter cable.

The minimum bend radius is 38.7 mm for the PCI Express cable. Fig. B.4 shows details of how this is measured relative to the I/O plate on the host interface card and relative to the cable/connector interface.

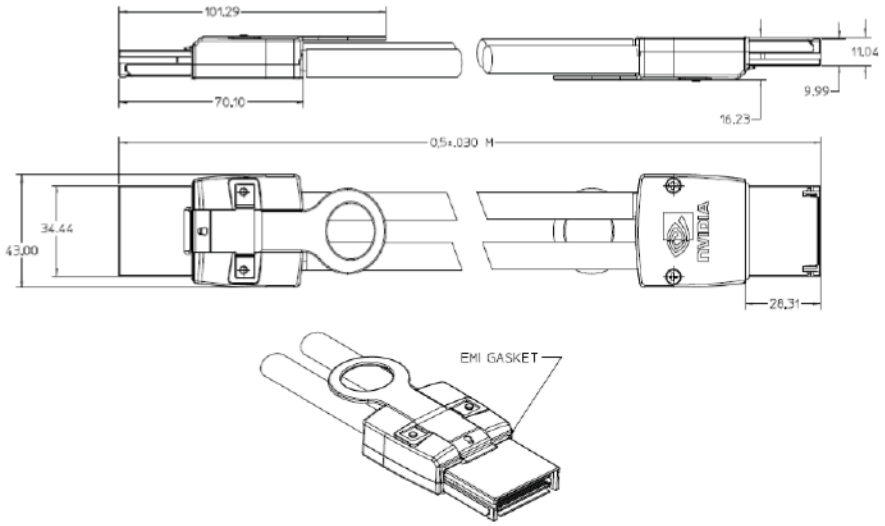
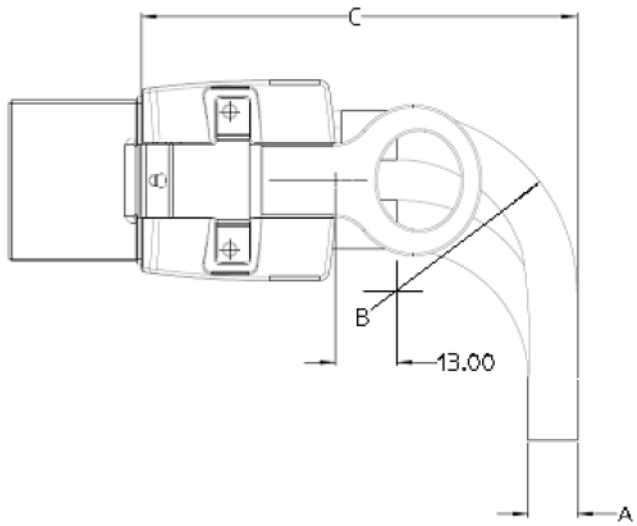


Figure B.3: PCI Express Cable (0.5 Meter).



	CABLE GAGE	30
A.	CABLE DIAMETER	8.6
B.	MINIMUM OUTER RADIUS	38.7
C.	FACEPLATE TO OUTER RADIUS	94

Figure B.4: PCI Express Cable Minimum Bend Radius.

B.4 ENVIRONMENTAL SPECIFICATIONS

Fig. B.5 shows the environmental specification of Tesla S2050.

Specifications		Conditions
Operating	Input power	Operating 100-240 VAC, maximum 90-264 VAC 50 to 60 Hz
	Temperature	10 °C to 35 °C (50 °F to 95 °F) at sea level with an altitude derating of 1.0 °C per every 1000 ft.
	Humidity	10 % to 80 % RH, 28 °C (82.4 °F) maximum wet bulb temperature, non-condensing
	Altitude	0 to 5000 feet mean sea level (MSL)
	Shock	Half sine 40 g, 2 ms duration
	Vibration	Sinusoidal 0.25g, 10 to 500 Hz, 3 axis. Random 1.0 Grms, 10 to 500 Hz
	Acoustics	TBD
	Airflow	143 CFM maximum
Non-operating	Temperature	-40 °C to 60 °C (-40 °F to 140 °F)
	Humidity	10 % to 80 % RH, 38.7 °C (101.7 °F) maximum wet bulb temperature, non-condensing
	Altitude	0 to 10,000 feet mean sea level (MSL) with maximum allowable rate of altitude change of 2000 ft/min.
	Shock	Half-sine: 80 G, 2ms Trapezoidal: 40 G, 150 in/sec
	Vibration (random)	0.015-0.008 G/Hz, 5-500 Hz, 10 minutes

Figure B.5: Environmental Specifications and Conditions.

C

Single Line Diagram of Test Systems

In this section the single-line diagram of the test systems used in the thesis are given. The *Scale* 1 system is IEEE 39-bus system (Fig. C.1) which its complete data and load flow results are also given in the PSS/E's *.raw file format. IEEE 39-bus system was duplicated and interconnected to create large-scale systems, whose dimension n is obtained as $n = 39 * 2^{S_c}$, where $S_c = \Gamma - 1$ with Γ being the test case index.

C.1 Scale 1

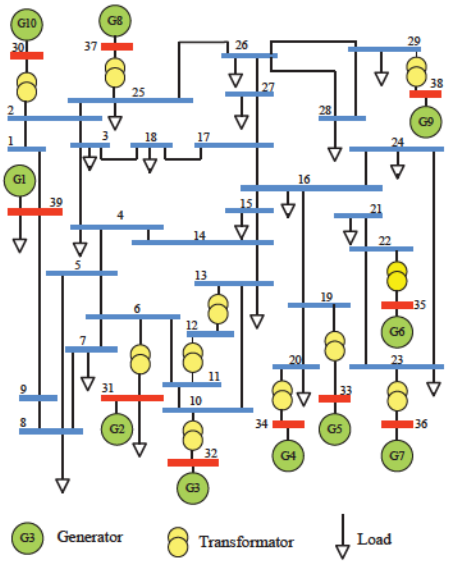


Figure C.1: Scale 1 system: 39 buses, 10 generators.

C.1.1 Load Data

```
     Bus         Pload          Qload
     Number      (MW)           (MVAR)
     1,            0.000,         0.000
     2,            0.000,         0.000
     3,          322.000,         2.400
     4,          500.000,        84.000
     5,            0.000,      -200.000
     6,            0.000,         0.000
     7,          233.800,       840.000
     8,          522.000,       176.000
     9,            0.000,         0.000
    10,            0.000,         0.000
    11,            0.000,         0.000
    12,            8.500,        88.000
    13,            0.000,         0.000
    14,            0.000,         0.000
    15,          320.000,       153.000
    16,          329.400,       323.000
    17,            0.000,         0.000
    18,          158.000,        30.000
    19,            0.000,         0.000
    20,          680.000,       103.000
    21,          274.000,       115.000
    22,            0.000,         0.000
    23,          247.500,        84.600
    24,          308.600,       -92.200
    25,          224.000,        47.200
    26,          139.000,        17.000
    27,          281.000,        75.500
    28,          206.000,        27.600
    29,          283.500,       126.900
    31,            9.200,         4.600
    39,         1104.000,       250.000
0 / END OF LOAD DATA, BEGIN GENERATOR DATA
```

C.1.2 Generator Data

```
    Bus      Pgen        Qgen        Qmax       Qmin        Vsched     Mbase        Rsource    Xsource
    Number   (MW)        (MVAR)      (MVAR)     (MVAR)      (pu)       (MVA)        (pu)       (pu)
     30,      250.000,    216.192,    800.000,  -500.000,    1.04750,   1000.000,    0.00140,   0.20000
     31,      572.930,    550.596,    800.000,  -500.000,    1.04000,   1000.000,    0.02700,   0.20000
     32,      650.000,    265.198,    800.000,  -500.000,    0.98310,   1000.000,    0.00386,   0.20000
     33,      632.000,    123.851,    800.000,  -500.000,    0.99720,   1000.000,    0.00222,   0.20000
     34,      508.000,    175.201,    400.000,  -300.000,    1.01230,   1000.000,    0.00140,   0.20000
     35,      650.000,    317.879,    800.000,  -500.000,    1.04930,   1000.000,    0.06150,   0.20000
     36,      560.000,    245.844,    800.000,  -500.000,    1.06350,   1000.000,    0.00268,   0.20000
     37,      540.000,     52.019,    800.000,  -500.000,    1.02780,   1000.000,    0.00686,   0.20000
     38,      830.000,    158.226,    800.000,  -500.000,    1.02650,   1000.000,    0.00300,   0.20000
     39,     1011.777,    342.702,   1500.000, -1000.000,    1.03000,   1000.000,    0.00100,   0.02000
0 / END OF GENERATOR DATA, BEGIN BRANCH DATA
```

C.1.3 Branch Data

```
     From Bus    To Bus    Line R     Line X     Charging
     Number      Number    (pu)       (pu)       (pu)
     1,          2,        0.00350,   0.04110,   0.69870
     1,          39,       0.00100,   0.02500,   0.75000
     2,          3,        0.00130,   0.01510,   0.25720
     2,          25,       0.00700,   0.00860,   0.14600
     3,          4,        0.00130,   0.02130,   0.22140
     3,          18,       0.00110,   0.01330,   0.21380
     4,          5,        0.00080,   0.01280,   0.13420
     4,          14,       0.00080,   0.01290,   0.13820
     5,          6,        0.00020,   0.00260,   0.04340
     5,          8,        0.00080,   0.01120,   0.14760
     6,          7,        0.00060,   0.00920,   0.11300
     6,          11,       0.00070,   0.00820,   0.13890
     7,          8,        0.00040,   0.00460,   0.07800
     8,          9,        0.00230,   0.03630,   0.38040
     9,          39,       0.00100,   0.02500,   1.20000
    10,          11,       0.00040,   0.00430,   0.07290
    10,          13,       0.00040,   0.00430,   0.07290
    13,          14,       0.00090,   0.01010,   0.17230
    14,          15,       0.00180,   0.02170,   0.36600
    15,          16,       0.00090,   0.00940,   0.17100
    16,          17,       0.00070,   0.00890,   0.13420
    16,          19,       0.00160,   0.01950,   0.30400
    16,          21,       0.00080,   0.01350,   0.25480
    16,          24,       0.00030,   0.00590,   0.06800
    17,          18,       0.00070,   0.00820,   0.13190
    17,          27,       0.00130,   0.01730,   0.32160
    21,          22,       0.00080,   0.01400,   0.25650
    22,          23,       0.00060,   0.00960,   0.18460
    23,          24,       0.00220,   0.03500,   0.36100
    25,          26,       0.00320,   0.03230,   0.51300
    26,          27,       0.00140,   0.01470,   0.23960
    26,          28,       0.00430,   0.04740,   0.78020
    26,          29,       0.00570,   0.06250,   1.02900
    28,          29,       0.00140,   0.01510,   0.24900
0 / END OF BRANCH DATA, BEGIN TRANSFORMER DATA
```

C.1.4 Transformator Data

```
     From Bus    To Bus     Specified R     Specified X     Winding
     Number      Number     (pu)            (pu)            (MVA)
     2,          30,        0.00000,        0.01810,        100.00
     6,          31,        0.00000,        0.02500,        100.00
    10,          32,        0.00000,        0.02000,        100.00
    11,          12,        0.00160,        0.04350,        100.00
    12,          13,        0.00160,        0.04350,        100.00
    19,          20,        0.00070,        0.01380,        100.00
    19,          33,        0.00070,        0.01420,        100.00
    20,          34,        0.00090,        0.01800,        100.00
    22,          35,        0.00000,        0.01430,        100.00
    23,          36,        0.00050,        0.02720,        100.00
    25,          37,        0.00060,        0.02320,        100.00
    29,          38,        0.00080,        0.01560,        100.00
0 / END OF TRANSFORMER DATA
```

C.1.5 Load-Flow Results

Bus Number	Code	G-Shunt	B-Shunt	Voltage (pu)	Angle (deg)
1,	1,	0.000,	0.000,	1.03297,	-9.3761
2,	1,	0.000,	0.000,	1.01107,	-6.5836
3,	1,	0.000,	0.000,	0.97373,	-9.6291
4,	1,	0.000,	0.000,	0.93270,	-10.4775
5,	1,	0.000,	0.000,	0.91852,	-9.0155
6,	1,	0.000,	0.000,	0.91880,	-8.1528
7,	1,	0.000,	0.000,	0.86315,	-10.6842
8,	1,	0.000,	0.000,	0.88084,	-11.4079
9,	1,	0.000,	0.000,	0.98072,	-11.2091
10,	1,	0.000,	0.000,	0.93851,	-5.4052
11,	1,	0.000,	0.000,	0.93050,	-6.3356
12,	1,	0.000,	0.000,	0.91230,	-6.3741
13,	1,	0.000,	0.000,	0.93620,	-6.2589
14,	1,	0.000,	0.000,	0.93615,	-8.2553
15,	1,	0.000,	0.000,	0.93910,	-8.7854
16,	1,	0.000,	0.000,	0.95674,	-7.1671
17,	1,	0.000,	0.000,	0.96640,	-8.3608
18,	1,	0.000,	0.000,	0.96767,	-9.3332
19,	1,	0.000,	0.000,	0.97919,	-1.8354
20,	1,	0.000,	0.000,	0.98066,	-3.2817
21,	1,	0.000,	0.000,	0.97306,	-4.4866
22,	1,	0.000,	0.000,	1.00987,	0.3478
23,	1,	0.000,	0.000,	1.00805,	0.1165
24,	1,	0.000,	0.000,	0.96783,	-7.0433
25,	1,	0.000,	0.000,	1.02018,	-5.1214
26,	1,	0.000,	0.000,	1.00002,	-6.3872
27,	1,	0.000,	0.000,	0.97808,	-8.5778
28,	1,	0.000,	0.000,	1.00181,	-2.5385
29,	1,	0.000,	0.000,	1.00379,	0.4750
30,	2,	0.000,	0.000,	1.04750,	-4.1349
31,	2,	0.000,	0.000,	1.04000,	0.3286
32,	2,	0.000,	0.000,	0.98310,	2.6947
33,	2,	0.000,	0.000,	0.99720,	3.3869
34,	2,	0.000,	0.000,	1.01230,	1.9120
35,	2,	0.000,	0.000,	1.04930,	5.3801
36,	2,	0.000,	0.000,	1.06350,	8.2185
37,	2,	0.000,	0.000,	1.02780,	1.7236
38,	2,	0.000,	0.000,	1.02650,	7.6230
39,	3,	0.000,	0.000,	1.03000,	-10.9600

C.2 Scale 2

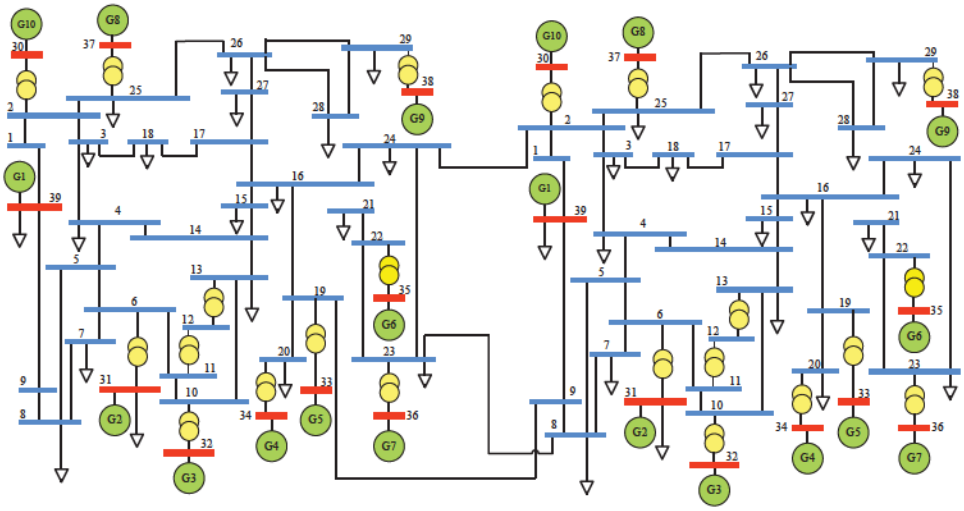


Figure C.2: Scale 2 system: 78 buses, 20 generators.

C.3 Scale 4

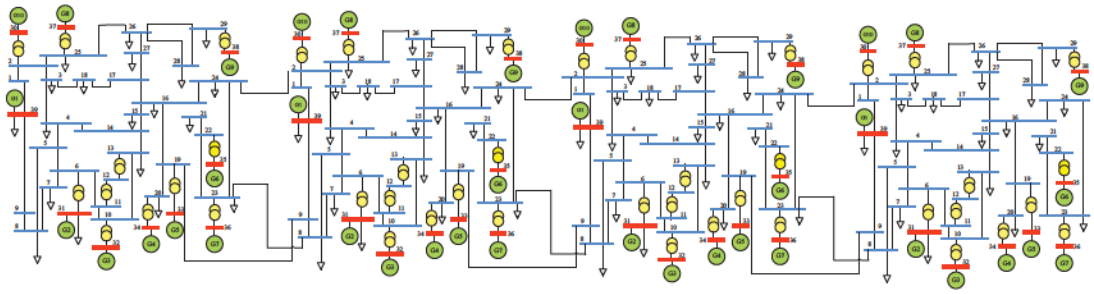


Figure C.3: Scale 4 system: 156 buses, 40 generators.

C.4 Scale 8

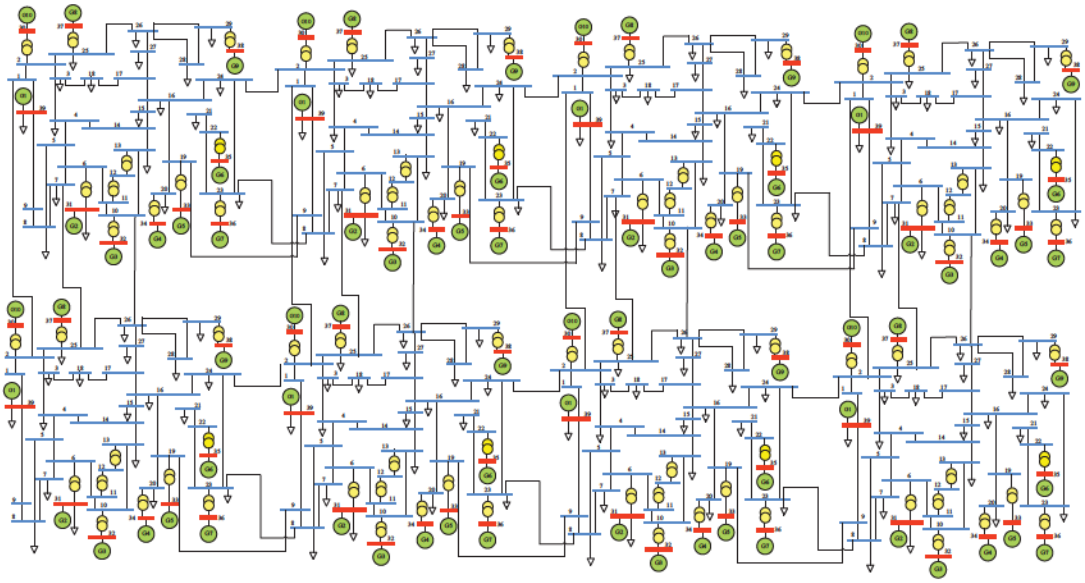


Figure C.4: Scale 8 system: 312 buses, 80 generators.

C.5 Scale 16

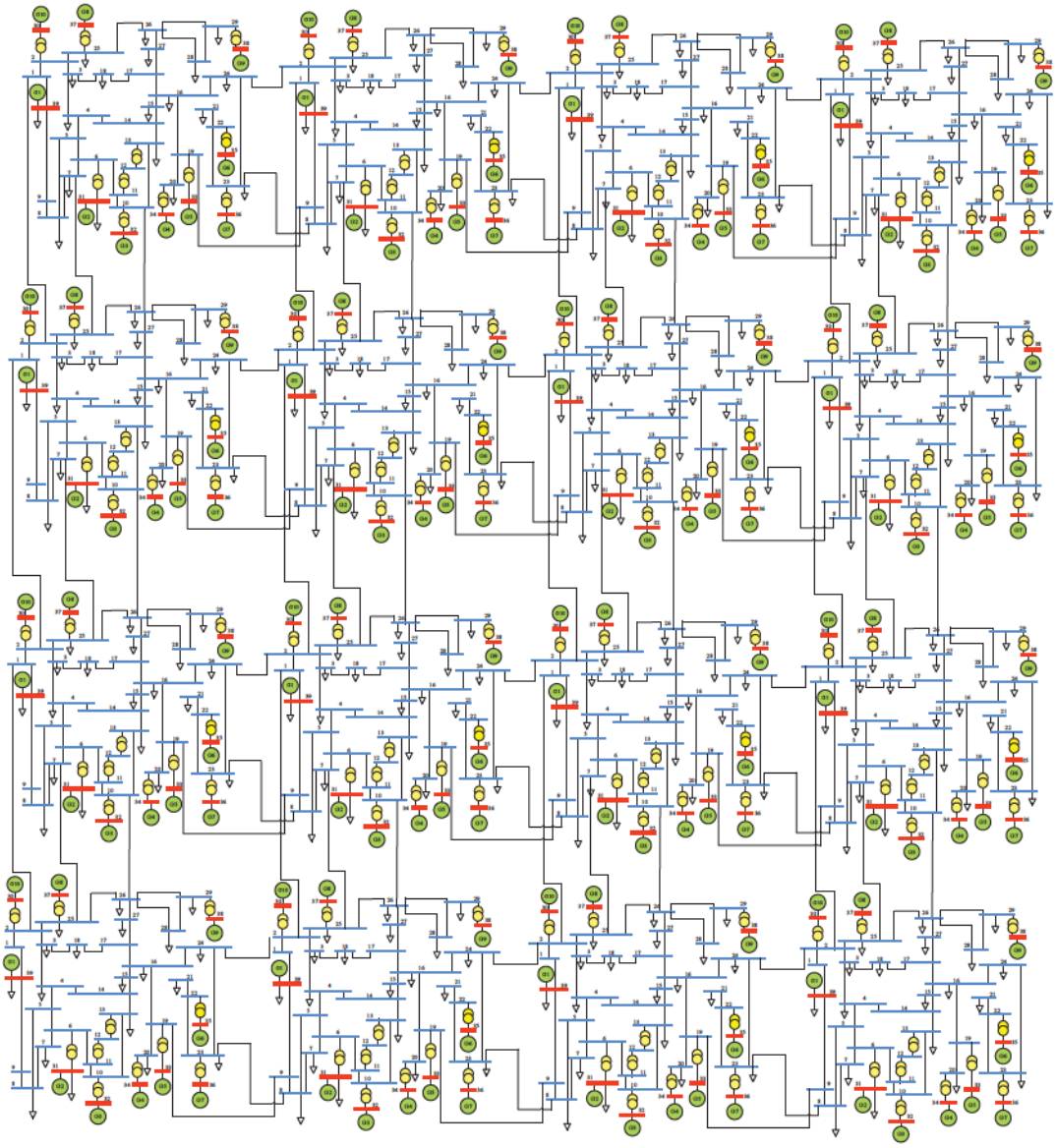


Figure C.5: Scale 16 system: 624 buses, 160 generators.

C.6 Scale 32

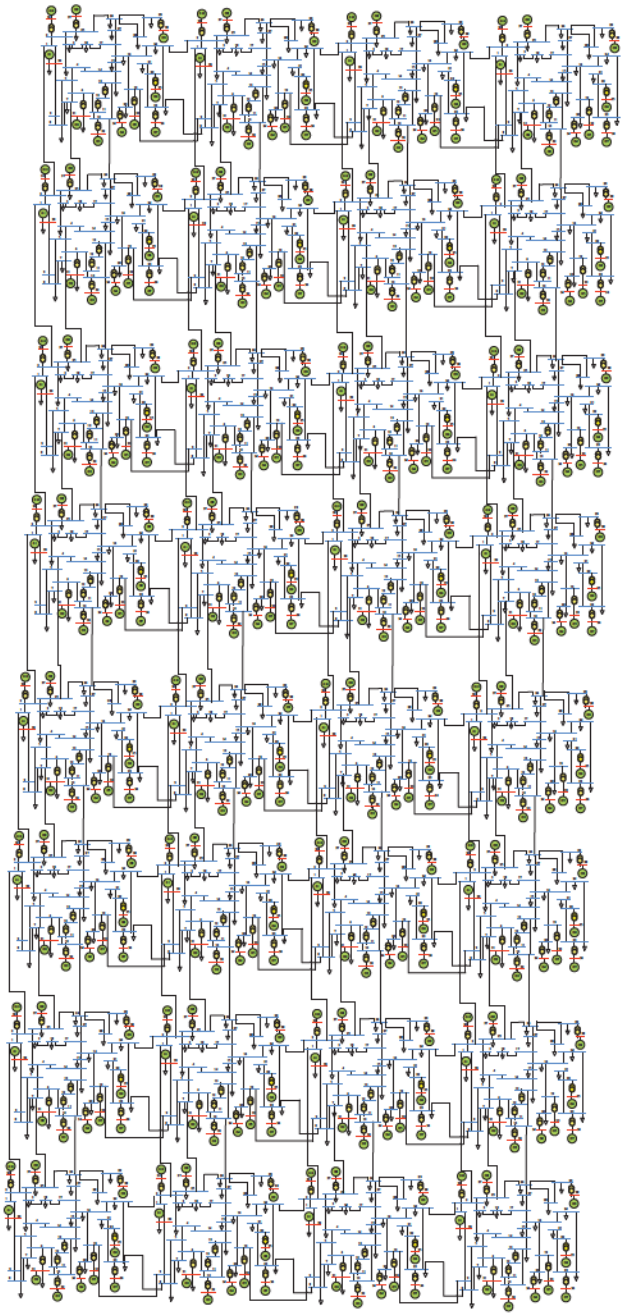


Figure C.6: Scale 32 system: 1248 buses, 320 generators.

C.7 Scale 64

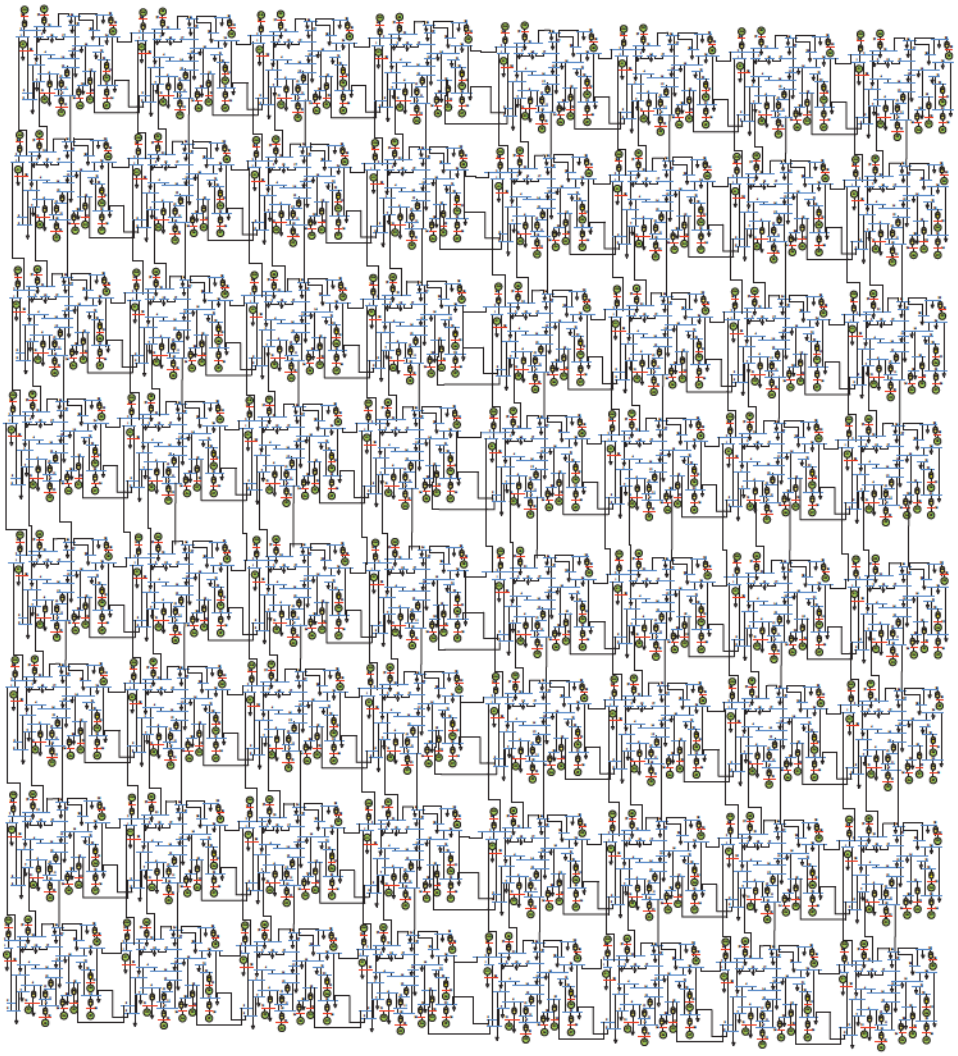


Figure C.7: Scale 64 system: 2496 buses, 640 generators.

C.8 Scale 128

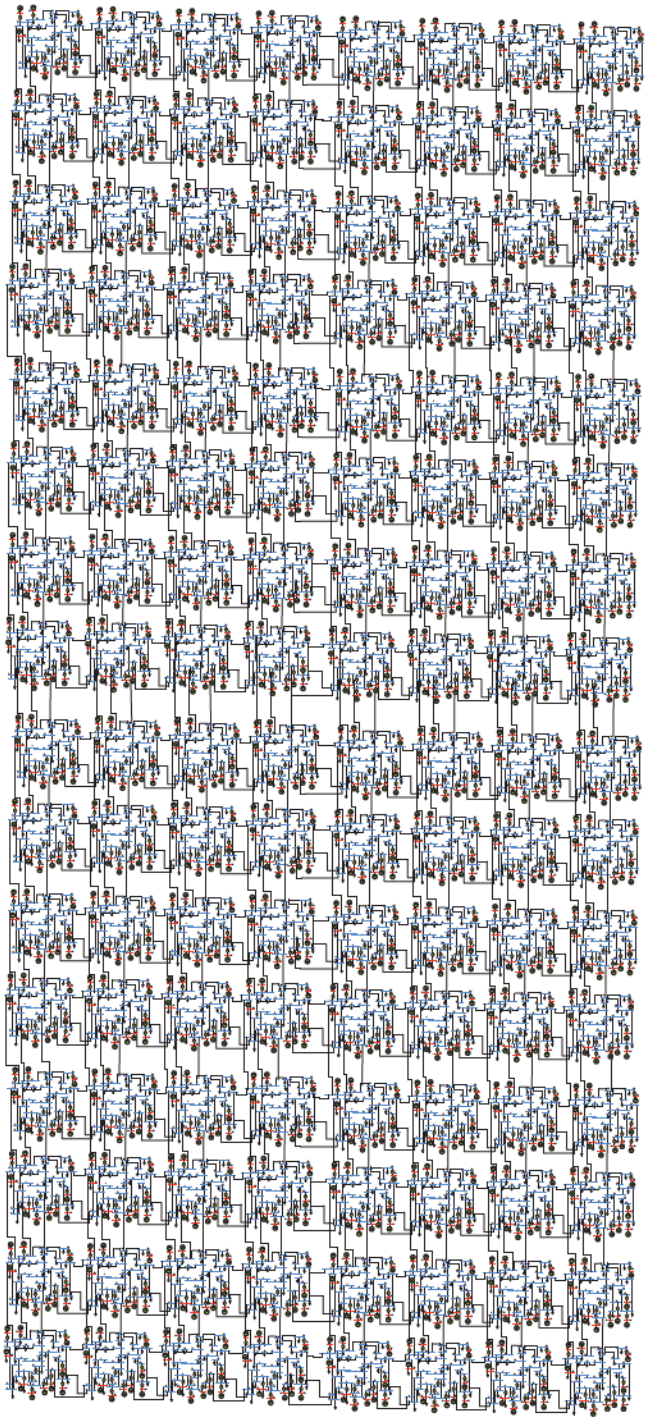


Figure C.8: Scale 128 system: 4992 buses, 1280 generators.